FABIO LOPES LICHT

AFINIDADE DE TIPOS DE APLICAÇÕES EM NUVENS COMPUTACIONAIS

Tese apresentada como requisito parcial à obtenção do grau de Doutor. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Bruno Schulze
Co-Orientador: Prof. Dr. Luis Carlos Erpen de Bona

CURITIBA

2014

FABIO LOPES LICHT

AFINIDADE DE TIPOS DE APLICAÇÕES EM NUVENS COMPUTACIONAIS

Tese apresentada como requisito parcial à obtenção do grau de Doutor. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Bruno Schulze
Co-Orientador: Prof. Dr. Luis Carlos Erpen de Bona

CURITIBA

2014

Licht, Fabio Lopes
Afinidade de tipos de aplicações em nuvens computacionais / Fabio Lopes Licht. – Curitiba, 2014.
153 f. : il., tabs., grafs.

Tese (doutorado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós Graduação em Informática
Orientador: Bruno Schulze
Coorientador: Luis Carlos Erpen de Bona
Bibliografia: p. 140-153

1. Afinidade. 2. Computação em nuvem. I. Schulze, Bruno. II. Bona, Luis Carlos Erpen de. III. Título.

CDD 005.74

FABIO LOPES LICHT

AFINIDADE DE TIPOS DE APLICAÇÕES EM NUVENS COMPUTACIONAIS

Tese aprovada como requisito parcial à obtenção do grau de Doutor no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comisso formada pelos professores:

Orientador: Prof. Dr. Bruno Schulze
Departamento de Informática, UFPR

Co-Orientador: Prof. Dr. Luis Carlos Erpen de Bona
Departamento de Informática, UFPR

Prof. Dr. Antônio Roberto Mury
Coordenação de Ciência da Computação, LNCC

Prof. Dr. Daniel Weingaertner
Departamento de Informática, UFPR

Prof. Dr. Fabiano Silva
Departamento de Informática, UFPR

Prof. Dr. Jose Neuman de Souza
Departamento de Computação, UFC

Curitiba, 2014

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa do aluno de Doutorado em Ciência da Computação, Fabio Lopes Licht, avaliamos a tese de doutorado intitulada *"Afinidade de tipos de aplicações em nuvens computacionais"*, cuja defesa pública foi realizada no dia 24 de setembro de 2014, às 09:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após avaliação, decidimos pela:

(X) **aprovação** do candidato. ( ) **reprovação** do candidato.

Curitiba, 24 de setembro de 2014.

Prof. Dr. Bruno Richard Schulze
LNCC – Orientador

Prof. Dr. Luis Carlos Erpen de Bona
DINF/UFPR – Coorientador

Prof. Dr. José Neuman Souza
UFC – Membro Externo

Prof. Dr. Antonio Roberto Mury
LNCC – Membro Externo

Prof. Dr. Fabiano Silva
DINF/UFPR – Membro Interno

Prof. Dr. Daniel Weingaertner
DINF/UFPR – Membro Interno

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

ATA DA DEFESA DE TESE DE DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO DO ALUNO: FABIO LOPES LICHT

No dia 24 de setembro do ano de dois mil e quatorze, às 09:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná, foi realizada a sessão pública da defesa de Tese de Doutorado em Ciência da Computação do aluno Fabio Lopes Licht. Estavam presentes, além do candidato, os Membros da Comissão Examinadora composta pelos Professores Luis Carlos Erpen de Bona(Orientador), Bruno Richard Schulze, José Neuman Souza, Antonio Roberto Mury, Fabiano Silva e Daniel Weingaertner. Após a apresentação do trabalho do candidato, intitulado *"Afinidade de tipos de aplicações em nuvens computacionais"*, o mesmo foi arguido pela Comissão. A seguir, a Comissão reuniu-se em local reservado e decidiu, por unanimidade, pela aprovação do candidato, condicionada as alterações sugeridas pela mesma. O resultado foi então comunicado ao candidato e aos presentes na sessão pública. A seguir, o Presidente declarou encerrada a sessão da qual eu, Jucélia Miecznikowski, Secretária do Programa de Pós-graduação em Informática, lavrei a presente Ata, que depois de aprovada será assinada por mim, pelo Presidente, e pelos demais membros da Comissão.

Prof. Dr. Bruno Richard Schulze
LNCC – Orientador

Prof. Dr. Luis Carlos Erpen de Bona
DINF/UFPR – Coorientador

Prof. Dr. José Neuman Souza
UFC – Membro Externo

Prof. Dr. Antonio Roberto Mury
LNCC – Membro Externo

Prof. Dr. Fabiano Silva
DINF/UFPR – Membro Interno

Prof. Dr. Daniel Weingaertner
DINF/UFPR – Membro Interno

Jucélia Miecznikowski
Secretária da PPGInf

AGRADECIMENTOS

Agradeço primeiramente a Deus por olhar por mim. Agradeço à minha esposa e filha que sempre estiveram a meu lado, à minha família e amigos. Um agradecimento especial a meu orientador e amigo Bruno Schulze por ter me auxiliado e orientado em todo meu caminho acadêmico, desde a graduação até o doutorado, a meu grande amigo Roberto pela ajuda e apoio em todas as fases do doutorado, por ter ouvido e aconselhado. Ao meu co-orientador Bona por ter me acolhido na UFPR e me dado a chance de chegar até aqui, ao CNPQ por ter financiado este sonho e a todos os que de alguma maneira estiveram presentes nesta conquista, meus sinceros agradecimentos.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O aumento do uso de ambientes virtualizados tem levado à inúmeras pesquisas sobre as possibilidades e restrições de seu uso na computação em nuvem ou para consolidação de recursos. Entretanto, a maioria destes estudos são limitados a um nível de análise de desempenho, que não aprofunda os efeitos da concorrência entre os vários ambientes virtuais, e como mitigar esses efeitos. O estudo apresentado a seguir propõe o conceito de afinidade, que define o grau de coexistência entre as classes de aplicações. Estas combinações têm como elementos que as influenciam: as classes e subclasses de algoritmos utilizados na implementação destas aplicações, associadas aos tipos de bibliotecas paralelas por elas utilizadas. Os resultados obtidos nesta pesquisa demonstram que os efeitos destas combinações entre classes de algoritmos e bibliotecas de paralelização têm valores tão diversos, que torna necessário o estudo e mensuração destes valores detalhadamente, justificando a proposta aqui apresentada quanto a definição e análise do conceito afinidade, buscando com isso contribuir para um melhor uso dos recursos, sobretudo no que tange à computação massivamente paralela e distribuída, com impactos tanto na elaboração de novas aplicações, quanto na elaboração de novos escalonadores para estes ambientes.

Palavras chave: afinidade, classes de aplicações, computação distribuída, dwarfs, concorrência, gerência, nuvem, virtualização.

ABSTRACT

The increased use of virtual environments has motivated extensive research on the possibilities and limitations of its use in cloud computing or resource consolidation. However the majority of these studies are limited to a degree of performance analysis, which does not deepens the effects of concurrency between multiple virtual environments, and how to mitigate these effects. The study presented below proposes the concept of affinity, that defines the degree of coexistence between classes of applications. These combinations have as elements that influence them: classes and subclasses of algorithms used in the implementation of these applications, associated with the types of parallel libraries used by them. The results obtained in this research show that the effects of these combinations between classes of algorithms and parallelization libraries have different values, which makes the study and measuring of these values in detail, justifying the proposal presented here as well as the definition and analysis of the affinity concept, looking thereby contribute to a better use of resources, especially regarding to the massively parallel and distributed computing, with impacts both in developing of new applications, as the development of new schedulers for these environments. Keywords: affinity, applications classes, distributed computing, dwarfs, concurrency, management, cloud computing, virtualization.

CAPÍTULO 1

INTRODUÇÃO

A preocupação crescente na qualidade dos serviços prestados por nuvens computacionais tem levado pesquisadores a buscarem mecanismos e metodologias que analisem e auxiliem na alocação de aplicações nos recursos computacionais. Neste sentido, conhecer como interagem as aplicações, os ambientes virtualizados e os efeitos causados pela concorrência nestes recursos contribui para esses esforços no sentido de minimizar as perdas causadas.

Considerando que tanto em ambientes de nuvens computacionais quanto consolidados em um único recurso físico pode haver vários ambientes virtualizados, um processo sendo executado em um destes ambientes virtuais poderia causar a degradação do recurso físico, e consequentemente, impactar em todos os outros ambientes existentes em uma mesma máquina "hospedeira"[1].

O impacto causado por esta concorrência entre processos sendo executados em mais de um servidor virtualizado, hospedados em um mesmo servidor real precisa, assim, ser avaliado afim de criar um conhecimento dos tipos de concorrência que podem existir, sem que uma aplicação prejudique o tempo de execução de outra por concorrerem pelos mesmos recursos, principalmente com respeito ao uso do processador por mais de uma aplicação.

Já com respeito às nuvens computacionais e seu uso como apoio à computação massivamente paralela e distribuída, especificamente quando sob a ótica das aplicações científicas, os estudos dedicados a verificar qual o efeito da camada de virtualização sobre o desempenho das aplicações, também deixam uma lacuna quanto aos efeitos da concorrência entre

[1] Máquina "real" na qual ficam alocadas, ou sendo executandas máquinas virtuais.

os diversos ambientes virtualizados hospedados em um mesmo recurso real.

Sobre a avaliação do provisionamento de aplicações, Upendra Sharma et al. [77] verificam o desempenho de dois tipos de nuvem. Os autores afirmam que com um "consumo consciente"é possível ter uma "economia"da infraestrutura. Neste sentido, se for possível avaliar o efeito da concorrência entre aplicações em um mesmo recurso físico, será possível predizer em que servidores uma aplicação conhecida previamente pode ser executada, sem prejuízo das demais aplicações, e assim, pode-se propor um "consumo consciente"e uma afinidade entre estas aplicações.

Um exemplo prático deste problema seria a degradação do servidor causada pela alocação de uma aplicação em uma máquina virtual, que viria a consumir toda a capacidade de processamento deste servidor como é apresentado em [93]. Neste caso um outro processo em uma máquina virtual diferente, mas alocada no mesmo servidor real teria que competir pelo uso de recursos físicos (memória, processador, etc.), o que poderia consequentemente prejudicar o seu processamento e desempenho, acarretando perdas que inviabilizariam o compartilhamento destes recursos físicos, neste caso, resta observar e predizer que tipos de concorrências não seriam possíveis. Foco do trabalho aqui proposto.

Já com respeito à disponibilidade, há neste trabalho a preocupação em contribuir com a análise da infraestrutura, que não só atenda à execução das aplicações, mas que também reduza o tempo de execução ou mesmo a indisponibilidade, o que acarretaria não somente na perda da confiabilidade, mas também na perda de dados de execução que criariam, nesse caso, um gargalo de execução de tarefas no ambiente, por efeito do retrabalho de geração dos novos dados, por exemplo.

Neste sentido, prever a degradação, pode ser a melhor maneira de criar escalonadores que, em caso de perda de desempenho, previamente conhecida, transfiram o processamento para recursos menos susceptíveis às falhas ou atrasos causados por uso. Assim,

o trabalho aqui proposto faria uma previsão de perda de desempenho pela concorrência entre tipos de aplicações "conhecidas"definidas com a abordagem dos tipos de aplicações e uso dos Dwarfs, que são apresentados na Seção 3.1.

Este estudo se motiva na medida em que as nuvens computacionais surgem como recursos extras, disponíveis sob demanda, capazes de suprir necessidades de recursos computacionais para a computação massivamente paralela e distribuída. Entretanto poucos trabalhos têm se dedicado a verificar em profundidade os efeitos que a camada de virtualização tem sob as aplicações em termos de classes, o que em muito contribuiria, tanto no uso mais efetivo destes recursos virtualizados, como no desenvolvimento de escalonadores, capazes de identificar que conjunto de aplicações poderiam se beneficiar com as nuvens computacionais e que conjunto de aplicações trariam perdas em termos de desempenho.

No que tange a qualidade de serviços, a análise deste tipo de interação entre os ambientes virtualizados contribui para reduzir a variabilidade observada em função desta concorrência. Nedeljko Vasic et al. [99] apresentam que a gestão eficaz dos recursos de ambientes virtualizados é uma tarefa desafiadora. Os autores consideram que os sistemas de gestão, precisam contar com modelos analíticos ou avaliar a alocação de recursos, executando experimentos reais.

Faz-se assim necessário, no contexto do trabalho, estabelecer quais conjuntos de aplicações hospedadas nestes ambientes virtualizados que podem coexistir em um mesmo ambiente real, assim como determinar quais aquelas combinações de aplicações que devem ser evitadas. Nesse sentido, este trabalho estabelece o conceito de "Afinidade", que pode ser utilizado tanto na concorrência para ambientes reais quanto virtualizados, entretanto, como foco do trabalho, já que as nuvens computacionais se utilizam da virtualização de ambientes, avalia o efeito da concorrência nas aplicações sendo executadas em servidores virtualizados.

Mesmo assim os efeitos nos servidores reais são também apresentados como informações e para demonstrar que estas aplicações sendo executadas concorrentemente também influenciam nestes ambientes. Assim, de posse destes dados, será possível definir que há tipos de aplicações que possuem maior influência do meio em que são executadas, e pode-se verificar que certos tipos de aplicação são melhor executados em ambientes reais, por exemplo, devendo ser evitadas em ambientes virtualizados, o que traria uma informação de que este tipo de aplicação não deve ser "portada"para nuvens computacionais.

Para definir o trabalho, propõem-se o uso do termo "Afinidade". Pode-se definir afinidade como: "Uma sintonia de conjunções ou instinto e sentimento que estabelece vínculos entre seres, e elementos da natureza.", ou ainda "Conformidade, aproximação, relação e simpatia". No contexto deste trabalho, afinidade aparece como conceitos similares a estes no sentido em que se procura com este termo, definir tipos de aplicações que são afins, ou seja, tipos de aplicações que podem coexistir, sem que uma aplicação ou tipo de aplicação prejudique a outra em sua execução concorrente pelos mesmos recursos físicos.

Outra definição permitiria definir que tipos de aplicações hospedadas em servidores virtuais são influenciadas por outras aplicações hospedadas em outros servidores virtuais, sendo que estes servidores são executados em um mesmo servidor real ("Máquina hospedeira").

O conceito de "Afinidade"que é proposto neste trabalho, e que será melhor detalhado na Seção 3.3 teve como propósito, mostrar quais os tipos de aplicações que podem conviver em um mesmo ambiente físico, em máquinas virtualizadas ou reais, sem que este tipo de compartilhamento de recursos venha a causar perda de desempenho que prejudique a execução das aplicações lá hospedadas.

Além de considerar afinidade como o efeito da concorrência entre aplicações que compartilham o mesmo recurso físico, propõe-se o termo para definir o impacto das bibliotecas

de programação paralela utilizadas nos testes (OpenCL e OpenMP).

Considera-se nesta tese que o desempenho é medido por tempo de execução da aplicação, desde o momento em que começa a ser executada, até o ponto em que retorna o resultado final do processamento, sem considerar o tempo de acesso aos dados, avaliando somente o tempo de processamento com uso de cronômetro e não com uso de tempo de processador.

Até a presente data não foram encontrados trabalhos que relacionassem afinidade ao efeito da concorrência, o conceito mais próximo do proposto neste trabalho é apresentado por Vignesh T. Ravi et. al [69]. O artigo apresenta os resultados da execução de aplicações dentro de máquinas virtuais em uma ou mais GPUs, buscando a virtualização destas GPUs eficientemente. Os autores introduzem um método para calcular o índice de afinidade entre dois ou mais núcleos, o que fornece uma indicação de potenciais melhorias de desempenho no processo de consolidação do kernel. O trabalho dos autores em definir que há "afinidade entre GPUs"levou à utilização deste termo nesta tese.

No que compete às aplicações, já existem estudos que procuraram categorizá-las em classes, agrupando-as com base em suas características em termos de consumo de recursos computacionais como é avaliado em [39], [50] e [66]. Este tipo de categorização permite que as novas aplicações desenvolvidas possam ser associadas a uma dessas classes trazendo como benefício a possibilidade de se poder prever o seu comportamento quando executada em um ambiente virtualizado.

O trabalho aqui apresentado se utiliza de uma das abordagens apresentadas nestes estudos, e que serão detalhadas na Seção 3.4, para obter resultados que comprovem a "Afinidade"entre as aplicações. Foram escolhidas para este trabalho três classes de aplicações e quatro algoritmos dos Dwarfs. Para uma das classes escolhidas foram utilizados dois algoritmos por serem de diferentes domínios de aplicações dentro de uma mesma classe. As classes escolhidas representam os seguintes tipos de aplicações:

1. Álgebra Linear Densa - Possui muita influência em aplicações de Sistemas Embarcados, SPEC, Games, ML, HPC, Imagens, Speech e Música. Possui baixa influência em aplicações de Banco de Dados, Algoritmos de Refinamento Delaunay, Clusterização Kmeans. Neste tipo de aplicação foram testados os seguintes algoritmos:
   - LUD - Aplicação de álgebra linear;
   - Kmeans - Aplicação de mineração de dados.
2. Grade Estruturada - Possui muita influência em aplicações de Sistemas Embarcados, SPEC, HPC e Processamento de Imagens. Possui baixa influência em games. Neste tipo de aplicação foi testado o seguinte algoritmo:
   - SRAD - Aplicação de Processamento de Imagens.
3. Grafo Transversal - Possui muita influência em aplicações de Sistemas Embarcados, ML, Saúde, Speech e Sistemas de Reserva. Possui baixa influência em SPEC, Banco de Dados, Games, Música e Sequenciamento de Genes.
   - B+Tree - Aplicação de busca.

Estas três classes foram também escolhidas porque, juntas, oferecem um conjunto diversificado de padrões, que compreendem um conjunto mais completo de experimentos, como é apresentado na Seção 3.1.

O uso desta abordagem se apresentou adequada já que a proposta do trabalho é a execução dos algoritmos acima definidos, concorrentemente, analisando a degradação causada quando um tipo de aplicação compete pelos mesmos recursos com outro. Sugere-se assim que, tomando como base o tempo de execução sem concorrência, que se este tempo aumenta, houve uma degradação de um algoritmo causada por outro algoritmo.

Serão feitos assim testes de desempenho individuais para serem usados como linha de base (tempo de execução individual) e novamente executando par a par todas as possíveis combinações de algoritmos para verificar o efeito da concorrência destes, quando precisam "disputar"pelo uso dos mesmos recursos. Como medida será utilizada a "hora relógio"com tempo de execução final sem considerar o acesso a dados.

Para definir a perda de desempenho em relação à não concorrência tomou-se o termo "grau"[2] de afinidade entre as classes, os testes realizados analisaram os efeitos que o uso de ambientes virtualizados, concorrendo pelos mesmos recursos reais, têm no desempenho dessas classes de aplicações, tendo por objetivo determinar quais classes que poderiam ser consolidadas em um mesmo recurso real e quais classes que acarretariam uma perda de desempenho que comprometeria o seu uso.

Para esta avaliação são feitos testes que apresentam dois tipos de resultados, média de desempenho (perda percentual) e distância entre perdas (estabilidade do ambiente), como proposto por Jörg Schad et al. [72]. No trabalho os autores apresentam um modelo de cálculo da variação de desempenho com recursos coletados através do uso de benchmarks e citam que, em geral, para esse tipo de análise são usadas fórmulas estatísticas como faixas, desvio-padrão e outros.

No contexto deste trabalho são usadas as fórmulas de intervalo de confiança, que permitiu definir a quantidade de estes necessária para obter o valor adequado de amostras (O resultado definiu que 20 amostras de cada teste com todas as combinações eram suficientes). A média aritmética também foi utilizada, bem como a normalização de valores usando a fórmula de normalização proposta na Seção 4.6 e publicada em [30].

Analisou-se ainda nesta tese, com respeito à implementação das aplicações, que também há variações quanto aos tipos de bibliotecas utilizadas, principalmente no tocante às

[2]Valoração quantitativa.

aplicações massivamente paralelas e distribuídas. Neste sentido, este trabalho apresenta resultados obtidos dos testes realizados com dois tipos de bibliotecas de programação paralela, os resultados contribuem na escolha do melhor tipo de biblioteca para implementação de algoritmos em ambientes virtuais ou reais, bem como na execução concorrente destes, ou seja, quando implementados com uma mesma biblioteca (a esta combinação propõe-se chamar de "concorrência homogênea") ou implementando cada algoritmo com uma biblioteca diferente (a esta combinação propõe-se chamar de "concorrência heterogênea").

Os termos concorrência homogênea e heterogênea são propostos neste trabalho por apresentarem conceitos relevantes ao proposto e por considerar que no trabalho foram utilizados dois tipos de bibliotecas paralelas, OpenCL e OpenMP. Tomou-se "Homogênea"[3] como associação ao uso do mesmo tipo de biblioteca, e que neste caso houve concorrência entre algoritmos implementados com a mesma biblioteca (OpenCL ou OpenMP). O uso do termo "Heterogênea"[4] se deu com a concorrência entre bibliotecas diferentes, sendo um algoritmo implementado com OpenCL e outro com OpenMP, competindo pelos mesmos recursos. Os termos se apresentaram eficientes para representar os conceitos necessários.

1.1 Definição do Problema

Quando há concorrência entre aplicações é possível perceber que alguns tipos destas aplicações interferem no desempenho de outras. Essa degradação é percebida mesmo quando a concorrência é feita em servidores virtuais distintos, desde que sendo executados em um mesmo servidor real (mesmo servidor hospedando estes servidores virtuais).

A análise permite definir que a concorrência pode levar a perdas de desempenho que podem ser consideradas prejudiciais, quando o tempo de execução final de uma aplicação é elevado a um "nível"intolerável, acarretando na perda da qualidade dos serviços pres-

[3]composto de partes ou elementos que são todos da mesma natureza
[4]composto de partes ou elementos de natureza diferente

tados por nuvens computacionais. A este "nível"pode-se propor o uso do termo "grau de degradação". O grau neste caso será considerado como a perda de desempenho de uma aplicação quando executada concorrentemente com outra.

O estudo e resultados obtidos são considerados importantes já que em grandes nuvens computacionais, como a Amazon [105] [76] o serviço inclui o fornecimento de servidores virtuais de acordo com a demanda do cliente, mas não há garantias de que este servidor está sendo executado em um recurso real livre de concorrência, por exemplo.

Considerando a proposta das nuvens computacionais, a própria definição leva à concorrência entre diversas aplicações e servidores virtuais, já que se a proposta fosse de fornecer um servidor real para cada serviço contratado, além do custo monetário, haveria uma quebra do paradigma principal das nuvens, de fornecer recursos sob demanda. Já que seria necessário ter sempre n+1 servidores disponíveis, considerando n o número de pedidos.

O trabalho aqui proposto é tão importante que Kambadur et al. [40] citam a necessidade de avaliar o impacto da concorrência entre aplicações, definindo que o trabalho dos autores é o precursor da análise de nuvens e do impacto da interferência quando há muitos acessos à datacenters.

Dwyer et al. [24] também define a importância da avaliação dos tipos de aplicações. Os autores apresentam um estudo com aprendizagem de máquina para avaliar o impacto de aplicações em ambientes sem prioridade e citam que há necessidade de conhecer os tipos de aplicações que são executadas para que haja uma previsão de uso destes ambientes.

Em função do anteriormente exposto, é necessário analisar, com base na categorização das aplicações, quais classes de aplicações podem compartilhar os mesmos ambientes reais e virtuais, focando nos virtuais por serem a base das nuvens computacionais, minimi-

zando assim a perda de desempenho causada pela concorrência e definindo quais classes de aplicações cujo compartilhamento poderia comprometer significativamente o desempenho (tempo de execução), assim como verificar os efeitos das bibliotecas utilizadas na implementação destas classes.

Para definir "Tipos de aplicações"foi utilizado o conceito de *Dwarfs* descrito na Seção 3.1 que categoriza as aplicações e identifica o tipo de algoritmo que pode ser usado para uma análise de desempenho. A verificação do grau de afinidade por meio dos Dwarfs permite determinar que aplicações podem coexistir e quais não devem coexistir em função da classe a que pertença.

A o uso pela definição dos Dwarfs garante que se um tipo de aplicação possui um comportamento em um servidor, o mesmo comportamento é esperado por todos os algoritmos que pertençam à mesma classe de Dwarf, e, logo, se a concorrência entre dois tipos destes Dwarfs influencia no tempo de execução, o mesmo é esperado em todas as concorrências das mesmas classes.

Objetivo do trabalho proposto é então o de permitir, com o estudo, a criação de escalonadores de tarefas que, dada uma classificação do tipo de programa, possa encontrar a melhor combinação de forma a otimizar o seu desempenho.

O trabalho aqui apresentado procura então preencher uma lacuna quanto à análise de desempenho em ambientes virtualizados no que tange a concorrência pelo uso dos recursos, contribuindo para o uso de ambientes de nuvens computacionais. Podendo seus resultados serem utilizados para o desenvolvimento de algoritmos para criação de novos escalonadores capazes de atender às características específicas destes ambientes, sobretudo no tocante ao uso de nuvens computacionais como recursos adicionais à computação massivamente paralela e distribuída.

1.2 Delimitação do Trabalho

Para definir que tipos de aplicações podem concorrer por um mesmo recurso físico, sem que uma aplicação cause atraso na execução de outra, ou seja, sem que uma degrade o funcionamento de outra, propõem-se o uso de classes de aplicações e o conceito destas classes definida com o uso dos Dwarfs, cujos conceitos serão apresentados na Seção 3.1.

Para avaliar os efeitos da concorrência, este trabalho analisou dentre as 13 classes de Dwarfs existentes as 3 classes que abrangem, em maior ou menor grau, todas as áreas de aplicações científicas. Também analisou os efeitos de duas bibliotecas utilizadas para implementação de aplicações paralelas (OpenCL e OpenMP).

O uso de OpenCL (*Open Computing Language*) se deu dado que esta biblioteca fornece APIs que são usadas para definir e controlar as plataformas heterogêneas tanto de CPUs, quanto de GPUs além de permitir programação paralela usando paralelismo de tarefas e de dados.

O uso de OpenMP (*Open Multi-Processing*) se deu dado que esta biblioteca fornece APIs para a programação multiprocessamento de memória compartilhada baseada em threads para múltiplas plataformas em um modelo portável e escalável que possui um paralelismo explícito e usa sincronização para evitar competição entre threads.

É importante definir que não faz parte do trabalho a criação dos escalonadores e sim, com o conhecimento gerado, criar mecanismos de testes e analisadores de desempenho para caracterizar a afinidade entre hardware e software com o uso do conceito de tipos de aplicações.

Também é importante definir que foram utilizados para propor o conceito de afinidade entre aplicações e o efeito destas quando são executadas em um mesmo recurso físico, três classes de algoritmos (Álgebra Linear Densa, Grafo Transversal e Grade Estruturada) e

dois tipos de bibliotecas de programação paralela (OpenCL e OpenMP).

1.3 Organização do Trabalho

Este trabalho está organizado da seguinte forma: No Capítulo 1 é apresentada a introdução, na Seção 1.1 é apresentado o problema tratado por esta tese e na Seção 1.2 a delimitação do tema, no Capítulo 2 são apresentadas as tecnologias envolvidas e os conceitos iniciais para pleno entendimento dos capítulos seguintes. No Capítulo 3 são apresentadas as considerações iniciais da pesquisa, na Seção 3.1 são apresentados os conceitos de Dwarfs, na Seção 3.2 é apresentado o benchmark Rodinia, na Seção 3.3 as definições de afinidade da proposta e na Seção 3.4 são apresentados os questionamentos e as hipóteses, na Seção 4.3 as justificativas do estudo e na Seção 4.4 as metodologias e infraestruturas utilizadas. No Capítulo 5 são apresentados todos os resultados obtidos, bem como os comparativos. No Capítulo 6 são apresentadas as conclusões, contribuições e considerações finais e finalmente na Seção 6.2 as sugestões de trabalhos futuros.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta as principais tecnologias envolvidas no trabalho, os conceitos e definições dos termos utilizados na pesquisa, buscando informar os conceitos relevantes ao entendimento dos próximos capítulos.

2.1 Tecnologias Envolvidas

Na Seção 2.1.1 é apresentado o conceito de computação em nuvem. As Características Essenciais de uma Nuvem são apresentadas na Seção 2.1.2 e na Seção 2.1.4 são apresentados os modelos de nuvem. Na Seção 2.1.6 descreve-se a computação de alto desempenho e na Seção 2.2 o conceito e os tipos de virtualização. Por fim, na Seção 2.2.4 apresentam-se as principais diferenças entre aplicações comerciais e científicas, seguido do conceito de dependabilidade na Seção 2.2.5.

2.1.1 Computação em Nuvem

Computação em Nuvem [102], do inglês *Cloud Computing* é a tecnologia que tem como objetivo proporcionar serviços de computação sob demanda, com pagamento de acordo com o uso. Pode-se dizer que os precursores da computação na nuvem foram os Clusters e Grades Computacionais [3]. Manjula e Karthikeyan [52] apresentam um comparativo entre nuvens computacionais e grades. No trabalho há uma apresentação dos modelos usados pelas duas infraestruturas e os comportamentos delas.

A diferença entre a nuvem e seus antecessores (grades computacionais e clusters) é que nesta, os serviços não estão vinculados a um grupo específico de usuários, nem precisam estar sob o mesmo domínio administrativo. Qualquer computador, tablet ou celular, pode

fazer parte da nuvem e interagir com ela. Como estes recursos normalmente são compostos de centenas ou milhares de máquinas, físicas ou virtuais de baixo custo, o crescimento deste tipo de infraestrutura se torna mais fácil e rápido.

Esta computação na nuvem surge de uma necessidade de aumentar os recursos computacionais, sem ter que necessariamente saber onde estes estão, ou mesmo ter que instalar sistemas operacionais, atualizações ou softwares. Necessitando apenas que o serviço seja disponibilizado para dar acesso aos recursos deste e por ser um ambiente heterogêneo, basta ter mais recursos disponíveis que estes podem ser agregados aos já existentes, aumentando a capacidade de processamento ou armazenamento rapidamente [98].

Uma das propostas da nuvem é que em um curto espaço de tempo, ninguém precise mais ter que montar infraestruturas de hardware e software para testar aplicações ou serviços sob demanda que possuam uma curta atividade ou mesmo para verificar a viabilidade do uso. Ao invés disso, alugaria-se o serviço disponível pelo tempo necessário a um custo menor em relação à compra dos mesmos recursos que poderiam não atender às expectativas. Na falta de recurso, por estouro de sua capacidade, acrescentaria-se mais recursos aos já disponíveis.

A proposta de ser global e prover serviços para qualquer usuário, que vão desde uma hospedagem de documentos pessoais à terceirização de toda uma infraestrutura de TI, permite que este serviço alcance qualquer usuário, quando a capacidade de armazenamento é degradada por excesso de informação ou danificação dos recursos de armazenamento, basta que se localize a replicação dos dados ou aumente os recursos disponíveis que o serviço ficaria novamente disponível. Esta infraestrutura é provida como um serviço e, estes são normalmente alocados através de centros de dados, utilizando hardware compartilhado para computação e armazenamento[13].

O Artigo de Zhang et al. [107] apresenta os principais conceitos de computação na

nuvem e mostra como a computação em nuvem pode ser atraente para os empresários, uma vez que elimina a necessidade de grande planejamento antecipadamente, já que permite que as empresas comecem com recursos reduzidos e podem aumentar de acordo com a demanda.

Outro artigo que trata da computação na nuvem e cita os desafios e problemas deste tipo de tecnologia considerada pelo autor como emergente é apresentado por Asma[6] et al. No artigo os autores descrevem a computação em nuvem como a rede do futuro.

Apresentam ainda uma visão geral de alguns trabalhos nesta área e alguns conceitos de como serviços na nuvem, e arquitetura orientada a serviços são projetados para facilitar a prototipagem rápida e implantação sob demanda, aumentando a flexibilidade, o desempenho da comunicação, a robustez e a escalabilidade. Por fim eles expõem o conceito de SOA[1] e virtualização.

Rajkumar Buyya et al. [12] apresentam uma visão detalhada do crescimento do uso da computação na nuvem mostrando as estratégias de mercado para gestão de serviços orientados aos clientes, como gestão de risco e SLA's[2][48].

Já o artigo de Yogesh Simmhan et al.[81] cita a complexidade na criação de aplicativos, invocação de aplicações em nuvem a partir de desktops clientes e transferência eficiente de dados de forma transparente na nuvem. O modelo proposto pelos autores facilita a execução de aplicações na nuvem com um aumento da carga inferior a 5% para aplicações científicas.

Um trabalho de interesse do ponto de vista de conhecimento da infraestrutura de nuvem é apresentado por Peter Sempolinski e Douglas Thain [74]. Eles apresentam um estudo comparativo entre três tipos de nuvem: Eucalyptus [29], OpenNebula [44] e Nim-

[1] Arquitetura Orientada a Serviço
[2] Acordo de Nível de Serviço *ServiceLevel Agreement*

bus [61]. Na comparação são abordados diversos modelos comparativos que auxiliam na tomada de decisão de qual nuvem usar para que tipo de aplicação e infraestrutura.

No artigo [42], os autores Iraklis e Joemon apresentam uma proposta de previsão e uso de nuvens computacionais para aplicações P2P, em especial a replicação e recuperação de arquivos e base de dados. Eles consideram a partilha e reutilização de recursos, tais como bancos de dados, avaliação de modelos e consultas, para validar que as redes P2P podem aumentar o fluxo de dados e disponibilizar acesso a mais usuários simultaneamente, diminuindo a degradação dos recursos.

Em outro artigo Shu-sheng explora um método para medir a capacidade de um supernó de resolver o problema de "escalonamento"de nós por insuficiência em atender a aplicação P2P [79].

2.1.2 Características Essenciais de uma Nuvem

Segundo Flávio Souza et al.[88], as características essenciais de uma nuvem são as vantagens que as soluções oferecem. Algumas destas características, em conjunto, definem exclusivamente a computação em nuvem e faz a distinção com outros paradigmas. Um exemplo citado pelos autores é a elasticidade rápida de recursos, amplo acesso e medição de serviço neste tipo de infraestrutura.

A computação na nuvem tem como um dos responsáveis e asseguradores das características essenciais o National Institute of Standards and Technology (NIST) [7] que é responsável pela elaboração de parâmetros técnicos para avaliação e padronização da infraestrutura, além de conceitos e definições sobre a computação em nuvem e seus diversos formatos. O NIST assegura as características essenciais que uma nuvem deve ter como:

- **Selfservice sob demanda:** O usuário que precisa de serviços como tempo de servidor e armazenamento de rede pode obtê-los automaticamente, sem necessidade

de interação humana com o provedor do serviço.

- **Amplo acesso à rede:** Os serviços estão disponíveis na rede e podem ser acessados através de mecanismos padrão por quaisquer plataformas, inclusive por telefones móveis, laptops e PDAs.

- **Compartilhamento de recursos:** Os recursos computacionais do provedor são compartilhados por múltiplos clientes, através de um modelo que oferece diferentes recursos que são dinamicamente atribuídos, de acordo com a demanda do usuário. Neste modelo, é como se a locação fosse independente, e o cliente em geral não controla e não sabe onde está a localização exata dos recursos fornecidos. Os recursos disponíveis incluem armazenamento, processamento, memória, capacidade de rede e máquinas virtuais.

- **Rápida elasticidade:** As necessidades do usuário podem ser atendidas com rapidez e flexibilidade, em alguns casos automaticamente, de forma a aumentar ou diminuir rapidamente a disponibilidade dos recursos, de acordo com a demanda.

- **Quantificação do serviço:** Sistemas em nuvem controlam automaticamente e otimizam o uso dos recursos computacionais através de ferramenta de medição apropriada ao tipo de recursos contratado (armazenamento, processamento, largura de banda, número de contas ativas de usuários). Em geral, o serviço é cobrado pelo uso (*payperuse*). A utilização dos recursos pode ser monitorada, controlada e informada de modo transparente tanto para o provedor como para o usuário.

2.1.3 Modelos de Serviços da Nuvem

Segundo o NIST [7] os modelos de serviços são:

- Software como serviço (*Software as a Service, SaaS*): O usuário utiliza aplicações do provedor que rodam em uma infraestrutura de nuvem. Tais aplicações podem ser acessadas a partir de vários dispositivos, através de uma interface como o navegador

(por exemplo, e-mail baseado na web), e neste caso, o consumidor não gerencia ou controla a infraestrutura de nuvem subjacente [21] [62].

- Plataforma como serviço (*Platform as a Service, PaaS*): Nessa modalidade, o usuário pode implementar, na infraestrutura de nuvem, aplicações próprias ou compradas (que controla), desenvolvidas em linguagens de programação e ferramentas suportadas pelo provedor. Mas o usuário não gerencia ou controla a infraestrutura de nuvem envolvida [8].

- Infraestrutura como serviço (*Infrastructure as a Service, IaaS*): Nesse modelo, a oferta do provedor inclui processamento, armazenagem, redes e outros recursos computacionais fundamentais com os quais o usuário possa usar e rodar quaisquer softwares, sejam eles sistemas operacionais ou aplicativos. O usuário não gerencia nem controla a infraestrutura de nuvem envolvida, mas controla os sistemas operacionais, o armazenamento, as aplicações implementadas e, possivelmente, os componentes selecionados de rede como firewalls [60].

Em um ambiente que fornece infraestrutura como serviço (IaaS) é importante controlar os recursos, aumentando ou diminuindo a capacidade de elasticidade em resposta às alterações na carga. Uma proposta de trabalho neste sentido é apresentada por Paul Marshall et al. [53], no trabalho é apresentado um site "elástico"que se adapta de acordo com as necessidades como aumento inesperado de acesso. Para testar o ambiente os autores criaram três políticas: sob demanda, fluxo constante e rajadas, e concluíram que, pode-se processar 10 vezes mais rápido, expandindo o cluster EC2 [25] em até 150 nós.

2.1.4 Modelos de Nuvem

O NIST [7] apresenta ainda os 4 modelos básicos de nuvem:

- **Privada:** A infraestrutura da nuvem é operada exclusivamente para uma empresa/organização. Ela pode ser gerenciada pelo usuário, ou por terceiros, e pode existir dentro ou fora da empresa/organização.

- **Comunitária:** A infraestrutura da nuvem é compartilhada por diversas entidades e suporta uma comunidade específica que tem objetivos comuns. Pode ser gerenciada pelas organizações ou por terceiros,além de poder ser interna ou externa.

- **Pública:** Sua infraestrutura está disponível para o público, ou para um grande grupo empresarial, e sua propriedade pertence ao provedor de serviços em nuvem.

- **Híbrida:** Tem a infraestrutura formada por duas ou mais nuvens (privada, comunitária ou pública) que mantêm suas próprias identidades, mas operam em conjunto através de uma tecnologia padrão ou proprietária que permite portabilidade de dados e aplicativos, como a suspensão da nuvem para balanceamento entre nuvens.

2.1.5 Escalabilidade em Nuvens Computacionais

Escalabilidade ou Elasticidade em nuvens computacionais são as capacidades dos ambientes e/ou programas, de diminuir ou aumentar automaticamente, e isso é uma das partes mais interessantes da computação em nuvem, onde pode-se pagar pelo uso básico e ir acrescentando mais recursos à medida em que forem necessários.

Quando usa-se mais recursos, paga-se mais, pagando-se menos para menor quantidade de uso. Um ponto importante desta abordagem é que todo o processo acontece de maneira transparente para o usuário.

Luis M. Vaquero et. al [97] cita que a escalabilidade é referenciada como uma das principais vantagens trazidas pelo paradigma de nuvem computacional. Entretanto os autores garantem que existem algumas questões importantes a serem levadas em consideração antes de fazer o dimensionamento adequado e automatizado para aplicações reais.

No trabalho são apresentadas as principais iniciativas em relação à escalabilidade de toda uma aplicação em ambientes de nuvem. Os autores apresentam ainda os esforços relevantes da tecnologia, fornecendo uma visão abrangente das tendências que cada uma

segue.

Eles mostram que houveram grandes avanços em direção automatização e gestão de VMs inter-relacionadas e dependentes do contexto (ou seja, um serviço) de uma forma holística através de políticas e regras. No trabalho, há uma preocupação na parte de gestão de aplicações distribuídas por várias nuvens, mantendo os objetivos de desempenho buscando algoritmos apropriados para balanceamento de carga.

Os autores mostram ainda que a Plataforma como Serviço idealmente escalável deve ser capaz de libertar ou instanciar os componentes dos utilizadores de acordo com a mudança na demanda, transparentemente para o usuário, distribuindo a carga entre os servidores na nuvem.

Para facilitar as tarefas dos desenvolvedores, o conceito de seção deve ser implementado pela plataforma, o que requer suporte para replicação de dados de forma transparente. Mas a replicação de dados exige que instâncias de componentes mantenham uma cópia atualizada dos dados.

Esta abordagem entretanto traz a necessidade de atualizações constantes, e estas consomem largura de banda e tempo, o que pode levar a grandes atrasos sobre os pedidos de processamento, inviabilizando o seu uso.

Um problema equivalente encontra-se em nível de banco de dados, que devem manter acesso aos tradicionais bancos de dados relacionais com suporte a transações ACID[3]. Mas, como mais réplicas são criadas para atender a uma demanda crescente, a sobrecarga necessária para manter a consistência acaba por induzir atrasos nos pedidos.

Outro trabalho experimental que trata da escalabilidade de nuvens computacionais é

[3]Atomicity, Consistency, Isolation, Durability

apresentado por Upendra Sharma et al. [77]. Eles apresentam um protótipo de software chamado Kingfisher usando a nuvem OpenNebula [44] para avaliar o desempenho em uma nuvem Xen [19] privada e uma nuvem Amazon EC2 [47] com respeito ao provisionamento de aplicações. Os autores afirmam que com um "consumo consciente"é possível ter uma "economia"na infraestrutura da ordem de 24% na nuvem Xen e de até 35% na nuvem EC2, comparados a outros trabalhos.

Os autores demonstram ainda que a integração de múltiplos mecanismos como a migração e replicação em uma abordagem unificada pode aumentar a economia em até 2 vezes. A abordagem explora a replicação e migração de forma dinâmica e funções lineares para calcular a carga do sistema e das aplicações.

Já Aarthi Raveendran et al. [68] apresentam um modelo para tornar as aplicações MPI [33] [90] [75] elásticas, considerando as limitações dessas aplicações atualmente.

O sistema apresentado pelos autores inclui uma camada de decisão que considera restrições de tempo e de custos, um quadro para modificar programas MPI, e um suporte baseado na nuvem que pode permitir a redistribuição de dados guardados, para apoiar a alocação de recursos de forma automatizada e reiniciar o aplicativo em um diferente número de nós.

Ainda sobre o ponto de vista de elasticidade, uma aplicação pode ter que ser portável de uma implementação para outra, nesse contexto, Dinesh Rajan et al. [67] apresentam uma implementação elástica, com melhor escalabilidade e adaptabilidade de recursos em tempo de execução, provendo tolerância a falhas e portabilidade na nuvem de forma transparente para o usuário. A proposta é a de conversão de uma implementação MPI de replicação implementada paralelamente, para uma aplicação na nuvem elástica usando a estrutura de fila.

Na mesma linha, a aplicação ELASTIN definida por Iulian Neamtiu et al. [59] propõe um modelo de conversão de aplicações não elásticas para elásticas, tirando do programador essa preocupação, funcionando como um compilador de programa.

2.1.6 Computação de Alto Desempenho

Um dos termos que mais são apresentados no modelo de computação atual, especificamente na nuvem, é o HPC. HPC do inglês *High-Performance Computing* refere-se ao uso de supercomputadores ou clusters de alto desempenho para execução de tarefas que demandam grande quantidade de recursos.

2.2 Virtualização

Virtualização, basicamente, é a técnica de separar Aplicação e Sistema Operacional dos componentes físicos. Por exemplo, um Servidor Virtual possui aplicação e sistema operacional como um Servidor Físico, mas este não está vinculado ao Hardware e pode ser disponibilizado onde for mais conveniente.

Como uma aplicação deve ser executada em um sistema operacional em um hardware, com o uso da virtualização esta aplicação pode ser executada em um servidor ou ambiente centralizado e ser portada para outros sistemas operacionais e hardwares, sem a necessidade de re-compilação, por exemplo.

A virtualização tem papel fundamental na computação na nuvem, no que tange a segurança, replicação e tolerância a falhas como define Simons et al. [83], principalmente com o uso da computação de alto desempenho e da possibilidade de migração de carga de trabalho de forma dinâmica. Outro ponto importante é a escalabilidade, quando há necessidade de mais recursos, basta que se crie novas instâncias de máquinas virtuais, por

exemplo.

Paulo Antonio Leal Rego et al. [70] propõem uma arquitetura para lidar com a alocação de máquinas virtuais com base na capacidade de processamento ou nuvens heterogêneas. Os autores citam que a grande contribuição do trabalho é uma representação da capacidade de processamento, em termos da unidade de processamento (PU) e a limitação do uso da CPU, afim de isolar a capacidade de processamento da máquina física (PM), onde a Máquina Virtual (VM) está "hospedada".

2.2.1 Ambientes Virtualizados

A virtualização de ambientes trouxe uma grande quantidade de benefícios para as empresas e em especial aquelas que usam a nuvem computacional ao invés de grandes CPDs. São estes ambientes que permitiram ao gerente de recursos disponibilizar maior flexibilidade e capacidade, escalando estes recursos à medida que forem necessários, sem ter que se preocupar com aquisição de novos servidores reais.

Outro ponto importante dos ambientes virtualizados é a facilidade de controle e disponibilidade, por exemplo, supondo que haja a necessidade de manter redundância de dois servidores para garantir alta disponibilidade: em ambientes sem virtualização seria necessário que os ambientes fossem "clonados"como um todo, ou seja, todos os arquivos que fossem alterados, precisariam ser também alterados na máquina "clone", se forem considerados os sistemas operacionais, seria necessário copiar milhares de arquivos.

Por outro lado, com o uso de virtualizadores, considerando que estes possuem um arquivo com a imagem do sistema, "visível"à partir da máquina hospedeira, seria necessário modificar somente este arquivo, simplificando todo o processo e garantindo maior confiabilidade.

Há ainda a possibilidade de "hospedar"diversas máquinas virtuais em um único recurso físico. Elder Rodrigues et al. [23] citam que a migração de máquinas virtuais entre máquinas físicas, acarreta em uma redução do tempo de *down time* de uma aplicação, no caso de uma falha do hardware ou do software.

Neste sentido, uma abordagem necessária seria estabelecer o impacto que estas aplicações sendo executadas em diversas máquinas virtuais, hospedadas em um mesmo servidor, traria para o desempenho do servidor, das próprias instâncias de máquinas virtuais e nas aplicações sendo executadas.

2.2.2 Formas Mais Comuns de Virtualização

Dentre as formas mais comuns de virtualização que são apresentadas por Lunsford et al. [49], e considerando o trabalho descrito nesta tese, ressaltam-se as seguintes:

1. **Virtualização de Servidor:** Técnica de execução de um ou mais servidores virtuais sobre um servidor físico. Esta abordagem permite uma maior densidade de utilização de recursos (hardware, espaço e etc.), mantendo isolamento e segurança.
2. **Virtualização de Aplicação.** A Virtualização de Aplicação permite executar aplicações em um ambiente virtualizado no desktop do usuário, isolando a aplicação do sistema operacional. Isso é possível através do encapsulamento da aplicação no ambiente virtual.
3. **Virtualização de Desktop.** Consiste na execução de múltiplos sistemas operacionais em uma única estação de trabalho, permitindo que uma aplicação seja executada em um sistema operacional não compatível.
4. **Virtualização de Apresentação.** A virtualização de apresentação permite executar e manter o armazenamento das aplicações em servidores centralizados, enquanto provê uma interface familiar para o usuário em sua estação.

5. **Infraestrutura de Desktop Virtual (Virtual Desktop Infrastructure - VDI):** O VDI permite que sejam hospedas máquinas virtuais cliente em uma infraestrutura de virtualização como acontece com a "Virtualização de Servidores".

6. **Virtualização de Perfil:** Com a virtualização de perfil, os usuários podem ter os documentos e perfis de uma máquina específica, o que permite a fácil movimentação do usuário para novas estações de trabalho em caso de roubo ou quebra de equipamento. A Virtualização de Perfil também permite ter uma experiência de desktop única quando utilizando outras tecnologias de virtualização, como VDI.

Normalmente uma infraestrutura de nuvem computacional é composta de hardware de diversos tipos e arquiteturas, com capacidades de disco, memórias e processadores variados, além de sistemas operacionais diferentes [87].

Nestas máquinas diferentes podem estar sendo executados diversos servidores virtuais, com características diferentes, o que torna difícil o monitoramento e controle dessa infraestrutura, ao mesmo tempo em que permite que diversas aplicações possam contar com um ambiente de monitoramento deste tipo de recurso e infraestrutura, para torná-las mais adaptativas e cofiáveis.

Mark Mergen et al. [54] têm uma visão diferenciada com respeito às nuvens, os autores mostram a virtualização como uma infraestrutura atraente aos sistemas comerciais, mostrando principalmente as vantagens em questões de variedade de sistemas operacionais, produtividade, desempenho, confiabilidade, segurança, disponibilidade e simplicidade. No trabalho são apesentadas diversas vantagens dessa tecnologia.

2.2.3 Teste de Desempenho

Para que o trabalho aqui exposto seja viável, é importante que haja um modelo de teste de desempenho que adeque cada recurso ao seu uso e mantenha a valoração de degradação

causada pela concorrência por estes recursos.

Para alcançar este objetivo, o teste de desempenho tende a determinar a confiabilidade, a capacidade de resposta de um recurso físico, o *throughput*, a confiabilidade de um recurso ou a escalabilidade de um host, sob uma determinada carga efetuada sobre cada um dos recursos disponíveis, como por exemplo, o cálculo do tempo de escrita e leitura de um arquivo no disco, ou o processamento de uma equação matemática, tanto executando individualmente, quanto concorrentemente, neste último caso avaliando a degradação causada no recurso real.

No contexto deste trabalho a avaliação foca no tempo de execução de uma aplicação categorizada previamente. Para esta avaliação, primeiramente é coletado o tempo individual de processamento em um ambiente livre de concorrência e novamente a aplicação é executada, só que neste teste são usadas outras aplicações concorrentemente. O tempo de execução é novamente tomado e calcula-se a diferença entre a execução com e sem concorrência. A esta diferença, quando existente, será dado o nome de grau de degradação.

Efetuar testes de desempenho é tão importante que no artigo [95], Radu Tudoran et al. demonstram um teste comparativo entre duas nuvens, Nimbus [61] e Azure [55] para avaliar o desempenho destas nuvens para aplicações científicas em infraestruturas públicas e privadas.

No trabalho foram consideradas as necessidades primárias de uma nuvem como: Poder Computacional, Armazenamento, Transferência de Dados e Custo. A avaliação foi feita usando Benchmarks e aplicações científicas. Por fim foi constatado que a nuvem Nimbus tem menos variabilidade e aumenta o desempenho para aplicações que acessam muitos dados. De acordo com os autores, a nuvem Azure tem menor custo e implementação mais rápida.

2.2.4 Aplicações Comerciais Versus Aplicações Científicas

Aplicações Comerciais são normalmente diferentes de aplicações científicas. Enquanto que a primeira muitas vezes requer a disponibilidade de um grande número de computadores para a realização de experimentos em grande escala. A segunda procura otimizar os recursos para compor o melhor custo/benefício, já que neste tipo de aplicação, sabe-se antecipadamente os requisitos mínimos para operar o sistema.

Em ambientes científicos, tradicionalmente, essas necessidades são abordadas, utilizando-se soluções de computação de alto desempenho e facilidades instaladas, como clusters e super computadores, que são difíceis de configurar, manter e operar [100].

Vecchiola [100] cita ainda que a computação na nuvem fornece aos cientistas um modelo completamente novo de utilizar a infraestrutura computacional. Recursos de computação, recursos de armazenamento, bem como aplicações, podem ser dinamicamente provisionados (e integrados na infraestrutura existente), normalmente "pagando-se"por uso. Esses recursos podem ser liberados quando eles não são mais necessários.

Esses serviços muitas vezes são oferecidos dentro do contexto de um acordo de nível de serviço (SLA), que garante a qualidade desejada de serviço (QoS). Aneka[4] [18], apresenta uma solução corporativa de computação em nuvem e aproveita os recursos de computação, baseando-se em nuvens públicas e privadas para oferecer aos usuários a qualidade de serviço desejada.

Sua infraestrutura flexível suporta serviço baseados em paradigmas de programação múltiplas que fazem Aneka abordar uma variedade de cenários diferentes a partir de aplicativos de finanças para a ciência da computação. No trabalho [100] são apresentados exemplos de computação científica na nuvem, com o uso do software Aneka para a classificação de dados de expressão genética.

[4]plataforma orientada a serviços para desktop baseado em grades computacionais

Outro modelo de aplicação científica na nuvem é demonstrado por Jared Wilkening [103], os autores apresentam uma aplicação de sequenciamento de nucleotídeos usando BLAST [43]. A proposta do trabalho é de fazer análise de conjuntos de dados, já que esses requerem recursos computacionais significativos.

No trabalho os autores usaram a computação na nuvem, por oferecer uma possibilidade "tentadora"para acesso sob demanda aos recursos de computação. No entanto, eles citam muitas questões em aberto e apresentam uma avaliação de desempenho do BLAST em dados reais de metagenômicos em um ambiente de nuvem, afim de determinar a viabilidade desta abordagem.

No caso de aplicações comerciais na nuvem, o artigo [56] apresenta uma solução comercial para o uso em serviços terceirizados e divide o plano de negócio em 4 camadas como é apresentado na Figura 2.1: Serviços de TI, Processos, Serviços e Contexto de Negócios.

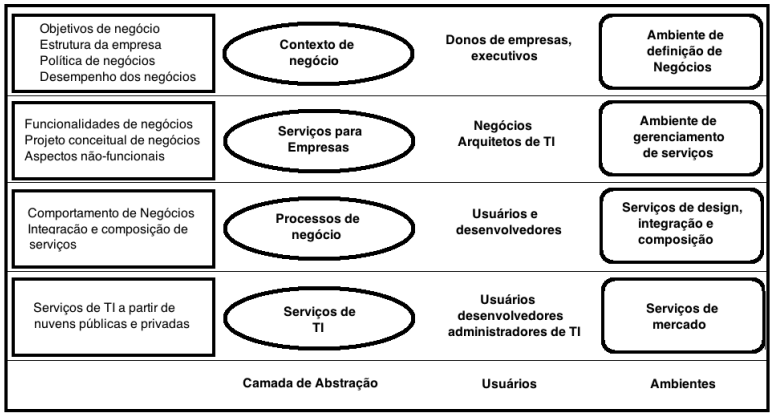

Figura 2.1: Arquitetura de Negócios em um Ambiente de Serviços Terceirizados [56].

Já no contexto de aplicações comerciais, para se manter a qualidade de serviço, os administradores devem provisionar recursos suficientes para lidar com as flutuações de carga de trabalho das aplicações, sem disponibilizar recursos em excesso, considerando que os recursos em excesso reduzem os lucros, enquanto que insuficiência de recursos degrada o desempenho.

No artigo [22] Wesam Dawoud et al. apresentam uma arquitetura de um sistema elástico para gestão de recursos e otimização dinâmica de aplicações em ambiente virtualizado. Na arquitetura implementada foram propostos três controladores de CPU, memória e aplicativos. Os autores concluíram que considerando a linha de otimização de aplicações, com uso dinâmico de CPU e memória, podem reduzir os objetivos de nível de serviço (SLAs) mantendo o desempenho das aplicações sendo executadas.

2.2.5 Dependabilidade

Dependabilidade foi traduzida dessa forma a partir do termo inglês *Dependability* [37]. Outras possíveis traduções poderiam ser - Garantia de Funcionamento e/ou Confiabilidade - Define-se nesse contexto, qualidade de um sistema computadorizado de ser dependível, que significa um sistema do qual se pode depender para a prestação de um determinado serviço ou desempenho de uma dada tarefa [80] [45].

Dependabilidade pode ser definida através de 2 conceitos chave - qualidade do serviço fornecido por um dado sistema e confiança neste serviço - Os principais atributos (medidas) neste contexto são: confiabilidade, disponibilidade, segurança (*safety*), mantenabilidade, testabilidade e *performability*.

Dos atributos mais conhecidos e usados como sinônimos de dependabilidade estão a disponibilidade e a confiabilidade. Disponibilidade que é a característica de manter o ambiente sempre em condições de uso e Confiabilidade, que dado que o ambiente está disponível, ele "responderá"corretamente às requisições e seus resultados serão concisos.

Pode-se definir que dependabilidade é o principal foco em praticamente todas as áreas de Ciência da Computação, em especial computação na nuvem. Devido principalmente à importância para o usuário que necessita de qualidade de serviço bem como do ambiente

de execução de processos de software que necessitam de garantias de execução.

No contexto deste trabalho, dependabilidade é tratada como o modelo de testes, confiabilidade do ambiente e disponibilidade quando são executadas aplicações concorrentemente.

CAPÍTULO 3

CONSIDERAÇÕES INICIAIS SOBRE A PESQUISA

Este capítulo apresenta os principais conceitos relacionados à tese e os trabalhos relacionados ao proposto. Na Seção 3.1 descreve-se o conceito de *Dwarfs*, seguido na Seção 3.2 pela apresentação do pacote de *Benchmarks* Rodinia, na Seção 3.3 apresenta-se a definição do termo principal deste trabalho "Afinidade"e finalizando este capítulo, apresentam-se alguns trabalhos com grau de similaridade com o proposto na Seção 3.4.

3.1 Dwarfs

Erich L. Kaltofen [39] define *Dwarfs* como "um método algorítmico que captura um padrão de computação e comunicação", este padrão tem importância no trabalho aqui proposto por servir de base de conhecimento e caracterização de tipos de aplicação, assim, quando avalia-se que uma determinada aplicação pode ser executada em ambientes compartilhados junto a outras aplicações, sem degradar o recurso compartilhado, deduz-se que outras aplicações do mesmo tipo (*Dwarfs*) podem também concorrer pelo mesmo recurso sem degradá-lo.

Com o objetivo de categorizar os estilos de computação, o trabalho de Phillip Colella [20] identificou sete métodos numéricos que ele acreditava serem importantes para a ciência e engenharia e introduziu os "Sete Dwarfs" de Computação Científica.

Estes Dwarfs são definidos em um alto nível de abstração para explicar o seu comportamento em diferentes aplicações HPCD e cada classe de Dwarf têm semelhança em computação e comunicação. De acordo com sua definição, as aplicações de uma determinada classe podem ser implementadas de forma diferente com a mudança de métodos

numéricos ao longo do tempo, mas os padrões subjacentes permanecerão os mesmos ao longo de gerações de mudança e continuará a ser o mesmo em futuras implementações.

A equipe de computação paralela de Berkeley estendeu estas classificações a treze Dwarfs depois de terem examinado domínios importantes de aplicações. Eles estavam interessados em aplicar os Dwarfs para um número mais amplo de métodos computacionais e investigar a forma como os Dwarfs podem capturar padrões de computação e comunicação para uma ampla gama de aplicações [5].

Os autores compararam diversas classes de Dwarfs com coleções de referência para computação embarcada (42 benchmarks EEMBC [1]) para desktop e servidores (28 benchmarks SPEC2006 [2]).

Além disso, examinaram domínios de aplicação de suma importância para: inteligência artificial, aprendizagem de máquina, software de banco de dados e computação gráfica/jogos. O objetivo foi definir os requisitos de aplicações, e então tirar conclusões sobre os requisitos mínimos de hardware para executar tais aplicações.

Um conjunto diversificado de aplicações científicas importantes são suportadas pelos atuais 13 Dwarfs. Na Tabela 3.1 é apresentada uma lista de classes de Dwarfs com base no trabalho de Berkeley, com uma breve descrição. Uma discussão mais completa é apresentada em [5, 38, 89].

As classes Dwarfs que estão sendo utilizadas neste trabalho são: Álgebra Linear Densa (DLA), Grade Estruturada (SG) e Grafo Transversal (GT). A escolha por essas três classes é dada pois há um grande número de aplicações científicas em diversas áreas científicas classificadas nestes, como é apresentado na Figura 3.1.

[1] http://www.eembc.org
[2] http://www.spec.org/cpu2006

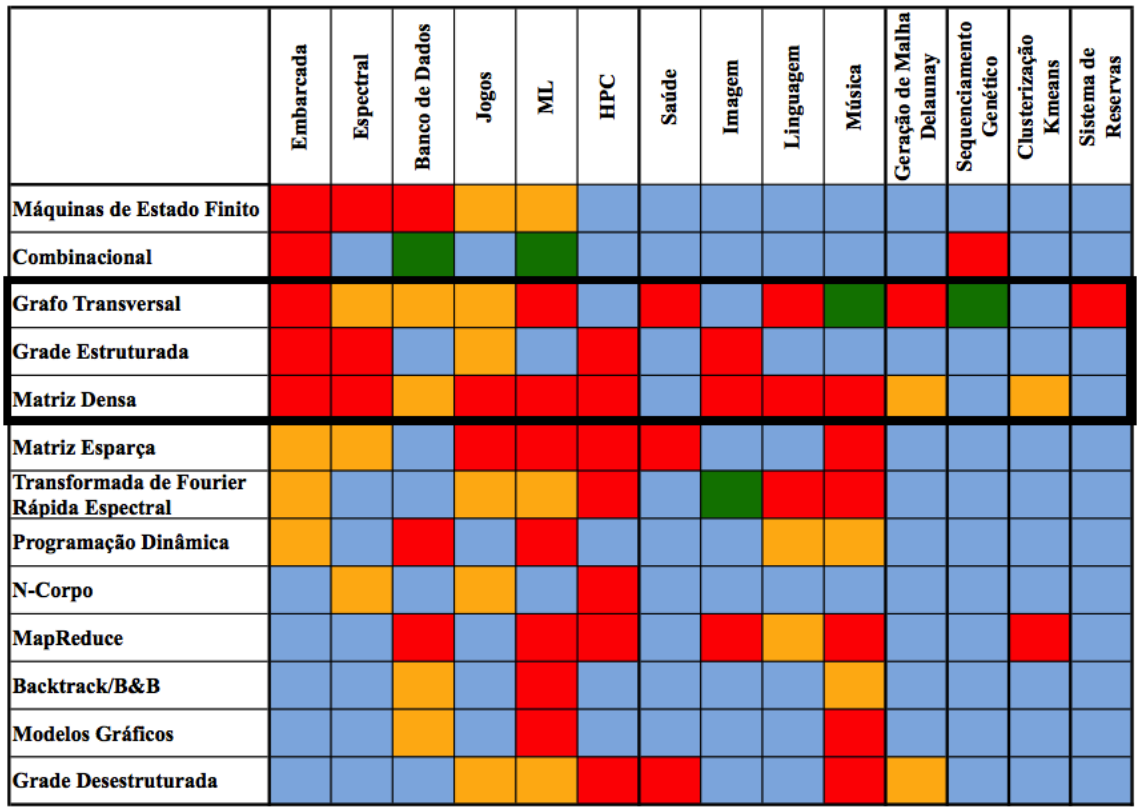

Figura 3.1: Alguns exemplos de áreas científicas que têm sua aplicação caracterizada por classes de Dwarfs (`http://stamp.stanford.edu`). As cores representam a relevância da classe para aplicações de domínio (mais relevantes - vermelho, menos relevantes - azul). A ênfase está em classes usadas neste trabalho.

Além disso, esta tese se concentra nestas três classes porque, juntas, oferecem um conjunto diversificado de padrões, que compreendem um conjunto mais completo de experimentos: aplicações científicas classificadas na classe DLA são limitadas computacionalmente, enquanto que na classe GT possuem maior latência de memória e em SG são limitadas na largura de banda de memória.

3.2 Rodinia

O Rodinia [17] é um pacote de benchmark para a computação heterogênea que ajuda no estudo de plataformas de hardware, em especial de GPUs (*Graphics Processing Units*) e CPUs (*Central Processing Unit*) multicore. As aplicações contidas no pacote são baseadas nos Dwarfs e este foi o motivo da escolha desta suíte de benchmarks.

Tabela 3.1: Os treze Dwarfs

Nome do Dwarf	**Descrição**
Álgebra Linear Densa	Este tipo de algoritmo faz operações com matrizes e vetores densos.
Álgebra Linear Esparsa	O mesmo problema que Álgebra Linear Densa, mas em matrizes com muitos valores zero. Para reduzir o espaço e computação, estes algoritmos operam em uma lista de valores e índices, resultando em mais acesso indireto à memória.
Métodos Espectrais	Este tipo de algoritmo envolve muitas vezes o uso de uma transformada rápida de Fourier e transformação de dados de/para em domínio espacial ou temporal. O perfil de execução normalmente tem vários estágios de processamento interdependentes.
N-Corpo	Este tipo de algoritmo calcula interações entre muitos pontos discretos que são caracterizados por um grande número de cálculos independentes dentro de uma iteração, seguido de comunicação todos-para-todos.
Grade Estruturada	Neste tipo de algoritmo os dados são organizados em uma grade regular, onde a computação prossegue como uma série de atualizações desta grade. Para cada atualização da grade, todos os pontos são atualizados com valores a partir de um pequena vizinhança em torno de cada ponto.
Grade Não-Estruturada	Os dados são organizados em grades irregulares. Atualizações normalmente envolvem vários níveis de indireção para referência à memória. Atualizar qualquer ponto requer primeiro determinar uma lista de pontos vizinhos e, em seguida, carregar os valores daqueles vizinhos.
Map Reduce	Este tipo de algoritmo executa independente e repetidamente uma função de mapa e os resultados são agregados no final através de uma função que reduz estes mapas (*map and reduce*). Não é necessária qualquer comunicação entre os processos em fase de mapa, mas a fase redução exige uma comunicação global.
Lógica Combinacional	Este tipo de algoritmo realiza operações simples em grandes quantidades de dados, muitas vezes exploram um baixo nível de paralelismo para atingir alto rendimento.
Grafo Transversal	Este tipo de algoritmo visita muitos nós em um grafo. Normalmente envolve uma quantidade significativa de memória RAM para pesquisas indiretas.
Programação Dinâmica	Este tipo de algoritmo calcula uma solução através da resolução de uma série de subproblemas mais simples que se sobrepõem.
Backtrack & Branch Bound	Este tipo de algoritmo faz uma pesquisa em um grande espaço para encontrar uma solução global ótima usando recursividade, dividindo a região viável em subdomínios.
Modelos Gráficos	Este tipo de algoritmo constrói gráficos que representam variáveis aleatórias como nós e probabilidades condicionais como bordas.
Máquina de Estado Finito	Este tipo de algoritmo possui um sistema cujo comportamento é definido por estados, transições definidas pela entrada e eventos associados com transições ou estado atual. Esses algoritmos são altamente dependentes de operações condicionais e dados interdependentes.

Os testes mostraram que o pacote de algoritmos implementados no Rodinia cobre muitos padrões de comunicação em sistemas paralelos, reforçando sua aplicação no trabalho aqui proposto. A Tabela 3.2 apresenta as aplicações, os dwarfs correspondentes, o domínio da aplicação e o tipo de biblioteca de paralelismo (OpenCL, OpenMP e Cuda).

Tabela 3.2: Aplicações do *Benchmark* Rodinia [84].

Aplicações	**Dwarfs**	**Domínios**	**Tipos de Paralelismo**
Leukocyte	Grade Estruturada	Imagens Médicas	CUDA, OMP, OCL
Heart Wall	Grade Estruturada	Imagens Médicas	CUDA, OMP, OCL
MUMmerGPU	Grafo Transversal	Bioinformática	CUDA, OMP
CFD Solver1	Grade Sem Estrutura	Dinâmica dos Fluidos	CUDA, OMP, OCL
LU Decomposition	Álgebra Linear Densa	Álgebra Linear	CUDA, OMP, OCL
HotSpot	Grade Estruturada	Simulação Física	CUDA, OMP, OCL
Back Propagation	Grade Sem Estrutura	Reconhecimento de Padrões	CUDA, OMP, OCL
Needleman-Wunsch	Programação Dinâmica	Bioinformática	CUDA, OMP, OCL
Kmeans	Álgebra Linear Densa	Mineração de Dados	CUDA, OMP, OCL
Breadth-First Search1	Grafo Transversal	Algoritmos Gráficos	CUDA, OMP, OCL
SRAD	Grade Estruturada	Processamento de Imagens	CUDA, OMP, OCL
Streamcluster1	Álgebra Linear Densa	Mineração de Dados	CUDA, OMP, OCL
Particle Filter	Grade Estruturada	Imagens Médicas	CUDA, OMP, OCL
PathFinder	Programação Dinâmica	Grade Transversal	CUDA, OMP, OCL
Gaussian Elimination	Álgebra Linear Densa	Álgebra Linear	CUDA, OCL
k-Nearest Neighbors	Álgebra Linear Densa	Mineração de Dados	CUDA, OMP, OCL
LavaMD2	N-Corpo	Dinâmica Molecular	CUDA, OMP, OCL
Myocyte	Grade Estruturada	Simulação Biológica	CUDA, OMP, OCL
B+Tree	Grafo Transversal	Busca	CUDA, OMP, OCL

3.2.1 OpenCL (*Open Computing Language*)

OpenCL ou ainda OCL [34] [57] é uma biblioteca de programação paralela que permite obter um padrão para escrever aplicativos que acessam todos os recursos de programação disponíveis, tanto em CPUs, GPUs quanto em outros processadores. Tem excelente potencial para PCs, servidores, dispositivos móveis, computação de alto desempenho, e até mesmo sistemas de nuvem.

A escolha por esta bilblioteca teve motivação relacionada à disponibilização desta no pacote Rodinia e que, mesmo sendo proposta para uso em GPUs é possível utilizá-la em CPUs.

3.2.2 OpenMP (*Open Multi Processing*)

OpenMP ou ainda OMP [65] é uma interface de programação (API), suportada pela maioria das arquiteturas de processadores e sistemas operacionais incluindo Solaris, AIX, HP-UX, Linux, plataformas Mac OS X e Windows, baseada no modelo de programação paralela de memória compartilhada para arquiteturas com múltiplos processadores. Atualmente é mantido pela OpenMP ARB (*Architecture Review Board*) e possui bibliotecas para três linguagens de programação, Fortran, C e C++.

Seus componentes básicos são diretivas de compilação, biblioteca de execução e variáveis de ambiente, os quais influenciam no comportamento e no tempo de execução. OpenMP usa um modelo portátil, escalável, que dá aos programadores uma interface simples e flexível para o desenvolvimento de aplicações paralelas para diversos tipos de arquitetura, de desktops aos supercomputadores.

A escolha por esta bilblioteca teve motivação relacionada à disponibilização desta no pacote Rodinia e que foi possível comparar os efeitos de uma biblioteca desenvolvida para GPUs executando em CPUs com uma feita exclusivamente para uso em CPUs.

3.3 Afinidade

Afinidade segundo o dicionário da língua portuguesa pode significar conformidade, analogia ou relação, definições estas que podem também ser usadas no contexto desse trabalho. Entretanto, para definir como será tratado o termo afinidade neste trabalho, esta seção apresenta algumas limitações relacionadas com a utilização de ambientes virtualizados, especialmente ambientes HPCD e o conceito de afinidade entre os ambientes virtuais (neste caso, nuvens, datacenters consolidadas ou qualquer tipo de ambiente computacional utilizando a virtualização como uma ferramenta para encapsular um aplicativo específico ou grupo de aplicações).

Pela definição proposta, serviços de afinidade em nuvens computacionais podem oferecer aos clientes, recursos de computação sob demanda diferentemente da infraestrutura tradicional, ao passo em que novas demandas de recursos permitem aos clientes acessarem o que precisam e onde precisam, já que possuem informações do tipo de atividade e de recursos necessários por cada tipo de aplicação, permitindo pagar pelo que realmente foi usado, e prevendo necessidades de uso sem excesso de recursos.

O termo afinidade é também proposto por Tanenbaum [92], ele define que há afinidade de processos em relação à CPU quando há um "algoritmo inteligente", que é escalonado para a mesma CPU que deixou de processar por ter acabado o tempo de execução. O motivo declarado no livro é de que se um processo "A" está sendo executado em um processador k, o cache desse processador pode conter ainda os dados de processamento desse processo quando retomar o tempo da CPU.

Com isso, o tempo gasto para troca de páginas da cache é reduzido, justificando a afinidade entre esse processo e o processador. Entretanto, esta não é uma situação de afinidade em nuvens e sim um conceito de afinidade para processadores, o que foge do contexto deste trabalho, servindo nesse sentido ao entendimento de afinidade entre hardware e software e suas vantagens em termos de ganho de velocidade de processamento.

Por outro lado, apesar de alguns trabalhos demonstrarem afinidade como sendo uma metodologia de ganho de velocidade de processamento, essas definições focam unicamente em máquinas reais e somente da afinidade entre processadores e memórias e não no contexto de máquinas virtuais em nuvens computacionais. Esses detalhes são apresentados em [15] e [71].

No contexto geral deste trabalho, definir-se-á afinidade como um modelo em que é possível predizer quais tipos de aplicações científicas podem concorrer, em ambientes vir-

tualizados, no mesmo recurso físico, ou seja, aquelas aplicações que estão sendo executadas concorrentemente em máquinas virtuais diferentes na mesma máquina "hospedeira"e competindo pelos mesmos recursos físicos (CPU, Memória, HD, etc.).

Com relação à concorrência e o problema da disputa por recursos computacionais (concorrência), há importância até mesmo para ambientes reais. Do ponto de vista do ambiente HPCD, a influência exercida pela arquitetura do sistema e a contenção são bem apresentadas no trabalho de Skinner e Krammer [85] onde são analisados os dados obtidos durante dois anos.

Os autores listam os seguintes fatores como causas da variabilidade: contenção de recursos, a comunicação entre e dentro de nós, agendamento de processos do kernel, contenção e atividades do sistema. Esses fatores tornam-se mais críticos quando se olha para um ambiente real, hospedando vários ambientes virtualizados.

Neste sentido os efeitos da concorrência e da degradação causada pelo processamento em um ambiente virtualizado, que consome todo o processador do servidor real são apresentados em [93], onde são mostrados os efeitos sobre um outro processo em um ambiente virtualizado diferente, quando são alocados no mesmo servidor real e têm que competir pelo uso da CPU, "minando"sua capacidade de processamento e desempenho, além de degradar todo o sistema real onde estão hospedados os ambientes virtualizados.

Os exemplos anteriores mostram o que pode acontecer com os ambientes virtuais quando hospedados no mesmo servidor real. Neste sentido, tenta-se propor qual seria a melhor abordagem a ser adotada para minimizar esta perda quando da atribuição de ambientes virtualizados, seja em uma nuvem, *data centers* ou mesmo em ambientes de pequena escala, para otimizar o uso de recursos reais.

Para isso, a abordagem proposta e adotada por esta tese é a de usar a avaliação de

classe de aplicações, como é definido no trabalho de Colella et al. [20]. Os autores citam o uso do Dwarfs, onde os aplicativos são divididos em classes, e cada uma destas classes são capazes de capturar todas as principais características dessas aplicações.

Este trabalho propõe assim a utilização do termo afinidade, caracterizado pelo o grau de compatibilidade entre as classes de aplicações, cuja execução simultânea no mesmo ambiente de computação resultaria numa perda de desempenho mínima para estas aplicações e ao próprio ambiente.

Pode-se notar que, embora o foco da análise são os ambientes virtuais e perda de desempenho destes, este conceito pode ser usado também para ambientes reais. Neste sentido, o trabalho aqui apresentado, mesmo focando nos ambientes virtualizados em nuvens, apresenta os dados dos ambientes reais como informativo.

O informativo proposto e apresentado pode ser utilizado como base de conhecimento sobre que tipos de aplicações tem variação de desempenho entre ambientes reais e virtuais.

3.3.1 Afinidade de Aplicações

Como apresentado na Seção 3.1, Dwarfs são caracterizações de tipos de aplicações científicas. Estas aplicações são caracterizadas dentro de 13 tipos, cada um destes definidos em [9].

Pela definição dos Dwarfs, se uma aplicação científica tem um comportamento quando executada, espera-se que o mesmo tipo de comportamento aconteça em outras aplicações que pertençam ao mesmo tipo de Dwarf, assim, não é necessário que se teste todos os tipos existentes de aplicações, bastando que os testes aconteçam por tipos e classes bem definidos.

Neste trabalho não serão feitos testes com todos os Dwarfs, bastando apresentar que o estudo é viável e que leva à base do conhecimento necessário para tratar os tipos de aplicações.

Destes termos e do estudo proposto pode-se definir que "Afinidade de Aplicações"podem ser delimitadas pelos tipos de aplicações que podem coexistir em um mesmo recurso físico, no mesmo ambiente, real ou virtual, sem acarretar degradação deste ambiente, o que degradaria todos os recursos físicos ou reais hospedados neste ambiente compartilhado.

Ressalta-se aqui que o termo Afinidade é proposto nesta tese e sempre será tratado no decorrer do texto como o efeito de tipos de aplicações que podem coexistir em um mesmo recurso, sem que uma aplicação acarrete aumento do tempo de execução de uma outra aplicação, sendo esta executada em um ambiente real ou virtual e pertencente à mesma classe ou à classes de aplicações difeerenciadas, como proposto com o uso dos Dwarfs.

3.4 Trabalhos Relacionados

Nessa seção são apresentados alguns trabalhos que possuem algum grau de similaridade com o trabalho proposto, buscando principalmente os pontos fracos das aplicações e abordagens em comparação com este trabalho.

Um dos primeiros artigos que trata do tema afinidade em nuvens computacionais é descrito por Vignesh T. Ravi et. al [69] os autores citam o aumento da disponibilização de clusters de GPU em ambientes de nuvem. O artigo apresenta um quadro que permite a execução de aplicações dentro de máquinas virtuais de forma transparente em uma ou mais GPUs, buscando a virtualização destas GPUs eficientemente. Eles introduzem um método para calcular o índice de afinidade entre dois ou mais núcleos, o que fornece uma indicação de potenciais melhorias de desempenho no processo de consolidação do kernel.

O trabalho de Kambadur et al. [40] apresenta a necessidade de testes de iterferência entre aplicações. Os autores avaliaram a interferência com uso de datacenters e definem o trabalho como precursor de novos trabalhos nessa área, focando a necessidade de que outros autores avaliem o impacto das interferências. Neste sentido, o trabalho aqui proposto avalia não o acesso à datacenters, mas o impacto de tipos de aplicações nestes ambientes compartilhados.

Já o trabalho de Dwyer et al. [24] define a importância da avaliação dos tipos de aplicações com o uso de aprendizagem de máquina para avaliar o impacto de aplicações em ambientes sem prioridade de execução, também citam que há necessidade de conhecer os tipos de aplicações nestes ambientes compartilhados.

Juan M. Tirado et. al [94] propõem uma melhoria na média de tempo de latência, na recuperação de problemas de infraestrutura P2P. Os autores expõem as limitações de escalabilidade de servidores, principalmente devido ao aumento do consumo de banda, demonstrando ainda os problemas relacionados à preservação da localidade de conteúdo.

No artigo são apresentados modelos de localização e auto-organização de redes P2P baseadas em afinidade de redes. No trabalho cada nó P2P pode decidir autonomamente entrar ou sair dos clusters com base na própria composição de conteúdo ou de interesse, podendo alterar as suas posições, a fim de alcançar uma alta clusterização. Por fim, citam que há um aumento na recuperação de pesquisas da ordem de 12% e da latência da ordem de 45%.

Em outro trabalho, a proposta da Microsoft resume-se em definir padrões e práticas para o uso da computação em nuvem, mas em um ambiente proprietário, não privilegiando a proposta de software livre. Outro ponto é a necessidade de uso de outras ferramentas Microsoft para o uso da plataforma Azure[55], como o Windows Server. Fazendo desta uma solução inviável para o projeto aqui proposto.

Já M. Tim Jones [35] define abordagens com o uso de virtualização para prover alta disponibilidade em ambientes de nuvem, mas não aborda diretamente sistemas de gerência deste tipo de recursos, ou como alocar novos servidores virtuais sem que as aplicações neles contidas degradem o ambiente real. Ele aborda ainda o uso de ferramentas e tipos de aplicações em ambientes práticos, mas não propõe uma solução ao uso de gerência dinâmica dos recursos disponíveis.

Keith R. Jackson[36] apresenta em seu trabalho um comparativo entre clusters HPC para Amazon EC2 e a mesma abordagem na nuvem, mostrando as vantagens e desvantagens da migração de algumas aplicações para nuvem, principalmente consolidada em ambientes virtualizados. Como a abordagem trata de algumas aplicações que podem ser mais facilmente monitoradas, não há como prever que tipo de aplicações não teriam perda de desempenho, por exemplo.

Outro trabalho relacionado à computação na nuvem foi descrito por Rodrigo N. Calheiros et al. [14] neste trabalho é apresentado o CloudSim, cujo objetivo é fornecer um sistema de simulação generalizada e extensível, permitindo a modelagem, simulação e experimentação de infraestruturas de computação em nuvem e serviços de aplicativos.

Com o CloudSim, pesquisadores e desenvolvedores podem se concentrar em questões específicas do sistema de design que eles queiram investigar, sem se preocupar com os detalhes de baixo nível, relacionados infraestruturas e serviços.

Os autores também mostram no artigo a necessidade de monitoramento de aplicações com intuito de otimizar o uso da nuvem e verificar os gargalos na infraestrutura. Entretanto o modelo apresentado foi proposto sobre um ambiente de testes, onde previsões são feitas, mas não há uma garantia de que o ambiente real siga o mesmo modelo de previsão, nem há um estudo que defina que tipos de aplicações poderiam coexistir nestes ambientes.

O trabalho desenvolvido por Kate et al. [41] descreve um sistema que permite aos usuários, automaticamente, terceirizar suas necessidades computacionais para os servidores na nuvem. Os autores descrevem as propriedades de escalabilidade e disponibilidade que são almejadas. Eles destacam EC2 [25] para trabalhar em grandes escalas, mas não garantem ou demostram estudos que avaliem a disponibilidade quando há concorrência entre aplicações neste ambiente.

Smita Vijayakumar et al. [101] apresentam um trabalho de análise de aplicações de *streaming*. Os autores citam como desafiador, o problema de provisionamento de recursos no ambiente de nuvem, eles consideram que, no contexto de aplicações de *streaming*, os dados são gerados por fontes externas, logo, o objetivo é cuidadosamente alocar recursos de modo que a taxa de transformação possa igualar a velocidade de chegada de dados.

No trabalho é apresentada uma solução que, segundo os autores, pode lidar com taxas de dados inesperados, incluindo as taxas transitórias. Eles propuseram um algoritmo que pode lidar com padrões dinâmicos nas chegadas de dados. O algoritmo evita qualquer abrandamento no processamento de aplicações, ao mesmo tempo, conservando os orçamentos de recursos.

O estudo é experimental e foi realizado com dois fluxos de algoritmos de mineração de dados mais populares. Como resultado, observou-se que o algoritmo converge para a alocação de CPU ideal com base na taxa de chegada de dados e necessidades computacionais.

O algoritmo identifica as condições de sobre-fluxo e sob-fluxo e através de estatísticas de carga são observados os ajustes feitos de CPU, para chegar à repartição ótima. Os autores consideraram o algoritmo como eficaz mesmo se houver mudanças significativas nas taxas de chegada de dados e, neste caso, a vantagem da abordagem é que o sistema

pode automaticamente otimizar-se com base nas necessidades de recursos.

O trabalho tem muita similaridade com o aqui proposto, entretanto trata de um problema específico de aplicação de *streaming*, não privilegiando outros tipos de aplicação, ou a definição se o recurso é ideal para a aplicação que se deseja executar, ou ainda definir quais os tipos de aplicações que podem concorrer.

No trabalho de Jörg Schad et al. [72] há a apresentação de um modelo de cálculo da variação de desempenho com recursos coletados através do uso de benchmarks. Os autores citam que, em geral, para esse tipo de análise são usadas fórmulas estatísticas como faixas, desvio-padrão e outros.

Eles afirmam que o desvio padrão é uma medida amplamente utilizada de variância, mas é difícil comparar para diferentes medidas. Quando um determinado valor é lido só pode indicar o quão alta ou baixa é a variação em relação a um único valor médio. O estudo apresentado envolve a comparação de diferentes escalas. Por fim, consideraram o coeficiente de variação (COV), definido como a razão entre o desvio padrão e a média. No trabalho aqui exposto há uma variação desta ideia com o uso de normalização para comparar valores de escalas diferentes.

Nedeljko Vasic et al. [99] consideram que a gestão eficaz dos recursos de ambientes virtualizados é uma tarefa desafiadora. Eles consideram que os sistemas de gestão, precisam contar com modelos analíticos ou avaliar a alocação de recursos, executando experimentos reais. No entanto, ambas as abordagens incorrem em uma sobrecarga significativa uma vez que há alterações de carga de trabalho.

No trabalho, propõem-se minimizar a sobrecarga de gerenciamento de recursos através da identificação de um pequeno conjunto de classes de carga de trabalho para tomada de decisões e assim: Adaptar rapidamente às mudanças de carga de trabalho usando assina-

turas e cache as alocações de recursos preferenciais em tempo de execução; E lidar com a interferência, estimando um "índice de interferência".

Os autores avaliaram DejaVu executando serviços de rede representativos no Amazon EC2 e ao permitir uma adaptação rápida, DejaVu economizou até 60% do custo de provisionamento de serviços.

Bradley Simmons et al. [82] propõem a criação de um sistema de tomada de decisão baseado em SLAs para agregação de recursos de forma otimizada. Na proposta há um controle de uso de recursos que pontua os gastos e compara com os níveis de serviços propostos, penalizando a carga excessiva. O trabalho baseia-se no cálculo de consumo, mas refere-se à plataforma como um serviço, não monitorando a carga como um todo e por processos em específico, nem avaliam os diferentes tipos de aplicações que podem concorrer por um mesmo recurso.

Outro trabalho sobre escalabilidade na nuvem é descrito por Zhiming Shen et al.[78] onde define-se que um dimensionamento de recursos de forma elástica permite que se atinja os objetivos mínimos de SLA's. No artigo foi descrita a criação de um sistema chamado CloudScale que controla a elasticidade da infraestrutura de uma nuvem.

Os autores citam que CloudScale pode prever demandas e fazer predição de erros, possibilitando a readaptação dos recursos ao mesmo tempo em que executa, sem sobrecarregar o sistema com o próprio software, como acontece em muitas aplicações de gerência de nuvem. Segundo eles, a carga de CPU imposta pelo CloudScale é da ordem de 2% em um cluster virtualizado.

No que diz respeito à camada de virtualização, muitos trabalhos têm se dedicado a analisar seus efeitos no desempenho dos aplicativos em execução, especialmente sobre com o uso da computação na nuvem em apoio à HPCD.

Estes trabalhos são importantes para HPCD como uma solução alternativa para a computação científica na nuvem, sendo capazes de fornecer computação sob demanda *just-in-time*, podendo de alguma forma alterar o padrão de uso de ambientes HPCD e sua estrutura, mesmo com alguma perda de desempenho [46] [95] [96] [16].

O trabalho de Bientinesi et al. [10], aprofunda o conhecimento e aponta para as diversas demandas que os ambientes de nuvem enfrentam para executar aplicações científicas e comerciais, usando a nuvem para suportar HPCD, onde é necessário rever as métricas utilizadas como uma forma de avaliar o desempenho do ambiente. Os problemas de contenção existentes podem afetar o desempenho geral, especialmente a comunicação.

O trabalho trata os resultados de uma série de testes para análise de aspectos, além de comunicação, tais como: a variabilidade do meio, os contextos de troca dos núcleos, o tipo de arquitetura do processador, a máquina virtual e, particularmente, o tipo de aplicação que está sendo executada. A mesma conclusão sobre a influência da comunicação, contenção e tipo de aplicação é observada em [36] e [27]. Em [26] há uma análise detalhada da perda resultante pelos efeitos de contenção em um ambiente virtualizado e a influência do tipo de tráfego.

Já com respeito à previsão do consumo de recursos e a afinidade de uma aplicação, define-se que não é tarefa simples, Armbrust et al. [4] abordam a dificuldade de previsão do uso destes recursos, mas não citam a afinidade em si. Segundo os autores, a taxa média de utilização dos servidores em um centro de computação convencional varia de 5 a 20%, e em alguns casos, dependendo do tipo de serviço utilizado, podem ocorrer picos de uso inesperados.

A proposta de avaliar continuamente os recursos e serviços pode ser usado para analisar esses picos, tentando prever a variação e consequentemente prever que o recurso é

suficiente para executar a aplicação em questão. Mesmo com a análise de consumo de recursos de servidores apresentada por Armbrust, grande parte do tempo os recursos ficam ociosos, mesmo assim, se houver aumento da carga além do esperado, haverá falta de recursos, e degradação do ambiente, acarretando perda da qualidade dos serviços prestados pela nuvem.

Já com respeito ao uso das classes de equivalência propostas por Dwarfs, também podem ser encontradas em algumas bibliotecas numéricas, tais como: The Fastest Fourier Transform in the West (FFTW) [32] uma biblioteca de software para computação da Transformada Discreta de Fourier (equivalente à Métodos Espectrais da classe de Dwarfs), LAPACK/ScaLAPACK [11] biblioteca de software para álgebra linear numérica (equivalente à classe Dwarf DLA) e OSKI [104] uma coleção de primitivas que fornecem kernels computacionais sintonizados automaticamente em matrizes esparsas (classe Álgebra Linear Esparsa - SLA).

Os treze Dwarfs também estão relacionados com a classificação de computação da Intel em três categorias: Reconhecimento, de mineração e de síntese (RMS). As aplicações RMS são consideradas importantes para orientar novas pesquisas e desenvolvimento arquitetônico que compreende aplicações em Inteligência Artificial e Aprendizado de Máquina, bancos de dados, jogos e computação gráfica. Estes aplicativos são representados por diversas classes da Dwarfs, como DLA, SLA , métodos espectrais, *Backtrack* e *Branch Bound*, entre outros [5].

Além delas, Rodinia [17], Parboil [91], Torch [38] e projeto paralelo de Dwarfs [3] são suítes com aplicações implementadas com código fonte aberto para referência, baseados em um subconjunto dos 13 Dwarfs. As aplicações Rodínia são projetadas para as infraestruturas de computação heterogêneas, e usam OpenMP, OpenCL e CUDA para permitir comparações entre GPUs multi-core vs CPUs multi-core. Implementações do Parboil são

[3] http://paralleldwarfs.codeplex.com/

desenvolvida para GPU e algumas implementações básicas para CPU.

Já o projeto Torch identificou vários kernels para fins de *benchmarking* classificados de acordo com os 13 Dwarfs. Os autores discutem possíveis estratégias de otimização de código que podem ser aplicadas a estes. Os Dwarfs paralelos também adotam a classificação dos 13 Dwarfs para descrever o cálculo em cada um de seus *benchmarks* e correspondem a um conjunto de 13 kernels paralelizados utilizando várias tecnologias, tais como OpenMP, TPL e MPI.

Estes exemplos motivam a sua utilização como forma de categorizar aplicações científicas, tanto para a importância das bibliotecas e áreas de aplicação mencionadas, bem como para estas recentes suítes de desenvolvimento de referência, que abrangem novas arquiteturas. Isso pode indicar a relevância e a contemporaneidade dessas classes para comunidade científica.

Além disso, alguns trabalhos propõem caracterização das aplicações científicas para melhorar o desempenho no ambiente de computação em nuvem. Em [58], é demonstrada a criação de uma ferramenta de benchmark (Hawk-i) para investigar como diferentes algoritmos científicos escalares em diferentes instâncias do Amazon EC2 em nuvem com o uso de diferentes Dwarfs.

Os experimentos implementam dois desses padrões, métodos espectrais e N-corpo, além disso realizaram um estudo detalhado das diferentes instâncias da Amazon com a ajuda de Hawk-i para identificar a estabilidade do desempenho de cada Instância do EC2. Em [66] a investigação foi baseada no desafio que os consumidores de (IaaS) têm em determinar qual provedor de IaaS e recursos são mais adequados para executar um aplicativo que pode ter qualidade específica de serviço (QoS).

Para estes, eles querem ser capazes de prever o desempenho de um aplicativo dada

uma descrição geral do hardware fornecido pelo provedor de IaaS. No trabalho os Dwarfs foram usados para medir o desempenho do hardware virtualizado com a realização de experiências na *"BonFIRE"* da Amazon EC2. Eles mostram que o hardware diferente é melhor adaptado para diferentes tipos de computação, e assim, o desempenho relativo das aplicações varia entre hardwares diferentes.

Isso se reflete por Dwarfs que são sensíveis às diferenças entre as arquiteturas dos hosts físicos sobre as quais as máquinas virtuais foram implantadas, mesmo que as diferenças sejam, por vezes pequenas. Ao examinar as correlações entre diferentes Dwarfs eles demonstraram que há diferentes Dwarfs para medir diferentes aspectos do desempenho computacional. As diferentes correlações entre Dwarfs e aplicações sugerem que os Dwarfs são úteis para prever o desempenho do aplicativo, como parte de um modelo de aplicação.

Em [28] eles investigaram o uso de Dwarfs para caracterizar recursos computacionais, que são utilizados como entrada para um modelo de aplicação para prever o desempenho da aplicação dos recursos. Com base em uma investigação *"BonFIRE"* em cinco nuvens públicas, eles demonstraram que a caracterização de recursos de computação utilizando Dwarfs é bem sucedida para a modelagem de aplicações para prever o desempenho de duas aplicações multimídia e uma aplicação científica.

Em [50] o objetivo foi o de investigar quais as classes de aplicações científicas poderiam ser migradas para a nuvem. Eles usaram as classes de Dwarfs para medir o desempenho na arquitetura de GPUs virtualizadas e não virtualizadas usando diferentes linguagens de programação, hypervisores e matrizes.

Neste sentido, um exemplo da necessidade de informações deste tipo de teste, que é proposto no trabalho aqui apresentado, seria o conhecimento prévio se a hospedagem de um SGBD[4] em conjunto com um servidor Web, degradaria uma máquina real ou vir-

[4]Sistema Gerenciador de Banco de Dados

tual, bastando para isso definir qual tipo de aplicação classificada nos Dwarfs equivale um SGBD e um servidor Web.

Isso seria feito e validado através do uso de monitoramento contínuo de recursos e processos, verificando a carga de uso de memória e processamento. Esta entretanto é uma tarefa complexa e que requer alto grau de abstração dada a quantidade de recursos totalmente heterogêneos disponíveis em uma nuvem computacional, associado aos efeitos causados pelas aplicações que são executadas em conjunto. Goscinski [1] apresenta a dificuldade de monitorar esta carga do sistema na nuvem computacional.

Entretanto definir um mecanismo de controle e monitoramento do impacto de uma aplicação, para saber se um recurso suporta tal execução não é uma tarefa de fácil implementação como citado por Goscinski [1].

Dror Feitelson e Larry Rudolph [31] demostram um mecanismo de controle deste tipo de tarefa sendo executada em um ambiente de programação paralelo, calculando o custo da aplicação, mas não mostram a degradação de determinados recursos causados pela aplicação no recurso real, por exemplo.

A relação aplicação/ambiente de nuvem é tão importante que o projeto Magellan [106] financiado pelo Departamento de Energia dos EUA (DOE) [64] e pelo Instituto de Pesquisas Avançadas de Computação Científica (ASCR) [63] visa investigar o papel da computação na nuvem para cargas de trabalho científicas, dado que estes possuem uma demanda diferenciada das aplicações comerciais, que podem ser vistas na seção 2.1.

A conclusão aponta para a necessidade de adaptação das atuais aplicações; e mais importante ainda é que há aplicações que podem ser portadas para nuvens computacionais e outras cuja perda de desempenho não indicaria o seu uso.

Um trabalho que mostra o interesse na economia de recursos é definido por Sudha Mani e Shrisha Rao [51], onde apresentam um modelo de economia de uso de recursos geograficamente distribuídos, com a coleta de estatísticas de uso e consumo de energia demandado por carga excessiva dos recursos.

Ainda segundo o artigo, os custos de energia em todo o mundo variam dinamicamente, o que reforça a necessidade de escalonamento de aplicações e também de controle do tipo de recurso é ideal para que tipo de aplicação. No modelo, apresentam a existência de padrões de carga que são semelhantes para os diferentes tipos de servidores, e que decisões de escalonamento são feitas considerando essas cargas e os custos operacionais dos servidores, e desta forma, os pedidos de recursos são programados para rodar em servidores que operam a baixo custo, que também tem carga esperada baixa.

Até a presente data na área de pesquisa de afinidade de recursos em julho de 2014, poucos trabalhos foram encontrados que tratam do tema proposto nessa tese de doutorado, talvez o mais relevante é apresentado por Jason Sonnek e Abhishek Chandra [86].

No artigo é argumentado que os provedores de nuvem podem reduzir significativamente os custos operacionais e melhorar o desempenho do aplicativo hospedado, pela contabilidade de afinidades e conflitos entre máquinas virtuais. O trabalho mostra como a afinidade pode ser inferida em máquinas virtuais e como é possível prever o consumo de recursos físicos para alcançar um maior consolidação e desempenho de determinado aplicativo em um ambiente de nuvem.

Considerando o compartilhamento de recursos, quando máquinas virtuais são hospedadas em um mesmo ambiente real, e nestes ambientes são executadas aplicações concorrentes, há a necessidade de verificar o grau de degradação causada por esta concorrência. Isso sem dúvida agravaria o problema de desempenho das aplicações concorrentes. O *ebook* da CA Technologies [93] apresenta alguns testes e comentários sobre este problema.

Basicamente, para efeitos deste trabalho, seria possível utilizar 3 tipos de afinidade de forma independente ou juntas. Customização, desempenho isolado de aplicação, diferenciação de QoS e persistência, como descrito por Julia Schroeter et. al em [73].

No artigo os autores citam a existência de milhares de instâncias de aplicativos executadas por centenas de usuários simultaneamente, o que ocorreria na dificuldade de controle dessa infraestrutura. Mostram ainda que a afinidade define como as solicitações dos diferentes usuários são vinculadas a nós de processamento.

Diferentemente do cenário normal, no qual cada aplicação é escalonada de acordo com a existência de recursos, afinidade baseia-se em atributos específicos para o encaminhamento de solicitações específicas do usuário. A Figura 3.2 apresenta os tipos de afinidade comuns em clusters.

Todos os trabalhos anteriormente apresentados apontam para a lacuna ainda existente e a necessidade de aprofundar os estudos dos efeitos da concorrência pelo compartilhamento de um ambiente real por vários ambientes virtualizados lá hospedados. Mesmo assim, eles não citam a necessidade de fazer este estudo com a utilização do conceito de classes de aplicações.

3.5 Conclusão do Capítulo

Este capítulo teve como principal valor bibliográfico a apresentação dos conceitos iniciais e das abordagens que estão sendo desenvolvidas dentro do ambiente de nuvens computacionais, coube principalmente nesse sentido:

Assim, com base no que foi possível investigar até o momento, ainda há uma lacuna na avaliação dos efeitos da concorrência em ambientes virtualizados e como minimizar

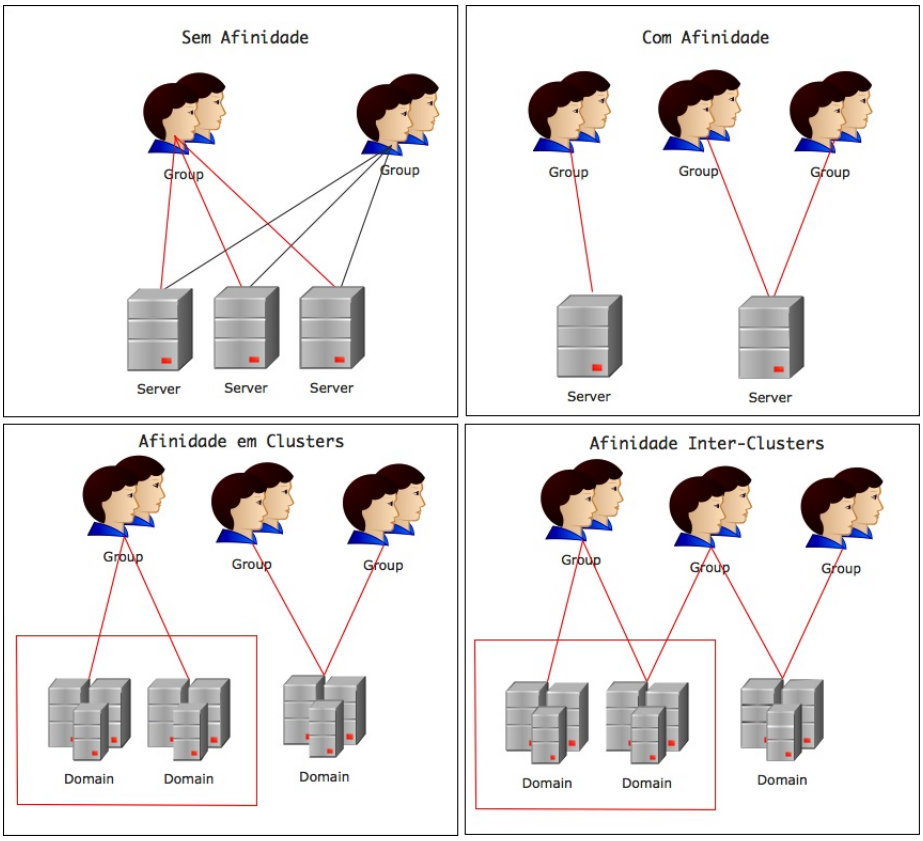

Figura 3.2: Tipos de Afinidade.

esses efeitos. Este trabalho propõe desta forma uma abordagem baseada na classificação de Dwarfs, avaliando os efeitos da concorrência entre as classes de Dwarfs, com foco na identificação das classes hospedadas em ambientes virtualizados, que poderiam compartilhar o mesmo ambiente real, com o mínimo de perda de desempenho.

Também são avaliados o tipo de biblioteca de programação paralela e o efeito destas bibliotecas na implementação como na concorrência entre as aplicações implementadas com mesma biblioteca e com bibliotecas diferentes. Foram usadas para avaliação OpenMP e OpenCL.

O trabalho proposto aqui precisa ser demonstrado sobre a infraestrutura de nuvens computacionais e esse é o principal motivo da explanação desse conceito, até o nível de escalabilidade e virtualização do ambiente. Como a apresentação do tema é baseado em afinidade entre recursos de hardware e os softwares executados nessa infraestrutura, faz-

se necessário o entendimento desses conceitos e de como eles serão incluídos no trabalho proposto. Por fim, foi feita uma pesquisa bibliográfica, buscando principalmente trabalhos similares ao proposto, para justificar sua apresentação na tese de doutorado. Desta pesquisa tomou-se que até o momento, não foram encontrados trabalhos similares ao proposto.

CAPÍTULO 4

DESCRIÇÃO DA PROPOSTA

Vimos com base no estudo do estado da arte apresentado nos Capítulo 2 e 3 que há necessidade de conhecer quais os tipos de aplicações que podem conviver em um mesmo recurso físico e quais não podem concorrer pelos mesmos recursos, sob pena de degradar todo o ambiente real, e, consequentemente, os ambientes virtuais hospedados nestes ambientes reais.

Para poder então melhor distribuir as aplicações em ambientes compartilhados, já que estes são comuns em ambientes de nuvens, torna-se necessário estudar o seu comportamento nestes ambientes e ser capaz de mensurar os impactos causados e sofridos por estas aplicações.

Para executar tal análise serão apresentados a seguir o escopo deste estudo, as assunções feitas acerca do comportamento das aplicações, a metodologia utilizada para os testes e a análise e o ambiente utilizado.

4.1 Problema

O primeiro passo tomado neste estudo foi formalizar a definição do problema e a sua delimitação. Foi observado nos trabalhos já realizados até o momento, que a camada de virtualização utilizada para uso em ambientes consolidados ou de nuvens computacionais causa uma degradação variável no desempenho das aplicações embarcadas nestes ambientes virtuais.

Entretanto estes estudos limitam-se a fazer a análise destes efeitos por meio de testes com diversas aplicações e pacotes de análise de desempenho, sem entretanto avaliar os

efeitos da concorrência entre os diversos tipos de aplicações e sobretudo sem considerar concomitantemente a contribuição que as bibliotecas que foram utilizadas no desenvolvimento destas aplicações poderiam ter no seu desempenho.

Para procurar cobrir essas lacunas os seguintes problemas para estudo foram definidos:

A perda causada pela concorrência que existe com uso de ambientes consolidados ou em nuvem computacional é dependente tanto da camada de virtualização, quanto das características das aplicações que concorrem por estes recursos.

Como estas características variam de aplicação para aplicação, a concorrência no uso dos recursos (processamento, memória, I/O) torna-se variável ao longo do tempo e dependendo da combinação de aplicações concorrentes em um dado intervalo de tempo, a degradação causada no ambiente também varia, podendo alcançar perdas significativas em termos de desempenho e consequentemente degradando a qualidade de serviço.

Torna-se necessário avaliar esta concorrência entre os diversos tipos de aplicações, avaliando a perda causada pelas diversas combinações entre elas, além de verificar se as bibliotecas utilizadas na implementação dos algoritmos que as compõem também influenciam o seu comportamento e consequentemente o seu desempenho.

Esta análise permitirá definir o grau de degradação causado pelas diversas combinações, considerando os tipos de aplicações e as bibliotecas utilizadas na sua implementação. Com isso será possível determinar quais combinações são possíveis de serem feitas, considerando um grau aceitável de perda no ambiente e em quais combinações a perda atinge um grau inaceitável face as garantias de qualidade de serviço.

4.2 Questionamentos da Pesquisa e hipóteses

Os problemas acima definidos nos remetem às seguintes questões que devem ser respondidas e as limitações que a avaliação apresentada neste estudo fará. As questões então formuladas para este estudo são:

- O grau de degradação causado pela concorrência no uso dos recursos é realmente diferenciado e dependente do tipo de aplicações combinadas neste recurso?
- O grau de degradação pelo efeito desta concorrência pode ser mensurado?
- É possível utilizar uma abordagem de classes de aplicações, de forma a facilitar a classificação prévia dos diversos tipos de aplicações existentes para avaliar esta degradação?
- É possível estabelecer um "Grau de Afinidade"entre estas classes de aplicações de forma a estabelecer quais as classes que podem concorrer por um mesmo recurso físico dentro de um grau de degradação aceitável e quais combinações devem ser evitadas?
- Os tipos de bibliotecas utilizados na implementação destas aplicações têm influência neste grau de degradação?

Para as perguntas acima formuladas, as seguintes hipóteses foram estabelecidas para verificação:

- O compartilhamento no uso de recursos pode causar perdas na execução das aplicações nele hospedadas e o grau de degradação observado é dependente da combinação das aplicações executadas concorrentemente.
- O uso de classes de aplicações é uma abordagem adequada para a determinação do grau de degradação causado pela concorrência de aplicações, cuja utilização servirá de base para uma pré-alocação destas aplicações.

- Há classes de aplicações que podem coexistir em um mesmo ambiente causando um nível de degradação aceitável, e há classes de aplicações, cuja concorrência pelos recursos causariam um nível de degradação, que deve ser evitado.
- O tipo de biblioteca utilizada na implementação dos algoritmos afeta esta combinação de classes.

Nos questionamentos e nas hipóteses anteriormente formuladas utiliza-se o termo *grau de perda aceitável considerando o nível de qualidade de serviços*, entretanto faz-se necessário ressaltar que este trabalho não terá como finalidade estabelecer especificamente este grau, mas sim avaliar e mensurar estas perdas, de forma a que os resultados aqui obtidos possam contribuir para outros estudos que visem utilizá-los na determinação destes graus.

Com a premissa de responder aos questionamentos esse trabalho busca criar um ambiente de teste, executando as aplicações da suíte Rodinia listadas na Seção 3.2, inicialmente em um ambiente livre de concorrência, tomando-se seus tempos de execução, para então comparar a mesma execução com os mesmos parâmetros iniciais, mas agora em um ambiente compartilhado, tomando seus tempos de execução e verificando a variação entre a concorrência e a não-concorrência.

4.3 Justificativa do Trabalho

O problema acima definido e as hipóteses formuladas, tiveram por objetivo avaliar o grau de degradação causado pela concorrência dos recursos quando de seu compartilhamento, por meio da combinação de diversas classes de aplicações, procurando cobrir uma lacuna ainda existente na análise destes ambientes.

A determinação dos graus de degradação causados pelas diversas combinações aqui avaliadas, considerando classes de aplicações e bibliotecas, possibilitará verificar a viabi-

lidade do uso do conceito de classes de aplicações como forma de introduzir e validar o conceito de *Afinidade* entre estas classes de aplicações e suas implementações.

Este estudo contribuirá assim para os esforços atualmente em andamento que buscam avaliar a melhoria no desempenho de ambientes consolidados ou de nuvem, sobretudo no tocante a sua utilização como recursos para execução de aplicações massivamente paralelas e distribuídas.

Do ponto de vista prático, os resultados deste estudo têm como contribuição o seu uso no desenvolvimento de escalonadores, capazes de utilizar o conceito de afinidade de forma a melhor distribuir os ambientes virtuais requisitados, e por conseguinte a qualidade de serviços nestes ambientes consolidados ou de nuvem.

Este estudo também traz um modelo inicial e uma proposta de escalonamento, apresentados na Seção 6.1 capaz de utilizar o conceito aqui tratado.

A análise do efeito do tipo de biblioteca utilizado na implementação das aplicações também procura contribuir para a definição de requisitos e para o desenvolvimento de novas aplicações em ambientes compartilhados.

4.4 Metodologia e Desenvolvimento

Nesta seção será apresentada a metodologia utilizada para a verificação das hipóteses apresentadas na Seção 4.2.

Para criação do ambiente de testes foram criados e configurados três servidores com processadores Xeon de 12 núcleos cada. Estes núcleos foram replicados na criação de 3 servidores virtuais em cada servidor real. Essa abordagem permitiu que os servidores virtuais compartilhassem os processadores reais igualmente.

O virtualizador utilizado foi o KVM sendo executado em cada servidor na versão 1.0 [1]. As definições de configuração dos servidores em ambiente real e virtual criados seguem abaixo:

4.4.1 Infraestrutura (Ambiente Real)

Para execução dos testes foram criados e configurados 3 servidores com as seguintes especificações:

1. Processador Intel(R) Xeon(R) CPU X5650 2.67GHz (12 núcleos)
2. 24 Gb de memória RAM.
3. Disco Rígido SATA de 500 Gb.
4. Sistema Operacional Ubuntu Server 12.04.

4.4.2 Infraestrutura (Ambiente Virtual)

Em cada servidor real foram criados 3 servidores virtuais que foram instanciados simultaneamente para os testes. Estes servidores foram configurados da seguinte maneira:

1. QEMU Virtual CPU version 1.0 (Processador Intel(R) Xeon(R) CPU X5650 2.67GHz (12 núcleos))
2. 4 Gb de memória RAM.
3. Disco Rígido SATA de 10 Gb.
4. Sistema Operacional Ubuntu Server 12.04.

[1] *QEMU emulator version 1.0 (qemu-kvm-1.0)*

4.4.3 Tipos de Testes de Concorrência Executados

Todos os servidores compartilharam os recursos de processamento em seus 12 núcleos para verificar a concorrência das aplicações. Os testes foram executados com as seguintes combinações de ambientes:

1. Ambiente Real com OpenMP;
2. Ambiente Real com OpenCL;
3. Ambiente Virtual com OpenMP;
4. Ambiente Virtual com OpenCL;
5. Ambiente Real com OpenMP X Ambiente Virtual com OpenMP;
6. Ambiente Real com OpenCL X Ambiente Virtual com OpenCL;
7. Ambiente Real com OpenMP X Ambiente Virtual com OpenCL;
8. Ambiente Virtual com OpenMP X Ambiente Real com OpenMP;
9. Ambiente Virtual com OpenCL X Ambiente Real com OpenCL;
10. Ambiente Virtual com OpenMP X Ambiente Real com OpenCL;
11. Ambiente Real com OpenMP X Ambiente Real com OpenMP;
12. Ambiente Real com OpenCL X Ambiente Real com OpenCL;
13. Ambiente Real com OpenMP X Ambiente Real com OpenCL;
14. Ambiente Virtual com OpenMP X Ambiente Virtual com OpenMP;
15. Ambiente Virtual com OpenCL X Ambiente Virtual com OpenCL;
16. Ambiente Virtual com OpenMP X Ambiente Virtual com OpenCL.

Para cada uma das combinações acima foram executados 20 testes com os 4 algoritmos citados na Seção 3.1, sendo computados os tempos de execução (individualmente e concorrentemente) e comparados os seus desempenhos. Os tempos de execução individuais foram utilizados como linha de base para a comparação com os demais testes.

Os parâmetros de entrada de cada algoritmo foram ajustados para que a execução de cada algoritmo fosse suficiente para que pudessem ser extraídos os tempos de processamento final. Não foram considerados tempos de entrada de dados em nenhum dos algoritmos, e para isso foram feitas mudanças nos códigos do Rodinia para que uma aplicação só fosse inicializada após toda a carga de dados e de forma manual.

Após as configurações de parâmetros de entrada adequados e ajustes dos códigos, todas as amostras foram tomadas com os mesmos dados de entrada, tanto nas execuções de linha de base (sem concorrência), quanto nas execuções concorrentes, assim, considerou-se que os testes foram executados sem comprometimento por testes tendenciosos.

É possível perceber que foram executados testes nos ambientes reais e virtuais, mesmo o foco do trabalho sendo os ambientes virtuais, consolidados em nuvens computacionais, o teste em ambientes reais se mostrou de interesse para demostrar que aplicações em ambientes reais podem degradar também os ambientes virtuais.

Outro ponto da abordagem no uso de testes em servidores reais é que foi possível criar uma base de conhecimento que pode ser utilizada para demonstrar quais as aplicações que podem se beneficiar de ambientes de nuvens.

Foi possível estabelecer assim, que alguns tipos de aplicações são melhores executadas em ambientes virtuais, que ambientes reais, quando há necessidade de compartilhamento destes recursos.

Vale também ressaltar que provedores de serviços de nuvens computacionais tendem a fornecer ambientes compartilhados, pois quando o usuário solicita um recurso, o que é oferecido é um servidor virtual, que na maioria das vezes, encontra-se hospedado em um servidor real compartilhado.

Essa abordagem e testes no ambiente real permite inclusive verificar que se no servidor real estiverem sendo executadas aplicações categorizadas nos tipos de dwarfs, a análise apresentará que tipos de servidores virtuais e suas aplicações não devem ser hospedadas nestes servidores.

4.5 Abordagem do Problema

Para o planejamento dos testes tomou-se por base as conclusões dos trabalhos apresentados na Seção 3.4. Entretanto como já citado, estes trabalhos nos conduzem para a necessidade de obter informações sobre o efeito da concorrência em ambientes compartilhados. Em toda a pesquisa realizada até o momento, muito foi feito para avaliar o impacto de ambientes virtuais em servidores reais, mas nenhum destes estudos dedicou-se ao efeito da concorrência causada pelas aplicações executadas nestes ambientes, e a contribuição do tipo de bibliotecas utilizadas na implementação destas aplicações.

E neste sentido, para que fosse possível avaliar o efeito do compartilhamento foram testadas quatro tipos de aplicações desenvolvidas com dois tipos de bibliotecas de programação paralela, OpenCL e OpenMP. As duas bibliotecas foram utilizadas por apresentarem boa documentação e por permitir testes de desempenho em diversas CPUs. Outro ponto importante é que esta abordagem permitiu o uso do Rodinia, *Benchmark* utilizado nos testes, que foi apresentado na Seção 3.2 e que possui implementação em OpenCL apresentado na Seção 3.2.1 e em OpenMP apresentado na Seção 3.2.2.

A versão da suíte Rodinia utilizada foi a 2.4 e as implementações dos algoritmos que

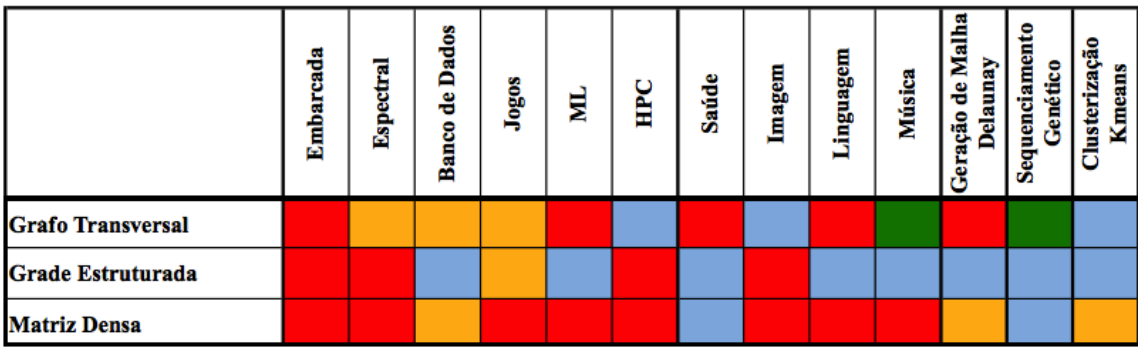

Figura 4.1: Destaque das classes de Dwarfs usadas neste trabalho.

foram utilizadas são implementadas em linguagem C++. O código fonte é aberto e precisou ser modificado para apresentar o tempo final de execução à partir do início do processamento, e não da carga de dados.

Outra modificação que precisou ser feita nos códigos dizem respeito à implementação com biblioteca OpenCL. Como os algoritmos para estas foi criado para ser executado em GPUs, o tamanho das matrizes eram limitados pela memória destas GPUs e para serem executadas em CPUs precisaram ser aumentadas para que a carga de processamento fosse maior.

Os testes com as duas bibliotecas (OpenCL e OpenMP) se mostraram necessários devido às diferenças encontradas nos tempos de execução quando usadas em um mesmo algoritmo.

Para os testes foram utilizados quatro *Benchmarks* do Rodinia que correspondem aos três tipos de classes. As classes de Dwarfs que foram escolhidas neste trabalho são: Álgebra Linear Densa (DLA), Grade Estruturada (SG) e Grafo Transversal (GT). A escolha por essas três classes foi porque há um grande número de aplicações científicas em diversas áreas, como foi apresentado na Figura 3.1 e em destaque na Figura 4.1. Além disso, o trabalho se concentra nestas três classes porque abrangem o maior conjunto de tipos de aplicações catalogadas pela abordagem dos Dwarfs.

4.5.1 Classes e Algoritmos Escolhidos

As seguintes classes de algoritmos foram escolhidas para testes de concorrência:

1. Álgebra Linear Densa é uma classe de Dwarf que envolve um conjunto de operadores matemáticos realizados em valores escalares, vetores ou matrizes, quando a maioria dos elementos da matriz ou vetor são diferentes de zeros. Densa neste Dwarf refere-se à estrutura de dados aceita durante a computação. A intensidade aritmética do cálculo operando os dados são de operadores de baixa intensidade (escalar por vetores, vetor-vetor, matriz-vetor, matriz-matriz, redução do vetor, vetor de digitalização e produto escalar) que carregam um número constante de operações aritméticas por elemento de dados. Tem uma razão elevada de operações matemáticas para carga e um elevado grau de interdependência entre segmentos de dados. Eles são a base de solucionadores mais sofisticados, como LU de decomposição (LUD) ou Cholesky e apresentam alta intensidade aritmética [38]. Aplicações classificados como DLA são relevantes através de uma variedade de domínios. Por exemplo, em ciência dos materiais para a física molecular e ciência em nanoescala; em garantia de energia para a combustão, fusão e energia nuclear; na ciência fundamental como astrofísica e física nuclear; no projeto de engenharia aerodinâmica. Algoritmos representativos desta classe são LUD, matriz transposta, matriz triangular, algoritmos de agrupamento, como K-means e Fluxo de cluster, e muitos outros. Os experimentos desta tese utilizaram algoritmos LUD e Kmeans.

   (a) LUD é um algoritmo para calcular as soluções de um conjunto de equações lineares que decompõe a matriz como o produto de uma matriz triangular inferior e uma matriz triangular superior para conseguir uma forma triangular, que pode ser utilizada para resolver um sistema de equações lineares facilmente. A matriz $A \in \mathbb{R}^{n \times n}$ tem uma fatoração LU *iff* todos os seus valores principais são não-zeros, ou seja, $det(A[1:k, 1:k]) \neq 0$ for $k = 1 : n - 1$.

(b) Kmeans é um algoritmo de agrupamento amplamente utilizado na mineração de dados, é um método que particiona n pontos que estão em espaço $d-$dimensional em k aglomerados. O algoritmo semeia k aglomerados inicialmente no centro e determina para cada ponto de dados o seu centro mais próximo, e então recalcula os novos centros como o meio de seus pontos atribuídos. Este processo de atribuição de pontos de dados e reajustar centros é repetido até que ele se estabilize.

2. Grafo Transversal é um tipo de Dwarf que deve atravessar um número de objetos em um gráfico e examinar as características desses objetos. Um gráfico ou uma rede são abstrações intuitivas e úteis para a análise de dados relacionais, onde as entidades singulares são representadas como vértices, e as interações entre elas são retratadas como bordas. Aos vértices e às bordas podem ser adicionalmente atribuídos atributos com base na informação que encapsulam. Tais algoritmos normalmente envolvem uma quantidade significativa de memória de acesso aleatório para pesquisas indiretas e pouca computação [38]. Domínios científicos que incluem aplicações importantes nesta classe são as de bioinformática (MUMmer), gráficos e pesquisa (BFS e B+Tree).

(a) B+Tree é uma árvore $n-$ária muitas vezes com grande número de filhos por nó. B+Tree consiste em uma raiz, nós internos e folhas. A raiz pode ser uma folha ou um nó com dois ou mais filhos. O valor principal de uma B+Tree é no armazenamento de dados para recuperação eficiente em um contexto de armazenamento orientados para o bloco. Isto se dá porque B+Tree tem alta "fanout"(número de apontadores para nós filhos em um nó, tipicamente na ordem de 100 ou mais), o que reduz o número de operações de E/S necessárias para obter um elemento da árvore. A ordem, ou fator de ramificação, b de uma B+Tree mede a capacidade de nós (ou seja, o número de nós filhos) para nós

internos da árvore. O número real de filhos para um nó, referido aqui como m, é limitada para nós internos de modo que $[b/2] \leq m \leq b$. Nós folha não tem filhos, mas são limitados, de modo que o número de chaves deve ser de pelo menos $[b/2]$ e no máximo $b-1$. Na situação em que a B+Tree está quase vazia, ela contém apenas um nó, que é um nó folha. A raiz é também a única folha, neste caso. A este nó é permitido ter uma chave, se necessário, e, no máximo, b

3. Grade Estruturada são algoritmos que servem para organizar dados em uma grade multidimensional regular, onde a computação se dá como uma série de atualizações desta grade. Para cada atualização da grade, todos os pontos são atualizados com valores a partir de uma pequena vizinhança em torno de cada ponto. A vizinhança está normalmente implícita nos dados e determinada pelo algoritmo. Devido ao seu paralelismo inerente e cálculo de natureza intensa, aplicações de grade estruturados são geralmente uma boa opção para as arquiteturas multi-core como GPU. Algoritmos de grade estruturada aparecem em muitos domínios científicos que são citados a seguir com respectivo exemplo de aplicação: imagens médicas (leucócitos, Parede Coração e filtro de partículas), simulações de física (*HotSpot*), processamento de imagem (*Speckle Reducing Anisotropic Diffusion*) e simulações biológicas (miócitos) [89].

   (a) Speckle Reducing Anisotropic Diffusion (SRAD) é uma aplicação de processamento de imagem para imagens de ultrassom e de radar. Ele reduz o ruído de uma imagem dada, mantendo suas características importantes. Além disso, cada elemento da grade estruturada representa um pixel da imagem.

4.5.2 Ambiente de Testes

O ambiente de testes necessita ser preciso afim de evitar que as avaliações sejam prejudicadas. Para conseguir uma base funcional é necessário que o sistema seja ajustado seguindo algumas métricas de testes, tanto de hardware quanto de software e desta forma criou-se uma estrutura livre de testes tendenciosos.

Para a análise de desempenho nos testes realizados foram colhidas 20 amostras, que se mostraram suficientes para gerar uma base de dados confiável. Este número foi conseguido após avaliar-se o intervalo de confiança[2] das amostras. Neste caso foi avaliado que, desconsiderando aquelas com tempos constantes, as que mostravam variações de tempo de execução nas mesmas comparações não eram significativas e ficavam em um intervalo próximo com 10 execuções. Assim, para se evitar conclusões incertas, optou-se por testar 20 vezes cada combinação de algoritmos, mesmo aquelas que possuíam tempos de execução constantes.

O intervalo de confiança se mostrou adequado para propor a quantidade de testes a serem feitos, por se tratar de uma fórmula largamente utilizada nos experimentos estatísticos, como apresentado em [2].

Entretanto, para que os testes fossem comprovados, algumas amostras foram executadas novamente, o que gerou 40 testes sem que fossem verificadas incertezas quanto aos testes, ou seja, os tempos continuaram dentro da margem de erro tolerável.

Mesmo assim, foram utilizados nas amostras o intervalo de confiança novamente, e esta fórmula utiliza tanto o desvio padrão quanto a média das amostras. Para todos os casos, as mesmas combinações foram usadas, com os mesmos dados de entrada, evitando assim que dados de entrada diferentes pudessem gerar resultados diferentes.

[2]Cálculo que apresenta o grau de confiança de amostras estatísticas.

Também houve o comprometimento em manter os parâmetros de entrada sempre iguais, além de monitorar o tempo de execução final e não o tempo de CPU, já que o item importante aqui é a verificação do aumento do tempo final de execução causado pela concorrência de aplicações em ambientes compartilhados.

Foram executados 4 tipos de algoritmos (B+Tree, Kmeans, LUD e SRAD) com combinações de 2 tipos de bibliotecas (OpenMP e OpenCL) em ambientes reais e virtuais. Inicialmente foram testados os algoritmos em um ambiente livre de concorrência para verificar os tempos de execução, afim de ter uma base de tempos para comparar a perda causada pela concorrência em relação à não concorrência.

Comparou-se então a concorrência com todos os 4 algoritmos implementados usando a biblioteca OpenMP em ambientes reais e virtuais, seguido de todos os 4 algoritmos implementados em OpenCL em ambientes reais e virtuais (considerou-se estes testes como homogêneos) e por fim testou-se a concorrência entre algoritmos implementados com as duas bibliotecas também executando a concorrência em ambientes reais e virtuais (considerou-se estes testes como heterogêneos).

Com todas as combinações possíveis e desconsiderando os testes iniciais de cada implementação, buscando a adequação dos testes, foram feitos 7120 testes, o que gerou uma base de conhecimento abrangente e confiável. Os resultados de todos estes testes são apresentados no Capítulo 5.

4.5.3 Definições e Metas dos Testes

Este trabalho tem o foco principal no teste de desempenho, mediante avaliação de tempo de execução quando há concorrência entre aplicações para o mesmo recurso físico. Qualquer variação dos requisitos estipulados pelos testes deve ser tomada como uma não-conformidade e deve ser tratada desta forma. Este é o motivo desta seção e fazem-se

necessárias as seguintes definições sobre testes:

- Capacidade - É a carga de trabalho total que um sistema pode manipular sem violar os critérios de aceitação.
- Investigação - Busca o recolhimento de informações relacionadas com a velocidade, escalabilidade e/ou características de estabilidade do sistema. A avaliação é frequentemente utilizada para provar ou refutar hipóteses sobre a causa de um ou mais problemas de desempenho observado.
- Latência - É uma medida da capacidade de resposta que representa o tempo que leva para completar a execução de um determinado processo. Pode-se também representar a soma de todas as latências de vários processos.
- Métricas - As métricas normalmente são obtidos através de testes de desempenho, que incluem a utilização do processador ou memória ao longo de um determinado tempo.
- Desempenho - Refere-se a informações relativas ao tempo de resposta de uma aplicação e/ou os níveis de utilização de recursos.
- Teste de Desempenho - É uma "investigação"feita para determinar e/ou validar a velocidade, escalabilidade e/ou características de estabilidade do sistema. Este teste é o superconjunto que contém todas as outras subcategorias de testes de desempenho.
- Teste de Unidade - É qualquer teste que visa as características de desempenho.
- Utilização - É a porcentagem de tempo que um recurso está ocupado. O percentual restante do tempo é considerado o tempo ocioso.

4.6 Normalização dos Dados

A necessidade de normalizar os resultados se deu devido a análise dos tempos observados, que se mostraram, apesar de mesma grandeza (da ordem de segundos, ou no máximo

minutos), serem diferentes quanto ao tempo de execução total, podendo um algoritmo executar em 50 segundos e outro em 8 minutos. Neste caso, houve a necessidade de executar o mesmo algoritmo (menor tempo de execução) diversas vezes para que houvesse concorrência durante toda a execução do teste (algoritmo com execução mais longa).

Com isso foi possível extrair o percentual de aumento na execução concorrente do algoritmo de tempo mais longo e do algoritmo de tempo mais curto. Mas isto não foi suficiente para comparar individualmente cada algoritmo, sendo necessário criar um mecanismo que "colocasse"todos os valores obtidos em uma mesma escala.

Neste sentido optou-se pelo uso da normalização de valores. Essa normalização tem importância para fornecer valores em uma escala pré-definida que pode ser utilizada por escalonadores de tarefas para predizer o melhor recurso para executar determinado tipo de algoritmo, quando for necessária a existência de concorrência.

Ressalta-se aqui, que pela definição de normalização, valores que forem normalizados dentro de suas propriedades e características, perdem a propriedade valoração, por exemplo, 1 litro, 1 Kg e 1 metro perdem sua escala e podem ser comparados um a um, como se pertencessem à mesma escala.

Para exemplificar melhor a normalização de dados publicada por estes autores em [30], optou-se por apresentar recursos coletados de ordem de grandeza variadas em um nível onde os valores poderão ser comparados mais facilmente, usou-se a Fórmula 4.1.

$$R = \frac{C - M2}{M1 - M2}(N1 - N2) + N2 \tag{4.1}$$

Onde:

R : é o valor normalizado de cada entrada.

C : é o valor bruto coletado, sem tratamento.

M1 : é o maior valor encontrado em todos os valores lidos.

M2 : é o menor valor encontrado em todos os valores lidos.

N1 : é o maior valor da escala que se deseja trabalhar.

N2 : é o menor valor da escala que se deseja trabalhar.

A maior vantagem do uso da Fórmula 4.1 é poder elevar ou reduzir a pontuação de tempos de acordo com a necessidade em cada medição e desta forma, já que existe a comparação entre todos os valores, a que tiver a maior pontuação, é consequentemente a que tem a maior perda. Repara-se que o valor tratado de cada recurso poderá ser somado, já que após o uso da fórmula, estarão na mesma ordem de grandeza. Nesse exemplo, considera-se que os valores terão escala entre 0 e 9, sendo 0 para a menor perda de desempenho e 9 para a maior perda de desempenho.

Tabela 4.1: Tabela de Dados Reais Versus Dados Normalizados

Algoritmo	**Tempo (segundos)**												**Média**
Real B+Tree	14	14	14	14	14	14	14	14	14	14	14	14	14,00
V. B+Tree x V. SRAD	221	218	220	223	219	219	221	225	223	217	221	218	220,40
R. B+Tree x V. LUD	21	21	22	20	20	21	21	22	21	21	21	21	21,00
V. B+Tree x R. SRAD	211	214	212	216	212	213	215	211	213	214	211	214	213,00
Algoritmo	**Tempo Normalizado**												**Média**
Real B+Tree	0	0	0	0	0	0	0	0	0	9	0	0	0,75
V. B+Tree x V. SRAD	4,5	7,875	5,625	2,25	6,75	6,75	4,5	0	2,25	9	4,5	7,875	5,15
R. B+Tree x V. LUD	4,5	4,5	0	9	9	4,5	4,5	0	4,5	4,5	4,5	4,5	4,50
V. B+Tree x R. SRAD	9	3,6	7,2	0	7,2	5,4	1,8	9	5,4	3,6	9	3,6	5,40

$$P = \frac{\sum_{k=1}^{r} x_k}{r} \tag{4.2}$$

Onde:

P é a pontuação obtida.

r é a quantidade de valores.

x_k é cada recurso já normalizado.

Para verificar a aplicação das fórmulas foram tomadas 4 amostras de tempos de execução e a média gerada por estes. É possível comparar na Tabela 3.1 as 4 primeiras linhas os valores (em segundos) da execução de alguns algoritmos. Percebe-se, por

exemplo, que a média obtida por "Real B+Tree"(14,05) tem uma ordem de valor bem menor que "V. B+Tree x V. SRAD"(220,6), não podendo ser comparados diretamente, entretanto, após normalizados, os valores aparecem como 2,25 e 4,95 respectivamente. Esta normalização permite atribuir valores de forma que um escalonador poderia definir as melhores combinações mediante uma escala de valores dentro de um intervalo pré-definido, gerados pela Fórmula 4.2.

Pode-se ainda, através da Fórmula 4.3 definir pesos para cada pontuação afim de dar prioridade para determinadas combinações, por exemplo, quando há implementações com OpenMP e OpenCL (maior ou menor peso de acordo com o tipo de biblioteca paralela para implementação).

As Fórmulas 4.1, 4.2 e 4.3 podem ainda ser utilizadas para outros tipos de dados, que referenciariam, capacidade de memória, disco, cpu, ou qualquer outro tipo de valor de grandezas diferentes, por exemplo, para uma aplicação que demanda maior quantidade de memória, poder-se-ia atribuir um peso maior para servidores com mais memória, esta entretanto é uma possibilidade a ser investigada em um novo projeto, não sendo contemplada por esta pesquisa.

$$P = \frac{\sum_{k=1}^{r} x_k p}{\sum_{k=1}^{r} p} \tag{4.3}$$

Onde:

P é a pontuação obtida por cada análise com os pesos de cada.

r é a quantidade de valores obtidos.

x_k é cada valor já normalizado.

p é o peso dado a cada item.

O uso da abordagem de normalização definida aqui será melhor apresentada nos resultados da Seção 5.4.

4.7 Conclusão do Capítulo

Este capítulo apresentou as metodologias utilizadas, bem como os testes efetuados afim de propor o conceito de afinidade e a avaliação dos critérios de coexistência de tipos de aplicações. No próximo capítulo serão apresentados os testes e os resultados obtidos à partir destas metodologias.

CAPÍTULO 5

RESULTADOS

Neste capítulo serão apresentados os resultados dos testes, considerando a concorrência entre os quatro algoritmos escolhidos. Embora o foco deste trabalho fosse o de avaliar os efeitos da concorrência ao consolidar ambientes virtualizados em um servidor real, também são apresentados os resultados considerando a concorrência em ambientes reais. Toma-se neste caso como referência a perda de desempenho das aplicações em execução quando o ambiente em que estão sendo executadas é compartilhado por outro ambiente executando outra aplicação.

Apesar do foco desta tese ser em nuvens computacionais e por este motivo a análise de máquinas virtuais, os resultados também apresentam o desempenho das máquinas reais por dois motivos, o primeiro com respeito à informações quanto ao desempenho e o impacto de aplicações sendo executadas nos ambientes reais, para monitoramento, por exemplo.

A segunda abordagem da análise de máquinas reais tem sentido na apresentação dos tipos de aplicações que podem se beneficiar de nuvens computacionais, por terem desempenho melhor em ambientes virtuais que em ambientes reais, quando estes são executados concorrentemente.

Foram consideradas para testes duas abordagens:

1. Testes Homogêneos - A concorrência é testada com algoritmos implementos com a mesma biblioteca (OpenCL versus OpenCL ou OpenMP versus OpenMP);

2. Testes Heterogêneos - A concorrência é testada entre os algoritmos implementados em OpenCL e OpenMP (cada algoritmo usando uma biblioteca).

5.1 Avaliação de Desempenho com Concorrência Homogênea

Esta seção apresentará o impacto da concorrência entre algoritmos usando biblioteca OpenCL ou OpenMP. Os resultados levam em consideração a concorrência entre dois algoritmos usando a mesma biblioteca e o comparativo entre a execução de uma ou outra biblioteca.

5.1.1 Avaliação Homogênea do Algoritmo LUD

A Figura 5.1 mostra os resultados obtidos com algoritmo LUD como linha de base, executado em um ambiente real com execução concorrente com outro algoritmo LUD ou Kmeans ou B+Tree ou SRAD em ambiente virtual ou real.

O ambiente real executando o algoritmo LUD é usado nesta demonstração para avaliar aplicações que poderiam degradar o próprio ambiente real onde estão hospedados os ambientes virtuais. Considerou-se esta abordagem de análise deste ambiente, já que quando há degradação do ambiente real causada pela concorrência do algoritmo LUD com outros algoritmos, todos os ambientes virtuais também sofrem o impacto desta degradação.

Na figura é possível perceber que, com exceção da execução simultânea do algoritmo B+Tree em ambiente real ou virtualizado, todos os outros algoritmos concorrentes implementados em OpenCL apresentaram menor perda, se comparado com OpenMP. Porém os melhores resultados foram obtidos com algoritmos B+Tree executando simultaneamente em um ambiente virtual (16% de perda) e em um ambiente real (perda de 41%).

As piores combinações aconteceram quando da execução de LUD em ambiente real com SRAD em ambiente virtual (perda de 143%), seguido pelo algoritmo LUD sendo executado em ambiente virtual (perda de 138%) e Kmeans virtual (137% de perda) implementados com bibliotecas OpenMP. Com relação às implementações usando OpenCL

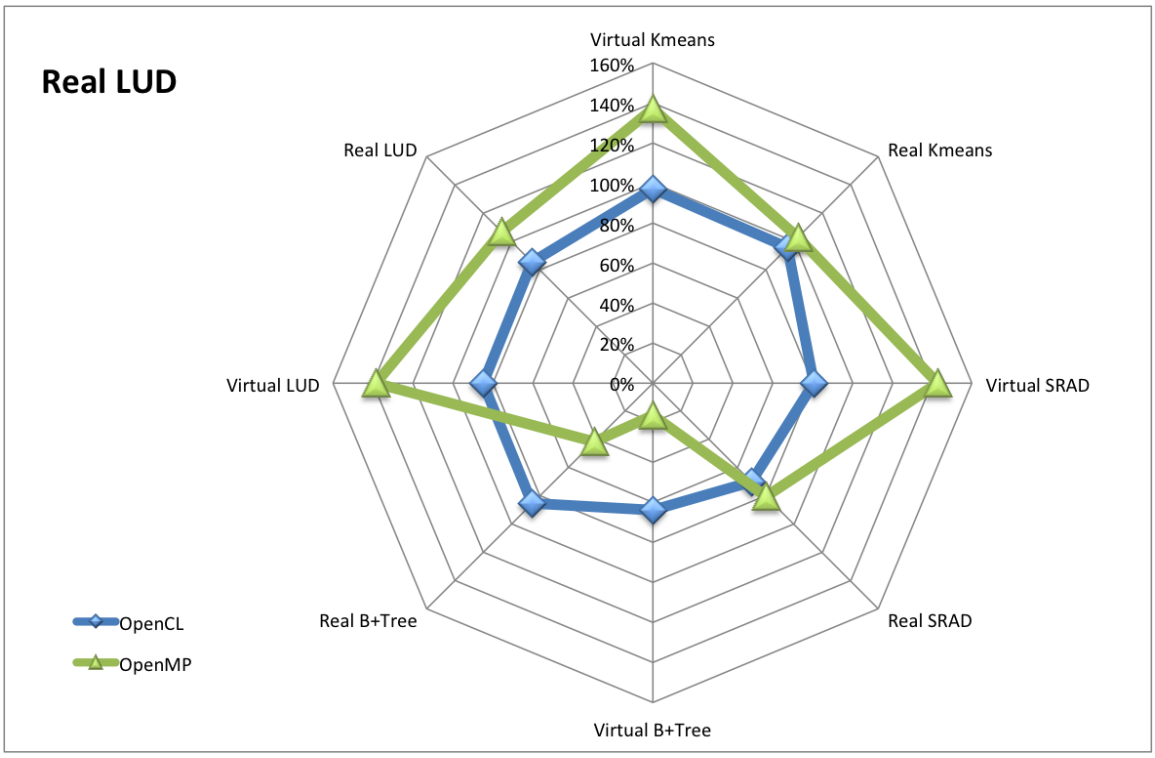

Figura 5.1: Valor de perda de desempenho do algoritmo LUD executado em um ambiente real, causada pela concorrência do LUD ou B+Tree ou Kmeans ou SRAD executando em um ambiente real ou virtual.

pode-se notar que a perda variou de 64% com B+Tree em ambiente virtual a 97% com Kmeans em ambiente virtual.

A Figura 5.2 mostra os resultados obtidos com algoritmo LUD como linha de base, executado em um ambiente virtual com execução concorrente de outro algoritmo LUD ou Kmeans ou B+Tree ou SRAD em ambiente virtual ou real.

Na figura também é possível perceber que, a exemplo do caso anterior, com exceção da execução simultânea do algoritmo B+Tree em ambiente real ou virtual, todos os outros algoritmos implementados com bibliotecas OpenCL são melhores do que com OpenMP, e os melhores resultados foram obtidos com SRAD sendo executado em ambiente real (11% de perda) e SRAD executado simultaneamente em ambiente virtual (perda de 21%) em OpenCL.

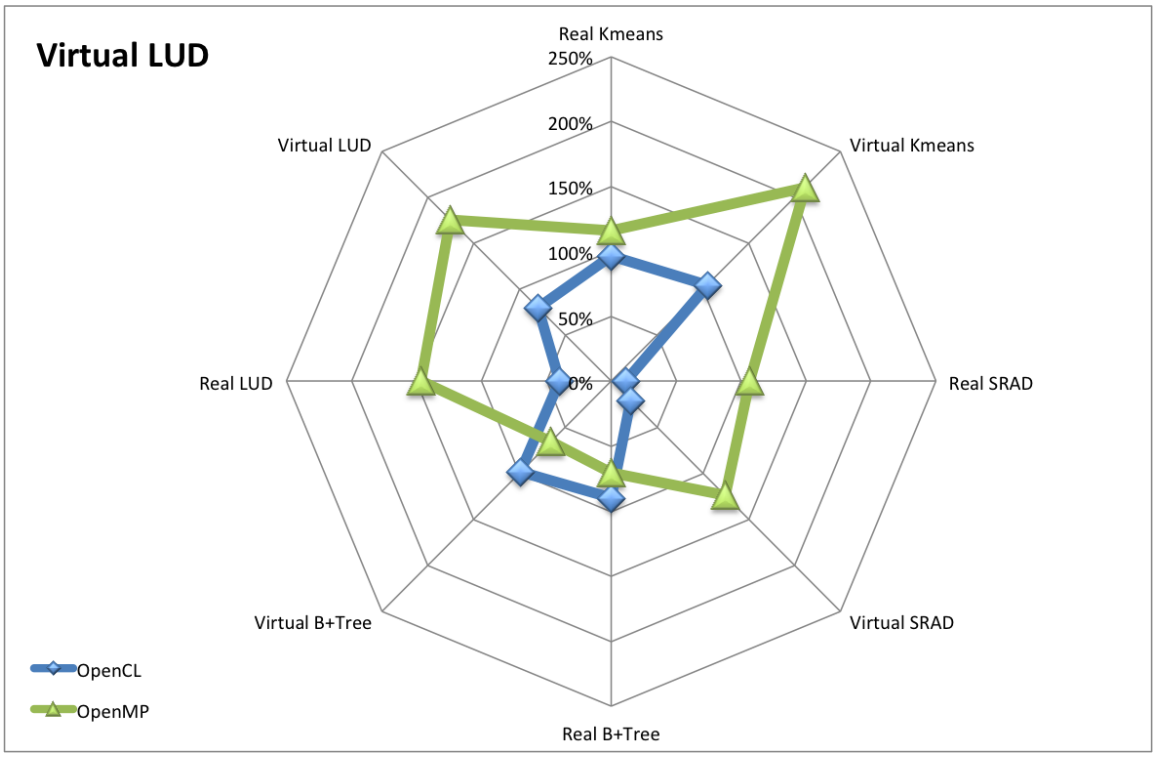

Figura 5.2: Valor de perda de desempenho do algoritmo LUD executado em um ambiente virtual, causada pela concorrência do LUD ou B+Tree ou Kmeans ou SRAD executando em um ambiente real ou virtual.

As piores combinações foram o algoritmo LUD executando em ambiente virtual com Kmeans em ambiente virtual (perda de 211%), seguido pelo algoritmo LUD em ambiente virtual (perda de 176%) e LUD em ambiente real (147% de perda) implementado com bibliotecas OpenMP.

Assim, os resultados apresentados nas Figuras 5.1 e 5.2 mostram que, se LUD estivesse sendo executado em um servidor real e houvesse a necessidade de compartilhar esse servidor com outro aplicativo, a melhor escolha seria a execução do algoritmo B+Tree (de preferência encapsulado em um ambiente virtual), sendo que a melhor biblioteca de programação seria OpenMP para este caso.

Para LUD sendo executado em um ambiente virtual, a melhor opção para compartilhamento de recursos seria a execução do algoritmo SRAD (de preferência no mesmo ambiente real), ambos implementados com bibliotecas OpenCL.

A conclusão que chega-se para aplicações com o algoritmo LUD é que a melhor opção é ter esta aplicação encapsulada em um ambiente virtual e compartilhar o ambiente real com uma aplicação SRAD em um ambiente real ou virtual, respectivamente, implementados bibliotecas OpenCL.

5.1.2 Avaliação Homogênea do Algoritmo B+Tree

A Figura 5.3 mostra os resultados obtidos com algoritmo B+Tree como linha de base, executado em um ambiente real com execução concorrente de outro algoritmo LUD ou Kmeans ou B+Tree ou SRAD em ambiente virtual ou real.

O ambiente real executando o algoritmo B+Tree é usado nesta demonstração para avaliar aplicações que poderiam degradar o próprio ambiente real onde estão hospedados os ambientes virtuais. Considerou-se esta abordagem de análise deste ambiente, já que quando há degradação do ambiente real causada pela concorrência do algoritmo B+Tree com outros algoritmos, todos os ambientes virtuais também sofrem o impacto desta degradação.

Na figura é possível perceber que, para um algoritmo B+Tree sendo executado em um ambiente real, com respeito a todas as possíveis combinações de algoritmos e ambientes, os melhores resultados foram obtidos com a implementação usando bibliotecas OpenMP executado concorrentemente com Kmeans em ambiente real ou virtual, com perda de 4% (apenas para o algoritmo B+Tree sendo executado concorrentemente com outro algoritmo B+Tree ou SRAD real, a perda de rendimento foi semelhante com o uso da biblioteca OpenCL).

Figura 5.4 mostra os resultados obtidos com algoritmo B+Tree como linha de base, executado em um ambiente virtual com execução concorrente de outro algoritmo LUD ou

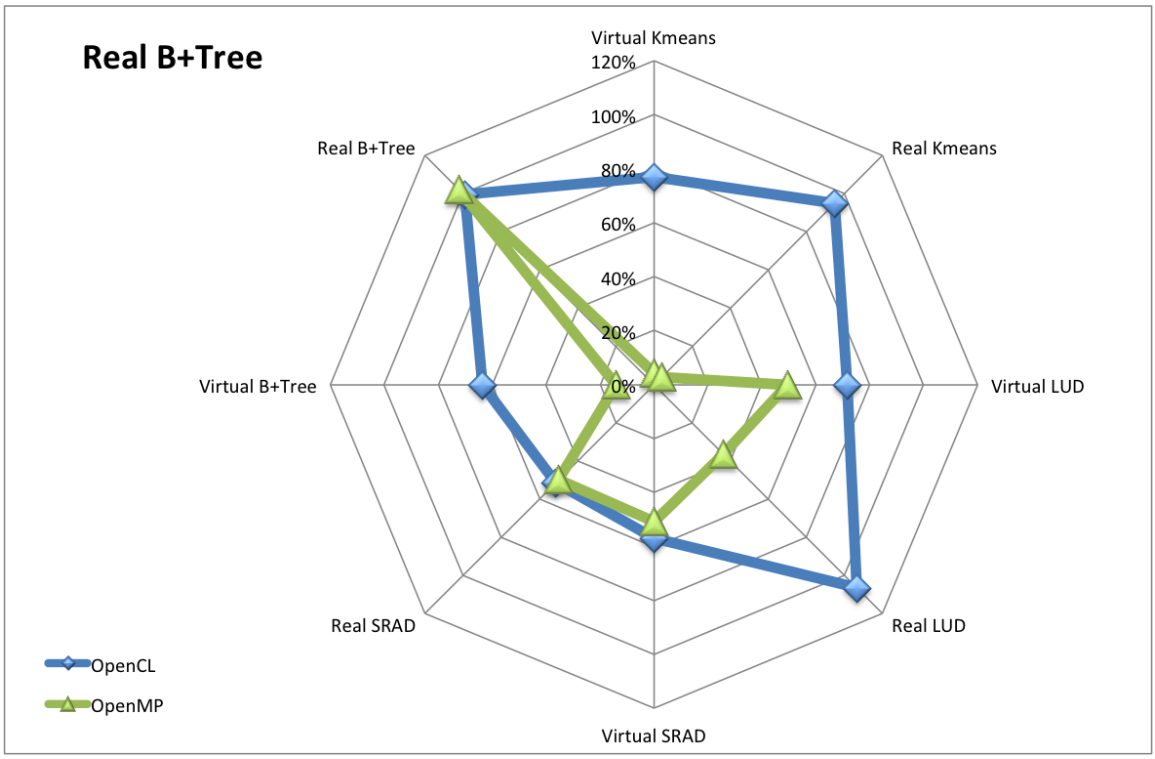

Figura 5.3: Valor de perda de desempenho do algoritmo B+Tree executado em um ambiente real, causada pela concorrência do LUD ou B+Tree ou Kmeans ou SRAD executando em um ambiente real ou virtual.

Kmeans ou B+Tree ou SRAD em ambiente virtual ou real.

A figura mostra que, para todos os tipos de algoritmos implementados com bibliotecas OpenCL, os resultados foram piores do que para os implementados com OpenMP, com exceção de algoritmo SRAD (Figura 5.5).

A Figura 5.5 mostra um detalhe da perda entre 0% a 20%. Pode-se perceber que o melhor resultado para a implementação com OpenCL foi obtida, na concorrência, quando o algoritmo SRAD é executado em ambiente real (perda de 5%) e SRAD em ambiente virtual (15% de perda).

É importante notar que, para todos os algoritmos implementados com bibliotecas OpenMP foram melhores que OpenCL, com exceção da concorrência com SRAD. O melhor caso ocorre com a concorrência com o algoritmo Kmeans em ambiente real e B+Tree

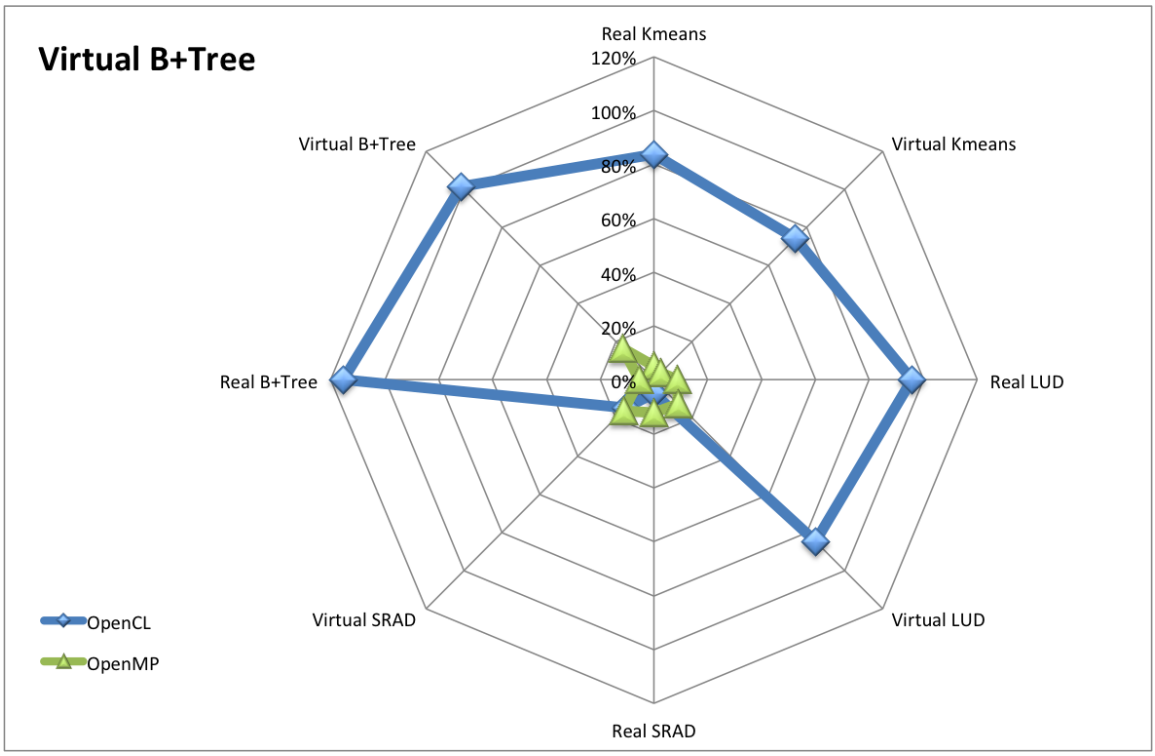

Figura 5.4: Valor de perda de desempenho do algoritmo B+Tree executado em um ambiente virtual, causada pela concorrência do LUD ou B+Tree ou Kmeans ou SRAD executando em um ambiente real ou virtual.

em ambiente virtual com perda de 5% e 4% respectivamente. No caso de implementações com OpenCL, somente pode haver concorrência com o algoritmo SRAD real ou virtual.

5.1.3 Avaliação Homogênea do Algoritmo Kmeans

A Figura 5.6 mostra os resultados obtidos com algoritmo Kmeans como linha de base, executado em um ambiente real com execução concorrente de outro algoritmo LUD ou Kmeans ou B+Tree ou SRAD em ambiente virtual ou real.

O ambiente real executando o algoritmo Kmeans é usado nesta demonstração para avaliar aplicações que poderiam degradar o próprio ambiente real onde estão hospedados os ambientes virtuais. Considerou-se esta abordagem de análise deste ambiente, já que quando há degradação do ambiente real causada pela concorrência do algoritmo Kmeans com outros algoritmos, todos os ambientes virtuais também sofrem o impacto desta de-

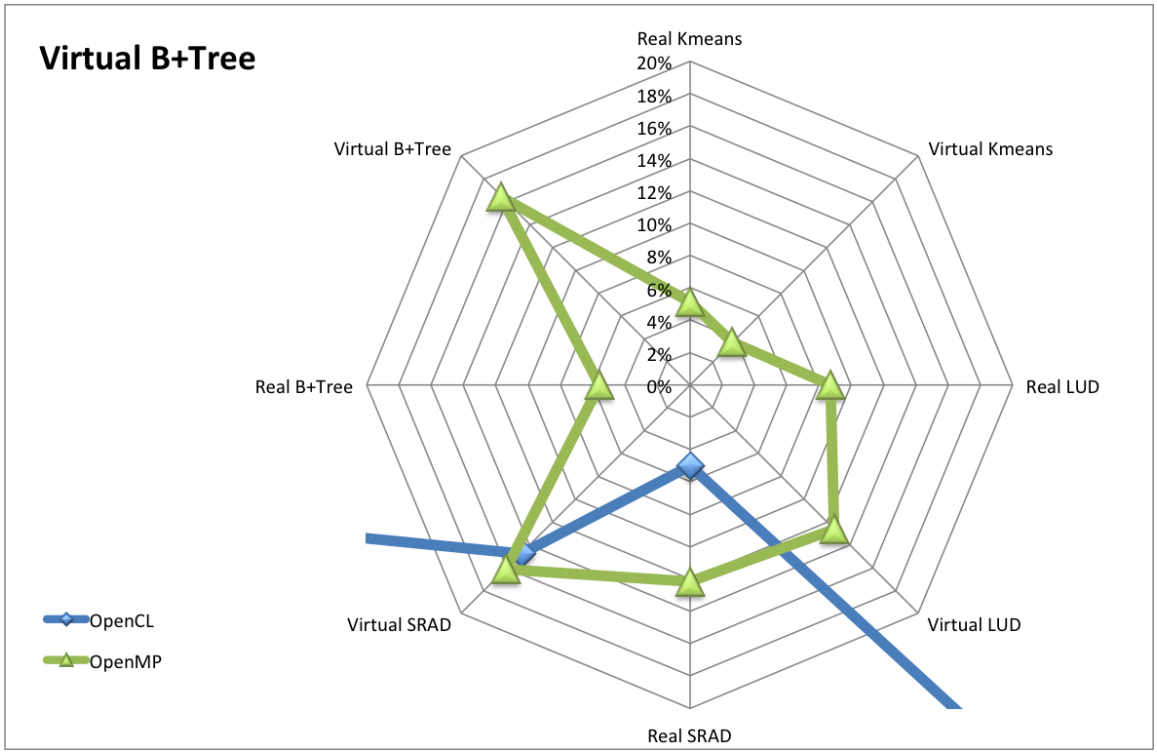

Figura 5.5: Detalhe no valor de perda de desempenho do algoritmo B+Tree executado em um ambiente virtual, causada pela concorrência do LUD ou B+Tree ou Kmeans ou SRAD executando em um ambiente real ou virtual.

gradação.

Na figura é possível perceber que, para um algoritmo Kmeans sendo executado em um ambiente real, com respeito a todas as possíveis combinações de algoritmos e ambientes, os melhores resultados foram obtidos com a implementação usando bibliotecas OpenMP, exceto para SRAD quando estes algoritmos são executados simultaneamente em ambiente real ou virtual e para LUD sendo executado em ambiente virtual, para estes três casos, os melhores resultados foram obtidos com os algoritmos implementados com biblioteca OpenCL. O melhor resultado foi obtido com uma perda de 4% para o compartilhamento de recursos com o algoritmo B+Tree em ambiente virtual e 8% de perda para o compartilhamento com o algoritmo B+Tree sendo executado em ambiente real (ambos com implementação OpenMP).

A Figura 5.7 mostra os resultados obtidos com algoritmo Kmeans como linha de base,

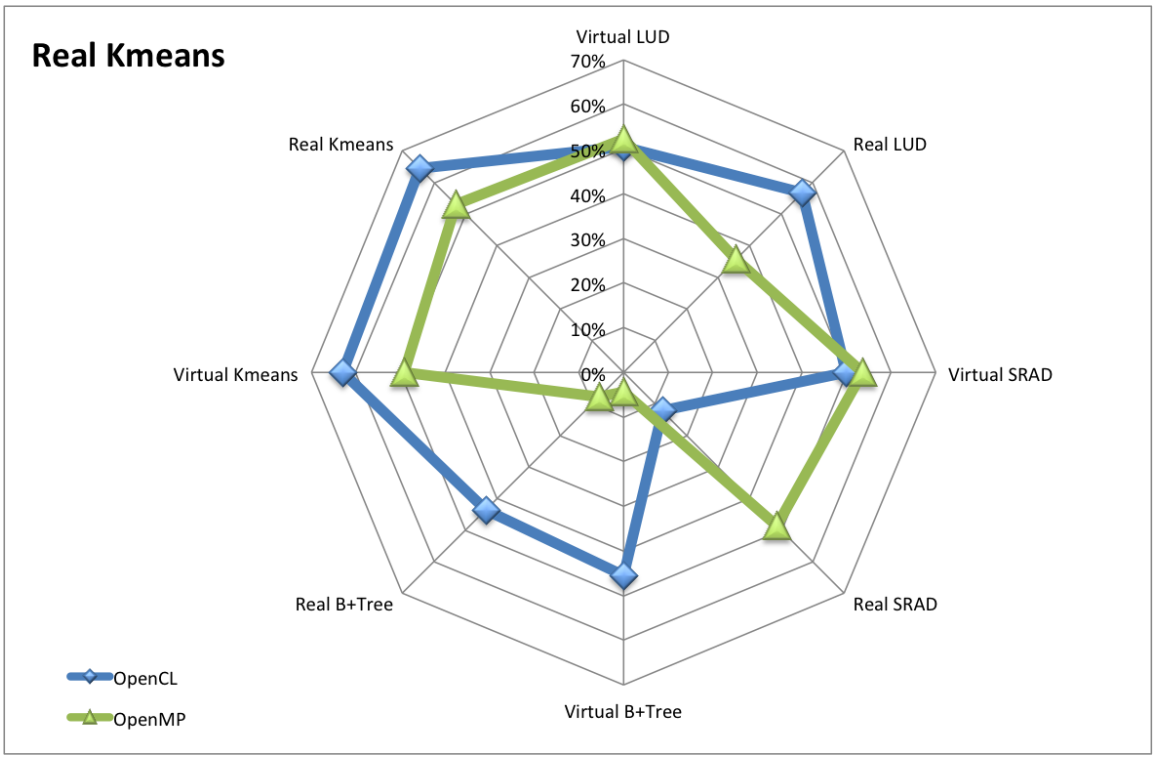

Figura 5.6: Valor de perda de desempenho do algoritmo Kmeans executado em um ambiente real, causada pela concorrência do LUD ou B+Tree ou Kmeans ou SRAD executando em um ambiente real ou virtual.

executado em um ambiente virtual com execução concorrente de outro algoritmo LUD ou Kmeans ou B+Tree ou SRAD em ambiente virtual ou real.

Na figura é possível perceber que, para um algoritmo Kmeans executado em um ambiente virtual, no que diz respeito a todas as combinações possíveis de algoritmos e ambientes, os melhores resultados foram obtidos com a implementação usando bibliotecas OpenMP, exceto para execução simultânea com algoritmo SRAD em ambiente real ou virtual e para o algoritmo Kmeans sendo executado em um ambiente real. Para estes três casos, os melhores resultados foram obtidos com os algoritmos implementados usando biblioteca OpenCL.

É importante ressaltar que os testes de comparação do algoritmo Kmeans como linha de base em ambientes virtuais ou algoritmo Kmeans como linha de base em ambiente real foram feitos após o carregamento dos dados. Isto se dá pois a carga de dados é mais

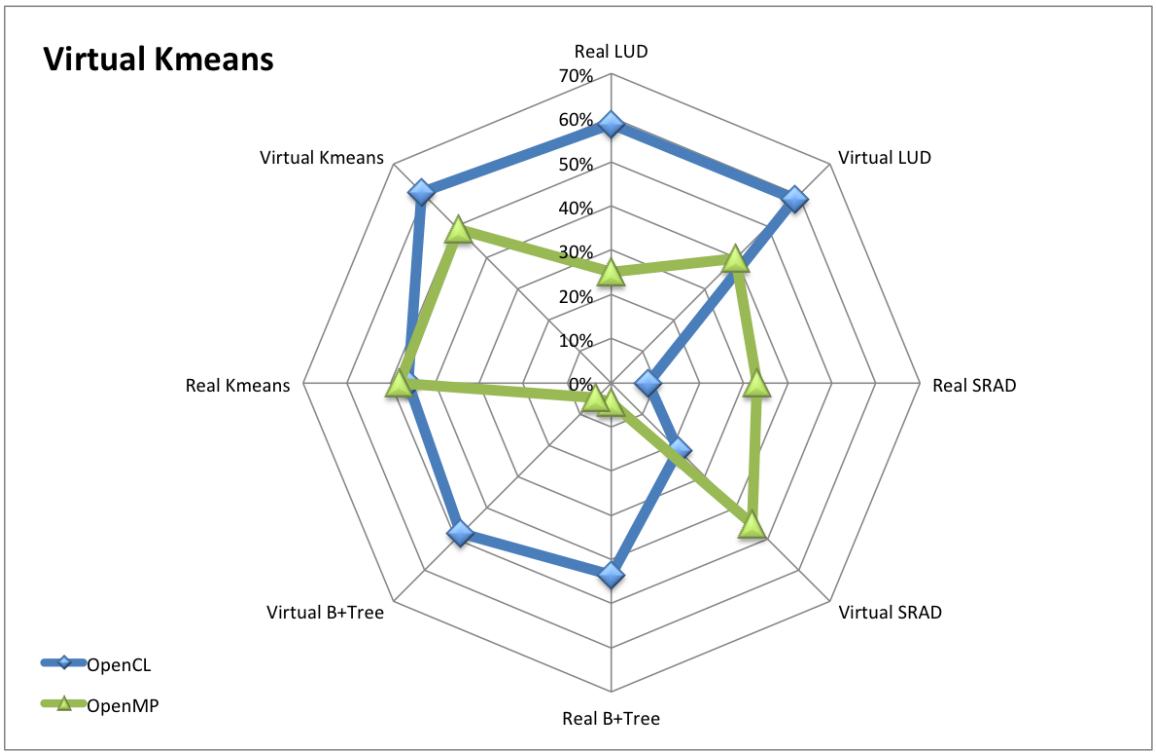

Figura 5.7: Valor de perda de desempenho do algoritmo Kmeans executado em um ambiente virtual, causada pela concorrência do LUD ou B+Tree ou Kmeans ou SRAD executando em um ambiente real ou virtual.

lenta no ambiente virtual que no ambiente real, além de que o foco desta avaliação foi no processamento e acesso à memória.

Os melhores resultados obtidos foram com Kmeans sendo executado em ambiente virtual, concorrentemente com B+Tree sendo executado em ambiente real, e Kmeans sendo executado em ambiente real, concorrentemente com B+Tree sendo executado em ambiente virtual, ambos com perda de 4% e implementados com biblioteca OpenMP.

A conclusão é que, para o algoritmo Kmeans em ambiente real ou virtual como base a melhor combinação é a concorrência de Kmeans com B+Tree, real ou virtual, usando bibliotecas OpenMP.

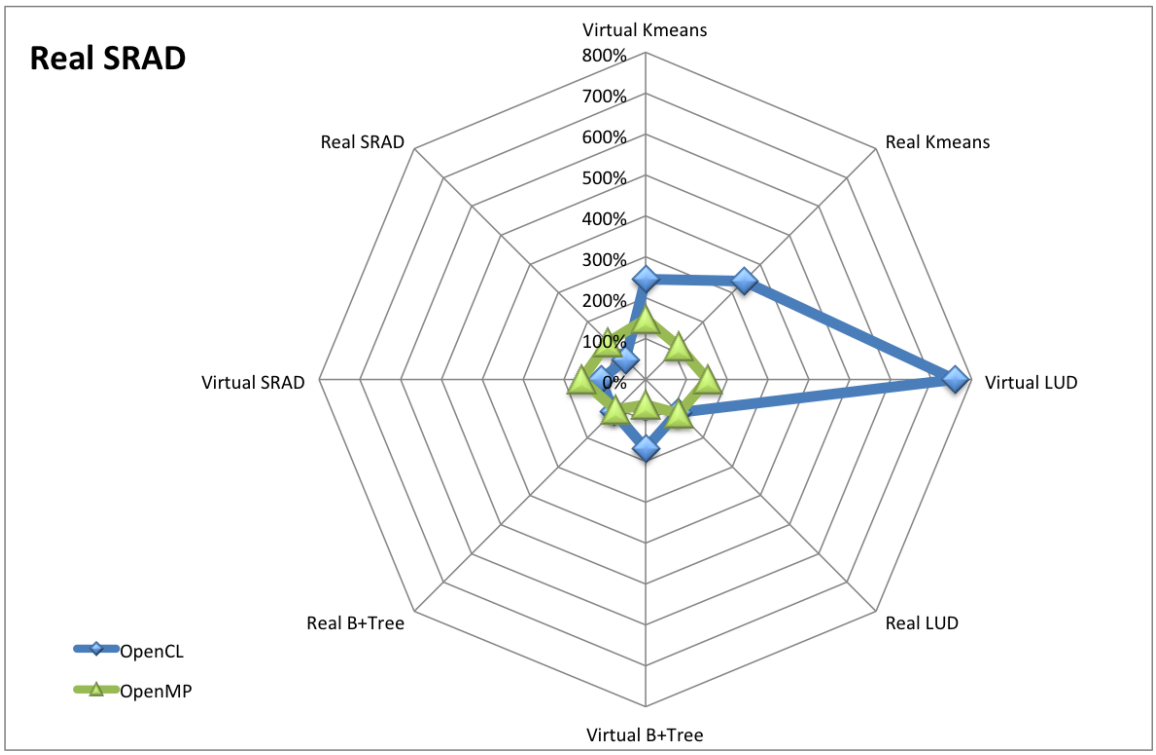

Figura 5.8: Valor de perda de desempenho do algoritmo SRAD executado em um ambiente real, causada pela concorrência do LUD ou B+Tree ou Kmeans ou SRAD executando em um ambiente real ou virtual.

5.1.4 Avaliação Homogênea do Algoritmo SRAD

A Figura 5.8 mostra os resultados obtidos com algoritmo SRAD como linha de base, executado em um ambiente real com execução concorrente de outro algoritmo LUD ou Kmeans ou B+Tree ou SRAD em ambiente virtual ou real.

O ambiente real executando o algoritmo SRAD é usado nesta demonstração para avaliar aplicações que poderiam degradar o próprio ambiente real onde estão hospedados os ambientes virtuais. Considerou-se esta abordagem de análise deste ambiente, já que quando há degradação do ambiente real causada pela concorrência do algoritmo SRAD com outros algoritmos, todos os ambientes virtuais também sofrem o impacto desta degradação.

Na figura é possível perceber que, para um algoritmo SRAD sendo executado em um ambiente real, com respeito a todas as possíveis combinações de algoritmos e ambientes,

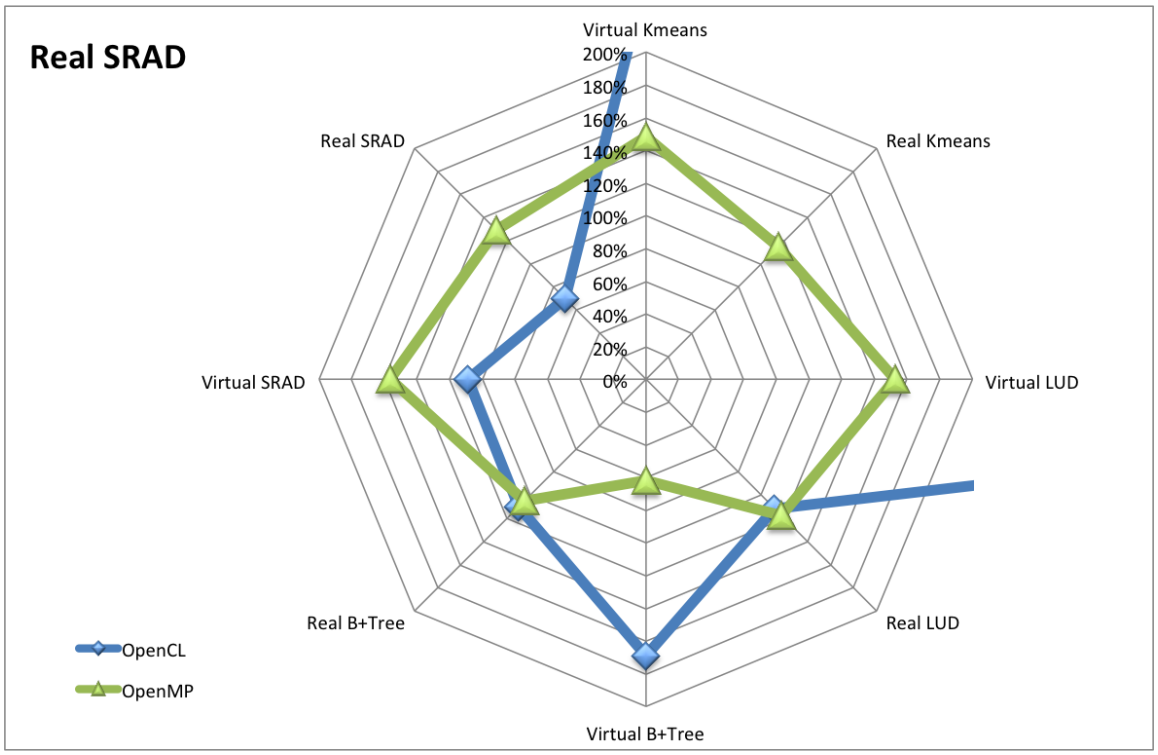

Figura 5.9: Detalhe no valor de perda de desempenho do algoritmo SRAD executado em um ambiente real, causada pela concorrência do LUD ou B+Tree ou Kmeans ou SRAD executando em um ambiente real ou virtual.

os melhores resultados foram obtidos com a implementação usando bibliotecas OpenMP, exceto para SRAD executado simultaneamente em ambiente real ou virtual ou para a LUD executado em um ambiente real (Figura 5.9), e para estes três casos, os melhores resultados foram obtidos com os algoritmos implementados usando OpenCL. O melhor resultado foi obtido com uma perda de 61% para o compartilhamento de recurso com o algoritmo B+Tree implementado com bibliotecas OpenMP executando em ambiente virtual, conforme detalhado na Figura 5.9. Neste teste pode ser visto a pior combinação de todos os testes realizados com algoritmos homogêneos: Algoritmo SRAD em ambiente real concorrendo com LUD em ambiente virtual com uma perda de 758% no desempenho.

A Figura 5.10 mostra os resultados obtidos com algoritmo SRAD como linha de base, executado em um ambiente virtual com execução concorrente de outro algoritmo LUD ou Kmeans ou B+Tree ou SRAD em ambiente virtual ou real.

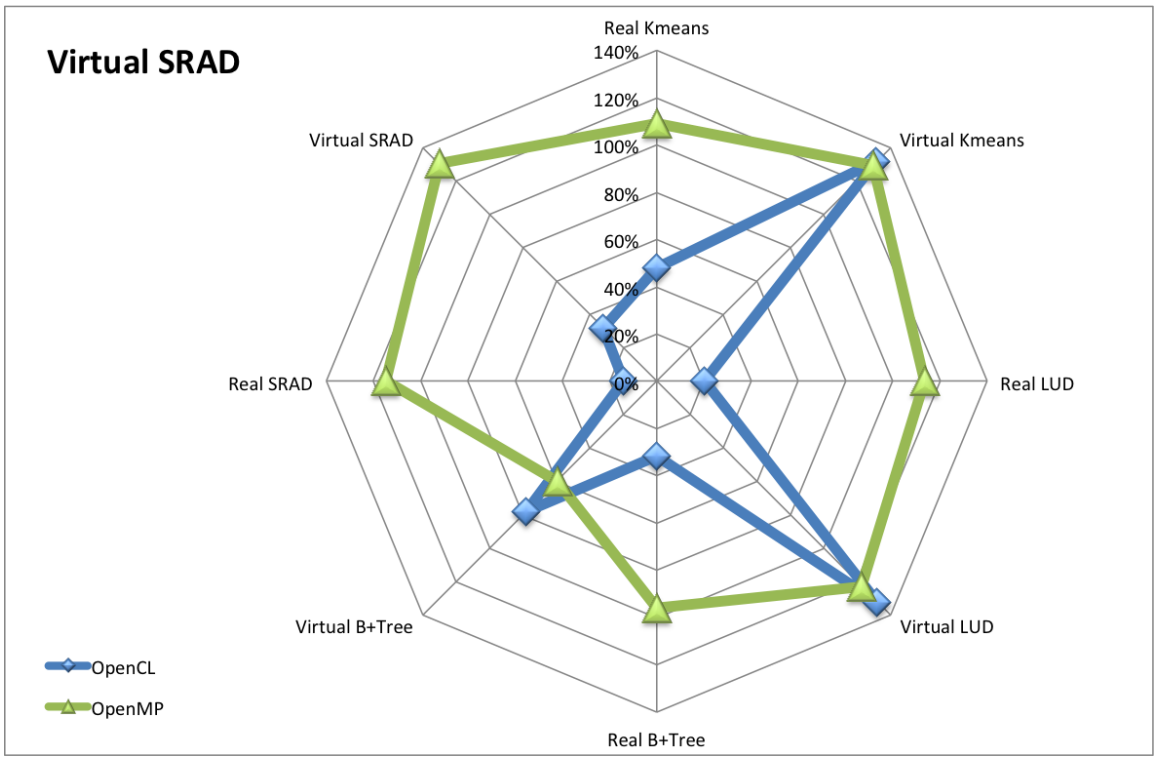

Figura 5.10: Valor de perda de desempenho do algoritmo SRAD executado em um ambiente virtual, causada pela concorrência do LUD ou B+Tree ou Kmeans ou SRAD executando em um ambiente real ou virtual.

Na figura é possível perceber que, para um algoritmo SRAD sendo executado em um ambiente virtual, no que diz respeito a todas as combinações possíveis de algoritmos e ambientes, os melhores resultados foram obtidos com a implementação usando bibliotecas OpenCL, exceto LUD executado simultaneamente em um ambiente virtual, Kmeans executado simultaneamente em um ambiente virtual ou B+Tree executado simultaneamente em um ambiente virtual, e para estes três casos, os melhores resultados foram obtidos com os algoritmos implementados usando biblioteca OpenMP.

O melhor resultado foi obtido com a perda de 14% para o compartilhamento com o algoritmo SRAD sendo executado em um ambiente real implementado com bibliotecas OpenCL.

A conclusão é que, para a execução do algoritmo SRAD real ou virtual como base a melhor combinação é a concorrência com SRAD (virtual ou real) usando bibliotecas OpenCL.

5.2 Avaliação de Desempenho com Concorrência Heterogênea

Esta parte dos testes avalia o impacto da concorrência entre algoritmos usando biblioteca OpenCL e OpenMP. Os resultados levam em consideração a concorrência entre dois algoritmos usando cada um uma biblioteca (de forma cruzada).

As avaliações de desempenho levam em consideração as perdas médias relativas em cada teste e a estabilidade na execução da concorrência, de forma a apresentar as melhores combinações e as piores perdas de desempenho.

Os valores apresentados permitem avaliar critérios de melhor e pior média entre os algoritmos com OpenCL competindo com OpenMP, que representam, o melhor e pior caso, respectivamente, e a distância entre pontos, as quais permitem avaliar a estabilidade do ambiente, onde, menor distância representa mais estabilidade e maior distância, menor estabilidade do ambiente.

Enquanto que a média representará um critério quantitativo, a distância entre os pontos representará um critério qualitativo do ambiente e dos algoritmos testados.

5.2.1 Avaliação Heterogênea do Algoritmo LUD

A Figura 5.11 mostra os resultados obtidos com algoritmo LUD como linha de base, executado em um ambiente real ou virtual com execução concorrente com outro algoritmo LUD em ambiente virtual ou real. Na figura a comparação ocorre conforme a ordem de aparecimento do texto: OpenCL, seguido de OpenMP. Exemplo: R.LUD x V.LUD o real está sendo executado com biblioteca OpenCL e o virtual com OpenMP.

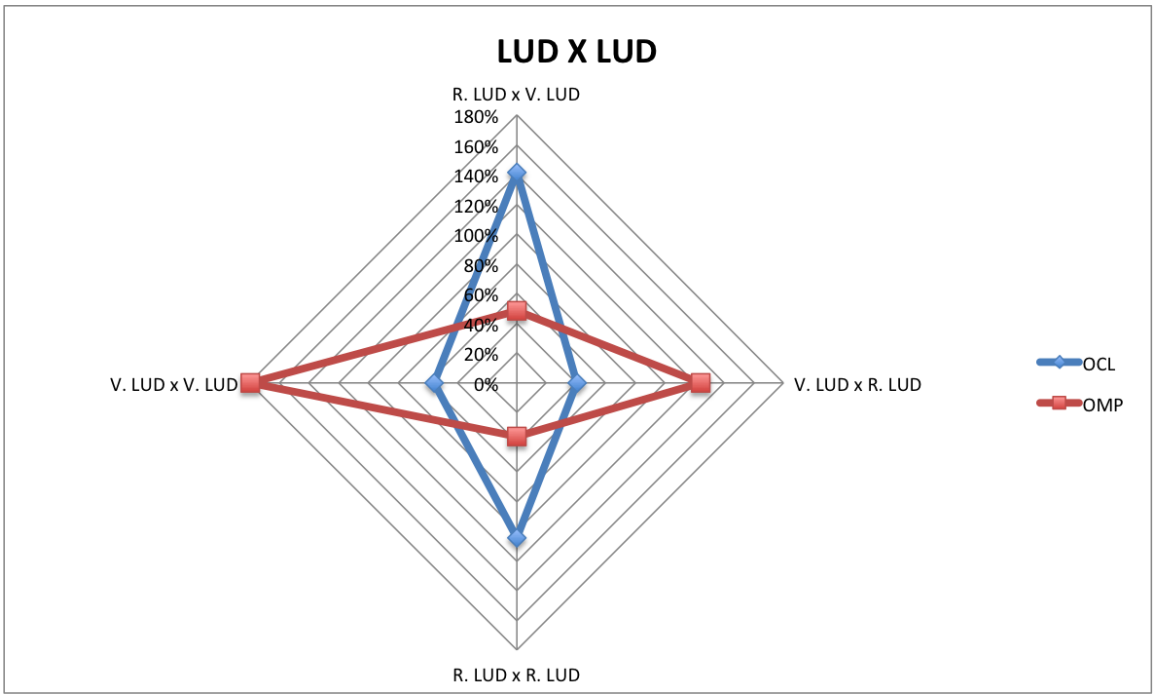

Figura 5.11: Valor de perda de desempenho do algoritmo LUD executado em um ambiente real ou virtual, causada pela concorrência do mesmo algoritmo LUD em um ambiente real ou virtual.

O ambiente real executando o algoritmo LUD é usado nesta demonstração para avaliar aplicações que poderiam degradar o próprio ambiente real onde estão hospedados os ambientes virtuais. Considerou-se esta abordagem de análise deste ambiente, já que quando há degradação do ambiente real causada pela concorrência do algoritmo LUD com outros algoritmos, todos os ambientes virtuais também sofrem o impacto desta degradação.

Na figura é possível perceber que, quando há concorrência do algoritmo LUD com bibliotecas OpenCL e OpenMP, avaliando-se os resultados por bibliotecas individualmente, o que está implementado com biblioteca OpenCL teve menor perda quando em ambientes virtuais.

A melhor combinação quantitativa, ou seja, com menor média, ocorre quando LUD é executado em ambiente real com biblioteca OpenCL, concorrendo com LUD em ambiente real com biblioteca OpenMP (105% e 37% de perda respectivamente), com média de 71%. Esta também é a melhor combinação com respeito à avaliação qualitativa do

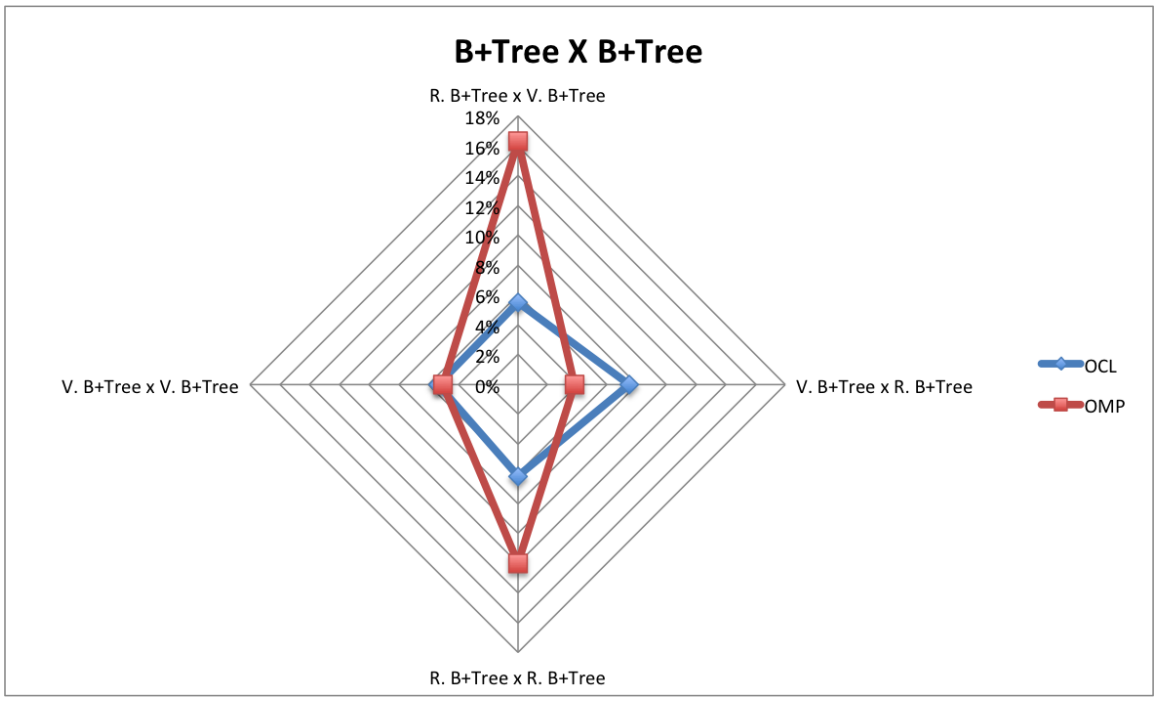

Figura 5.12: Valor de perda de desempenho do algoritmo B+Tree executado em um ambiente real ou virtual, causada pela concorrência do mesmo algoritmo B+Tree em um ambiente real ou virtual.

ambiente/algoritmo, com distância (diferença entre percentuais) de 68% entre os pontos.

Já para a avaliação do pior caso, ocorre quando há execução concorrente de LUD em ambiente virtual implementado com OpenCL com LUD em ambiente virtual implementado com OpenMP (55% e 179% de perda respectivamente). A média de perda de desempenho neste caso é de 117% e a distância entre os dois pontos é de 124%.

5.2.2 Avaliação Heterogênea do Algoritmo B+Tree

A Figura 5.12 mostra os resultados obtidos com algoritmo B+Tree como linha de base, executado em um ambiente real ou virtual com execução concorrente com outro algoritmo B+Tree em ambiente virtual ou real. Na figura a comparação ocorre conforme a ordem de aparecimento do texto: OpenCL, seguido de OpenMP. Exemplo: R.B+Tree x V.B+Tree o real está sendo executado com biblioteca OpenCL e o virtual com OpenMP.

Na figura é possível verificar que a melhor combinação na concorrência de B+Tree

com implementação OpenCL e OpenMP ocorre com execuções em ambientes virtuais, com perdas de desempenho iguais a 5% para ambos algoritmos e dado a isso, a distância entre os dois pontos é 0 (zero), fazendo deste ambiente a melhor combinação (menor média e mais estável).

Na figura é possível ainda perceber que o pior caso ocorre com a concorrência de B+Tree em ambiente real implementado com OpenCL e B+Tree em ambiente virtual implementado com OpenMP (perda de 5% e 16% respectivamente) com perda média e distância entre os pontos de 11%. Percebe-se que este tipo de concorrência do algoritmo B+Tree com bibliotecas OpenCL e OpenMP não traz grande influência em termos de perda.

5.2.3 Avaliação Heterogênea do Algoritmo Kmeans

A Figura 5.13 mostra os resultados obtidos com algoritmo Kmeans como linha de base, executado em um ambiente real ou virtual com execução concorrente com outro algoritmo Kmeans em ambiente virtual ou real. Na figura a comparação ocorre conforme a ordem de aparecimento do texto: OpenCL, seguido de OpenMP. Exemplo: R.Kmeans x V.Kmeans o real está sendo executado com biblioteca OpenCL e o virtual com OpenMP.

Na figura é possível perceber que os algoritmos implementados com OpenCL tiveram uma perda de desempenho menor, quando executados concorrentemente com OpenMP. A melhor combinação de perda de desempenho média, caso seja necessário haver concorrência destes algoritmos, seria a execução de Kmeans em ambiente real implementado com OpenCL e Kmeans sendo executado em ambiente virtual implementado com OpenMP (perda de 47% e 76% respectivamente), com perda média de 61%.

Já a menor distância entre os pontos, ou seja, o ambiente/algoritmo mais estável é a execução de OpenCL e OpenMP em ambientes virtuais (50% e 78% respectivamente),

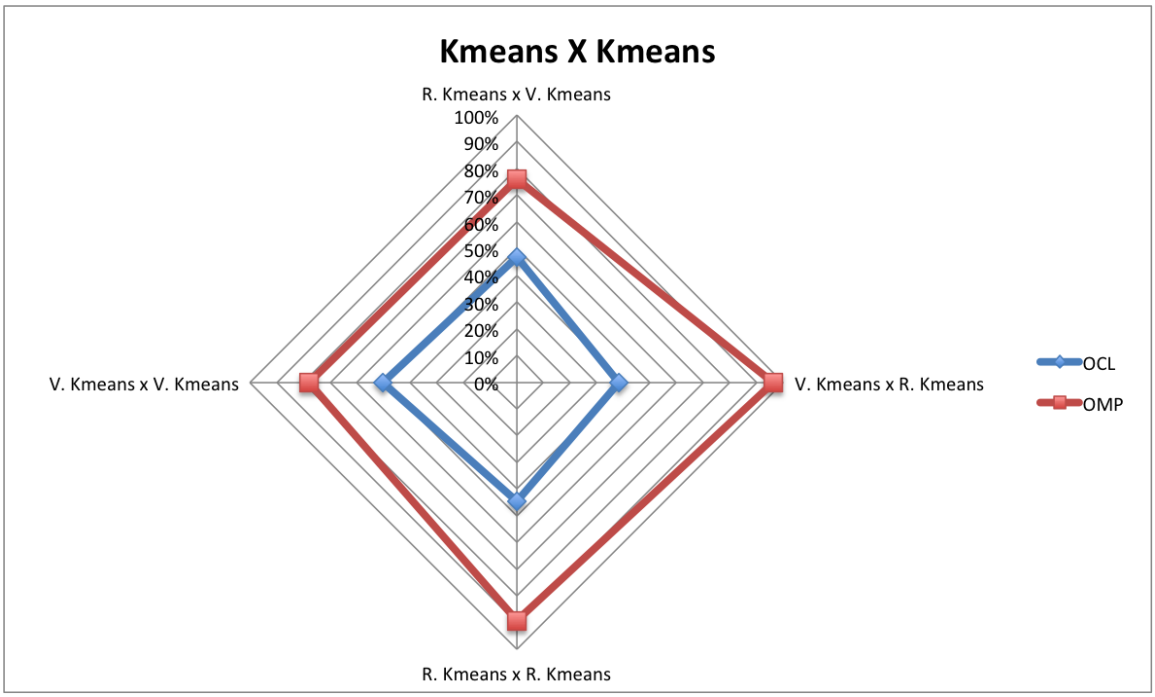

Figura 5.13: Valor de perda de desempenho do algoritmo Kmeans executado em um ambiente real ou virtual, causada pela concorrência do mesmo algoritmo Kmeans em um ambiente real ou virtual.

com distantes um do outro de 28%.

Considerando que a distância média do Kmeans sendo executado em ambiente real implementado com OpenCL concorrendo com Kmeans em ambiente virtual implementado com OpenMP foi de 29%, e a perda média de Kmeans executando em ambiente virtual implementado com OpenCL concorrendo com Kmeans executando em ambiente virtual implementado com OpenMP foi de 64%, a combinação mais adequada é Kmeans em ambiente real com OpenCL e Kmeans em ambiente virtual com OpenMP.

Nestes testes acontecem dois casos nos quais a média de perda do algoritmo Kmeans são piores, OpenCL em ambiente real com OpenMP em ambiente virtual e OpenCL em ambiente virtual com OpenMP em ambiente real, para os dois casos, a perda de desempenho média é de 67%. O mesmo não ocorre com a distância entre os pontos que mostra a estabilidade do ambiente/algoritmo, onde a combinação Kmeans em ambiente virtual implementado com OpenCL concorrendo com Kmeans em ambiente real implementado

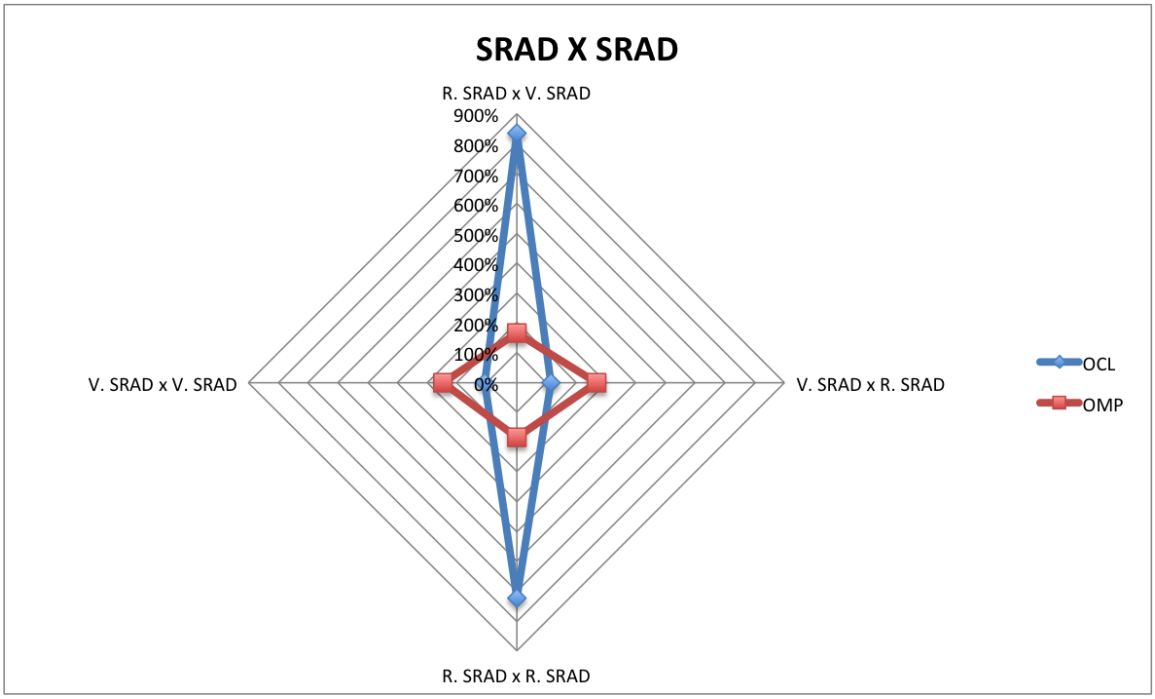

Figura 5.14: Valor de perda de desempenho do algoritmo SRAD executado em um ambiente real ou virtual, causada pela concorrência do mesmo algoritmo SRAD em um ambiente real ou virtual.

com OpenMP teve variação entre pontos de 58% (45% para a combinação inversa).

5.2.4 Avaliação Heterogênea do Algoritmo SRAD

A Figura 5.14 mostra os resultados obtidos com algoritmo SRAD como linha de base, executado em um ambiente real ou virtual com execução concorrente com outro algoritmo SRAD em ambiente virtual ou real. Na figura a comparação ocorre conforme a ordem de aparecimento do texto: OpenCL, seguido de OpenMP. Exemplo: R.Kmeans x V.Kmeans o real está sendo executado com biblioteca OpenCL e o virtual com OpenMP.

Na figura é possível perceber que o algoritmo SRAD implementado com biblioteca OpenMP é mais estável quando há execução concorrente com outro algoritmo SRAD implementado com biblioteca OpenCL, ambos sendo executados em ambientes virtuais, entretanto, nenhuma combinação deste algoritmo com bibliotecas OpenCL concorrendo com OpenMP deve existir, já que há uma perda média entre os algoritmos de 176%

para o melhor caso, quando SRAD é executado concorrentemente em ambientes virtuais, cada um algoritmo implementado como uma biblioteca (OpenCL com perda de 106% e OpenMP com perda de 247%). Esta combinação é descrita como a mais estável, com distância entre pontos igual a 141%.

Já com respeito ao pior caso, este ocorre com a execução concorrente de SRAD sendo executado em ambiente real com OpenCL e SRAD sendo executado em ambiente virtual com OpenMP (836% e 166% de perda respectivamente). Esta combinação acarreta perda média de 501% e é nela que há a maior instabilidade do ambiente, com distância entre pontos de 669%.

5.2.5 Avaliação Heterogênea do Algoritmo Kmeans X LUD

A Figura 5.15 mostra os resultados obtidos com algoritmo Kmeans ou LUD como linhas de base, executados em um ambiente real ou virtual com execução concorrente com outro algoritmo LUD ou Kmeans em ambiente virtual ou real. Na figura a comparação ocorre conforme a ordem de aparecimento do texto: OpenCL, seguido de OpenMP. Exemplo: R.Kmeans x V.LUD o real está sendo executado com biblioteca OpenCL e o virtual com OpenMP.

Na figura é possível verificar que o melhor caso, quando há necessidade de concorrência entre estes algoritmos, ocorre quando Kmeans é executado em ambiente real implementado com OpenCL executado concorrentemente com LUD em ambiente virtual implementado com OpenMP, apresentando perda média de 9% (sem perda para Kmeans e com perda de 19% para LUD), entretanto esta não é a combinação mais estável, que ocorre quando Kmeans é executado em ambiente real com OpenCL e LUD é executado em ambiente real com OpenMP (72% e 73% de perda respectivamente) com distância entre pontos de 1%.

Com base na perda média deste caso de 72%, apesar de ser mais estável, a melhor

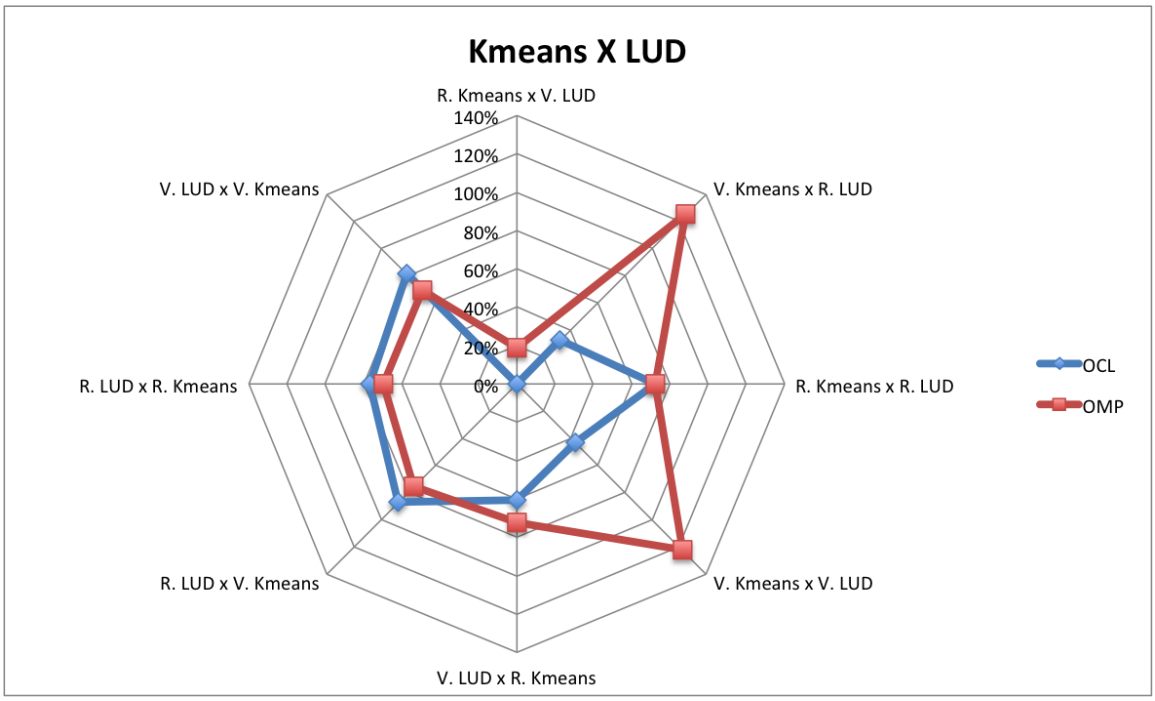

Figura 5.15: Valor de perda de desempenho do algoritmo Kmeans ou LUD executado em um ambiente real ou virtual, causada pela concorrência do algoritmo LUD ou Kmeans em um ambiente real ou virtual.

combinação é a apresentada no caso anterior, ou seja, Kmeans sendo executado em ambiente real implementado com biblioteca OpenCL, concorrendo com LUD executado em ambiente virtual implementado com biblioteca OpenMP, mesmo com maior instabilidade que o anterior (10%).

A pior combinação para a execução concorrente destes algoritmos ocorre com perda média de 83%, quando Kmeans é executado em ambiente virtual com OpenCL concorrentemente com LUD executado em ambiente virtual com OpenMP (43% e 123% de perda respectivamente). Já a combinação que se mostrou mais instável com distância entre pontos de 93% foi a concorrência de Kmeans sendo executando em ambiente virtual com OpenCL, com LUD sendo executado em ambiente real com OpenMP (32% e 125% de perda respectivamente).

Outro ponto de interesse relacionado à este teste ocorre com a comparação entre o algoritmo Kmeans executado em ambiente virtual e o algoritmo LUD executado em ambi-

ente real, repara-se na figura que quando LUD é implementado com biblioteca OpenCL e Kmeans com OpenMP, quando executados concorrentemente, há uma maior estabilidade do ambiente, com distância de 11% entre os pontos (LUD com perda de 87% e Kmeans com perda de 76%), entretanto quando LUD é implementado com biblioteca OpenMP e Kmeans com biblioteca OpenCL os valores ficam mais distantes um do outro 93% (LUD com perda de 125% e Kmeans com perda de 32%). Percebe-se ainda que quando os dois algoritmos são executados em ambientes virtuais, também há diferença entre os valores, o que mostra que o tipo de biblioteca também interfere na degradação do ambiente.

5.2.6 Avaliação Heterogênea do Algoritmo Kmeans X SRAD

A Figura 5.16 detalhada na Figura 5.17 mostra os resultados obtidos com algoritmo Kmeans ou SRAD como linhas de base, executados em um ambiente real ou virtual com execução concorrente com outro algoritmo SRAD ou Kmeans em ambiente virtual ou real. Na figura a comparação ocorre conforme a ordem de aparecimento do texto: OpenCL, seguido de OpenMP. Exemplo: R.Kmeans x V.SRAD o real está sendo executado com biblioteca OpenCL e o virtual com OpenMP.

Nas figuras é possível verificar que, quando há necessidade de compartilhar o meio físico entre os algoritmos Kmeans e SRAD implementados com biblioteca OpenCL e OpenMP, a melhor combinação ocorre com perda média de 46% quando SRAD é executado em ambiente virtual implementado com OpenCL e Kmeans é executado em ambiente real implementado com OpenMP (76% e 15% de perda respectivamente), sendo esta combinação a mais estável com distância entre pontos de 61%.

Já a pior combinação com perda média de 361% ocorre com a combinação de SRAD sendo executado em ambiente real com OpenCL concorrendo com Kmeans em ambiente real com OpenMP (721% e 0% de perda respectivamente), e esta foi também a combinação que teve a maior instabilidade no ambiente com distância entre pontos de 721%.

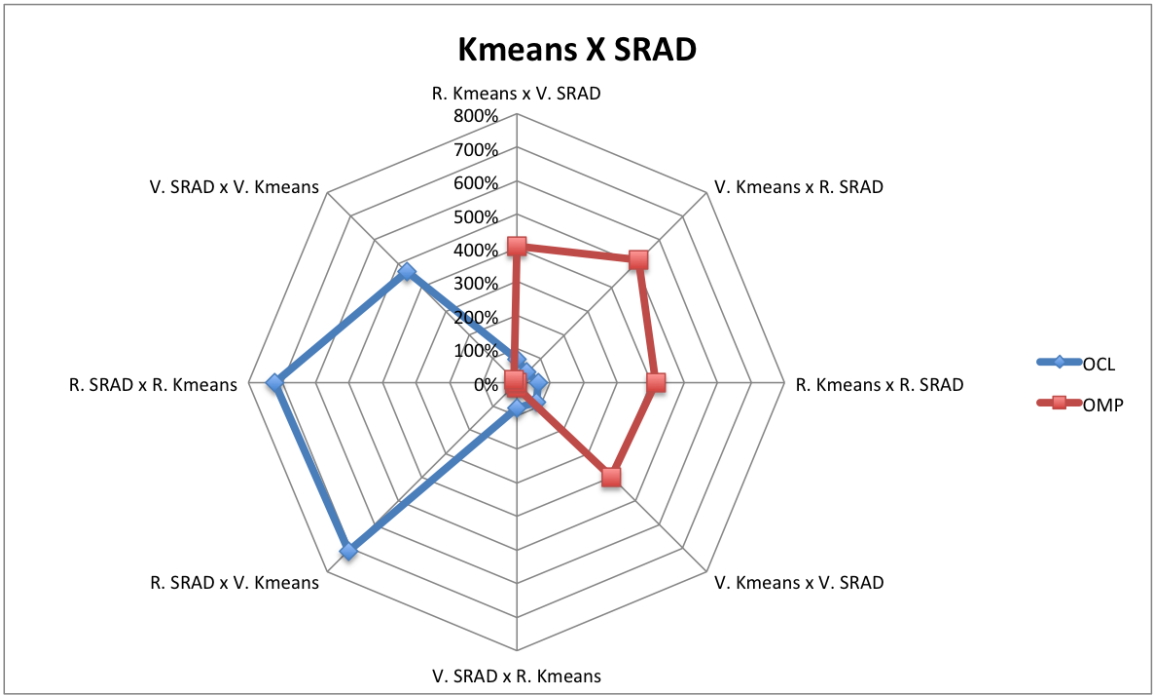

Figura 5.16: Valor de perda de desempenho do algoritmo Kmeans ou SRAD executado em um ambiente real ou virtual, causada pela concorrência do algoritmo SRAD ou Kmeans em um ambiente real ou virtual.

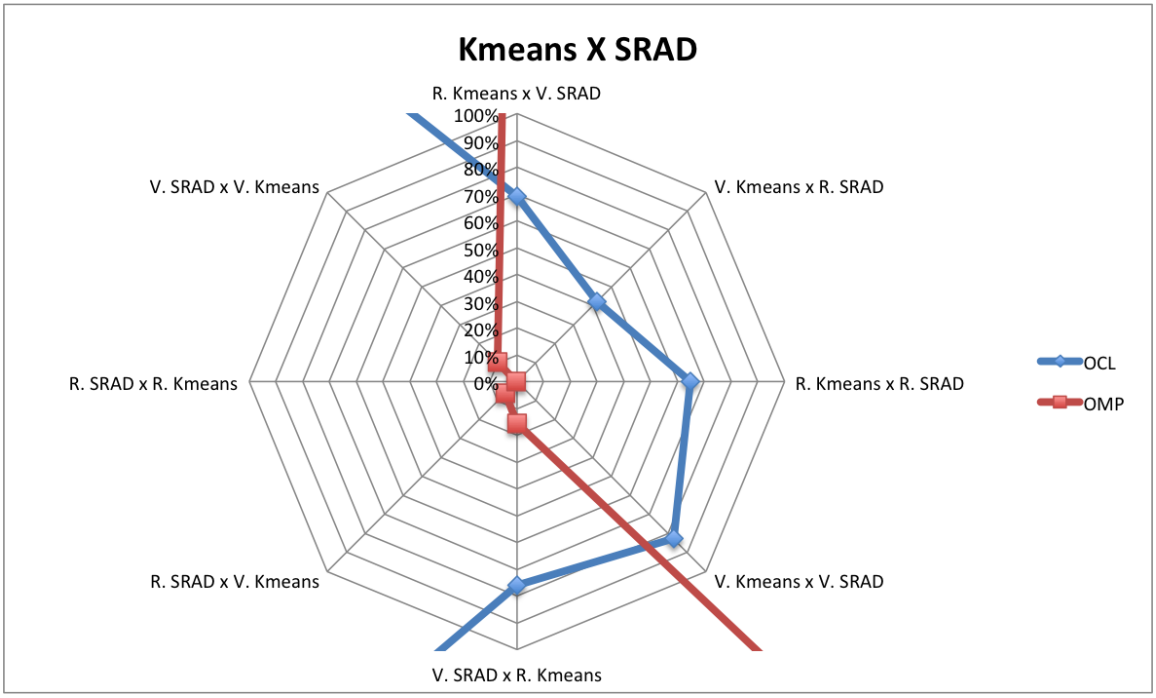

Figura 5.17: Detalhe do valor de perda de desempenho do algoritmo Kmeans ou SRAD executado em um ambiente real ou virtual, causada pela concorrência do algoritmo SRAD ou Kmeans em um ambiente real ou virtual.

Outro ponto de interesse relacionado à este teste ocorre com a comparação entre o al-

goritmo Kmeans executado em ambiente real e o algoritmo SRAD executado em ambiente virtual, repara-se na figura que quando SRAD é implementado com biblioteca OpenCL e Kmeans com OpenMP, quando executados concorrentemente, há uma maior estabilidade do ambiente, com distância de 61% entre os pontos (SRAD com perda de 76% e Kmeans com perda de 15%), entretanto quando SRAD é implementado com biblioteca OpenMP e Kmeans com biblioteca OpenCL os valores ficam mais distantes um do outro 335% (SRAD com perda de 404% e Kmeans com perda de 69%). Percebe-se ainda todas as outras combinações apresentam diferença entre os valores, o que mostra que o tipo de biblioteca também interfere na degradação do ambiente. Tomou-se este exemplo dado que é a melhor combinação para a concorrência entre estes dois tipos de algoritmos.

5.2.7 Avaliação Heterogênea do Algoritmo Kmeans X B+Tree

A Figura 5.18 mostra os resultados obtidos com algoritmo Kmeans ou B+Tree como linhas de base, executados em um ambiente real ou virtual com execução concorrente com outro algoritmo B+Tree ou Kmeans em ambiente virtual ou real. Na figura a comparação ocorre conforme a ordem de aparecimento do texto: OpenCL, seguido de OpenMP. Exemplo: R.Kmeans x V.B+Tree o real está sendo executado com biblioteca OpenCL e o virtual com OpenMP.

Na figura é possível perceber que a melhor combinação, quando há necessidade de compartilhar o mesmo recurso físico com algoritmos Kmeans e B+Tree, implementados com OpenCL e OpenMP, ocorre com perda média de 35%, quando Kmeans é executado em ambiente virtual implementado com OpenCL concorrentemente com B+Tree executado em ambiente virtual implementado com OpenMP (65% e 4% de perda respectivamente). Já a combinação que obteve a maior estabilidade no ambiente com distância entre pontos de 7% foi a combinação de B+Tree sendo executado em ambiente virtual com OpenCL concorrentemente com Kmeans sendo executado em ambiente real implementado com OpenMP (58% e 65% de perda respectivamente).

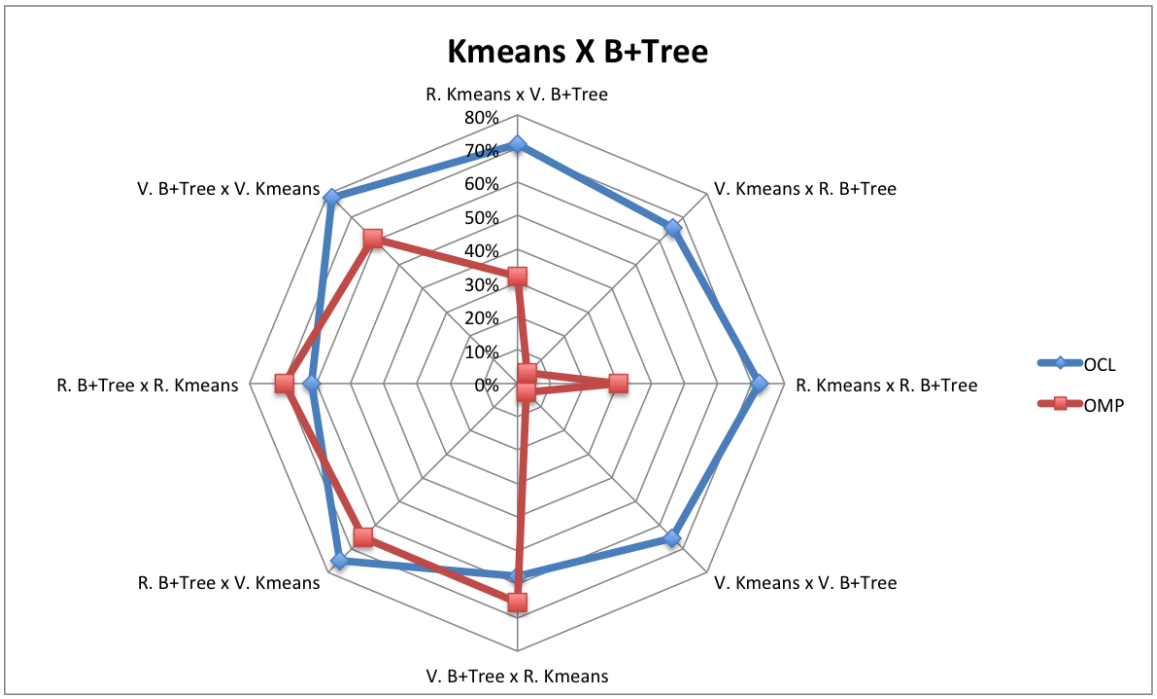

Figura 5.18: Valor de perda de desempenho do algoritmo Kmeans ou B+Tree executado em um ambiente real ou virtual, causada pela concorrência do algoritmo B+Tree ou Kmeans em um ambiente real ou virtual.

Este caso tornou-se um caso de difícil comparação, pois para a menor perda média tem-se a maior distância entre pontos (61%) e para a menor distância tem-se perda média acima de 60%. Ficando a decisão da melhor combinação baseada nos critérios de otimização do desempenho (primeiro caso) ou estabilidade do ambiente (segundo caso).

A pior combinação com perda média de 70% ocorre com B+Tree sendo executado em ambiente virtual implementado com biblioteca OpenCL concorrentemente com Kmeans em ambiente virtual implementado com biblioteca OpenMP (78% e 61% de perda respectivamente). Já o ambiente que se mostrou mais instável na combinação destes dois algoritmos teve distância entre pontos de 62% com Kmeans sendo executado em ambiente virtual implementado com biblioteca OpenCL concorrentemente com B+Tree sendo executado em ambiente real implementado com biblioteca OpenMP (66% e 4% de perda respectivamente).

Outro ponto de interesse relacionado à este teste ocorre com a comparação entre o algoritmo Kmeans executado em ambiente virtual e o algoritmo B+Tree executado em ambiente real, repara-se na figura que quando B+Tree é implementado com biblioteca OpenCL e Kmeans com OpenMP, quando executados concorrentemente, há uma maior estabilidade do ambiente, com distância de 10% entre os pontos (B+Tree com perda de 75% e Kmeans com perda de 65%), entretanto quando B+Tree é implementado com biblioteca OpenMP e Kmeans com biblioteca OpenCL os valores ficam mais distantes um do outro 61% (B+Tree com perda de 4% e Kmeans com perda de 66%). Percebe-se ainda todas as outras combinações apresentam diferença entre os valores, o que mostra que o tipo de biblioteca também interfere na degradação do ambiente. Tomou-se este exemplo dado que é a pior combinação para a concorrência entre estes dois tipos de algoritmos quando alternam-se as bibliotecas.

5.2.8 Avaliação Heterogênea do Algoritmo SRAD X LUD

As Figuras 5.19 e em detalhe 5.20 mostram os resultados obtidos com algoritmo SRAD ou LUD como linhas de base, executados em um ambiente real ou virtual com execução concorrente com outro algoritmo LUD ou SRAD em ambiente virtual ou real. Na figura a comparação ocorre conforme a ordem de aparecimento do texto: OpenCL, seguido de OpenMP. Exemplo: R.SRAD x V.LUD o real está sendo executado com biblioteca OpenCL e o virtual com OpenMP.

Nas figuras é possível perceber que a melhor combinação, quando há necessidade de compartilhar SRAD e LUD implementados com bibliotecas OpenCL e OpenMP, ocorre com perda média de 56%, quando SRAD é executado em ambiente virtual implementado com biblioteca OpenCL, concorrentemente com LUD sendo executado em ambiente real implementado com biblioteca OpenMP (81% e 30% de perda respectivamente). Já o ambiente que se mostrou mais estável, com distância entre pontos de 41% foi o algoritmo SRAD sendo executado em ambiente virtual implementado com biblioteca OpenCL con-

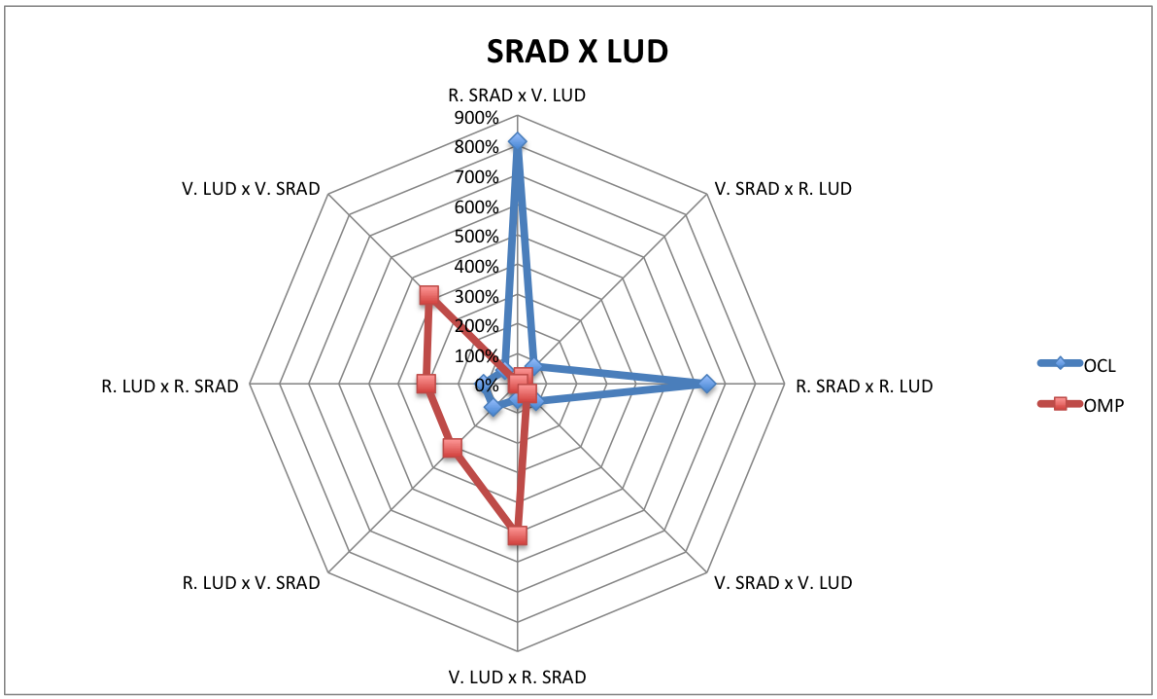

Figura 5.19: Valor de perda de desempenho do algoritmo SRAD ou LUD executado em um ambiente real ou virtual, causada pela concorrência do algoritmo LUD ou SRAD em um ambiente real ou virtual.

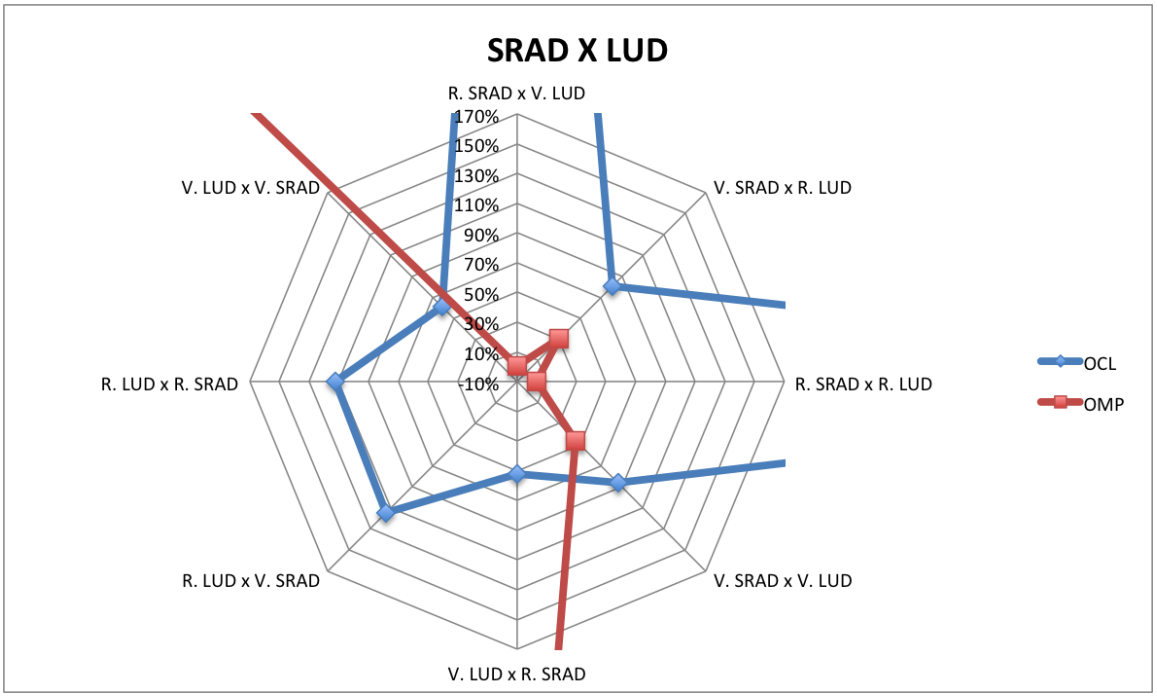

Figura 5.20: Detalhe do valor de perda de desempenho do algoritmo SRAD ou LUD executado em um ambiente real ou virtual, causada pela concorrência do algoritmo LUD ou SRAD em um ambiente real ou virtual.

correntemente com o algoritmo LUD executado em ambiente virtual implementado com biblioteca OpenMP (87% e 46% de perda respectivamente).

Este caso também tornou-se um caso de difícil comparação, pois para a menor perda média (56%) tem-se a distância entre pontos de 51% e para a menor distância entre pontos (40%) tem-se perda média acima de 66%. Ficando a decisão da melhor combinação baseada nos critérios de otimização do desempenho (primeiro caso) ou estabilidade do ambiente (segundo caso).

A pior combinação com perda média de 406% ocorre com a execução de SRAD em ambiente real implementado com OpenCL concorrentemente com LUD executando em ambiente virtual, implementado com biblioteca OpenMP (811% e 1% de perda respectivamente), sendo esta também a combinação que apresentou a maior instabilidade nos ambientes testados com distância entre pontos de 810%.

Outro ponto de interesse relacionado à este teste ocorre com a comparação entre o algoritmo SRAD executado em ambiente virtual e o algoritmo LUD executado em ambiente virtual, repara-se na figura que quando SRAD é implementado com biblioteca OpenCL e LUD com OpenMP, quando executados concorrentemente, há uma maior estabilidade do ambiente, com distância de 40% entre os pontos (SRAD com perda de 87% e LUD com perda de 46%), entretanto quando SRAD é implementado com biblioteca OpenMP e LUD com biblioteca OpenCL os valores ficam mais distantes um do outro 358% (SRAD com perda de 419% e LUD com perda de 61%). Percebe-se ainda todas as outras combinações apresentam diferença entre os valores, o que mostra que o tipo de biblioteca também interfere na degradação do ambiente. Tomou-se este exemplo dado que é a melhor combinação para a concorrência entre estes dois tipos de algoritmos.

5.2.9 Avaliação Heterogênea do Algoritmo B+Tree X LUD

As Figuras 5.21 e em detalhe 5.22 mostram os resultados obtidos com algoritmo B+Tree ou LUD como linhas de base, executados em um ambiente real ou virtual com execução

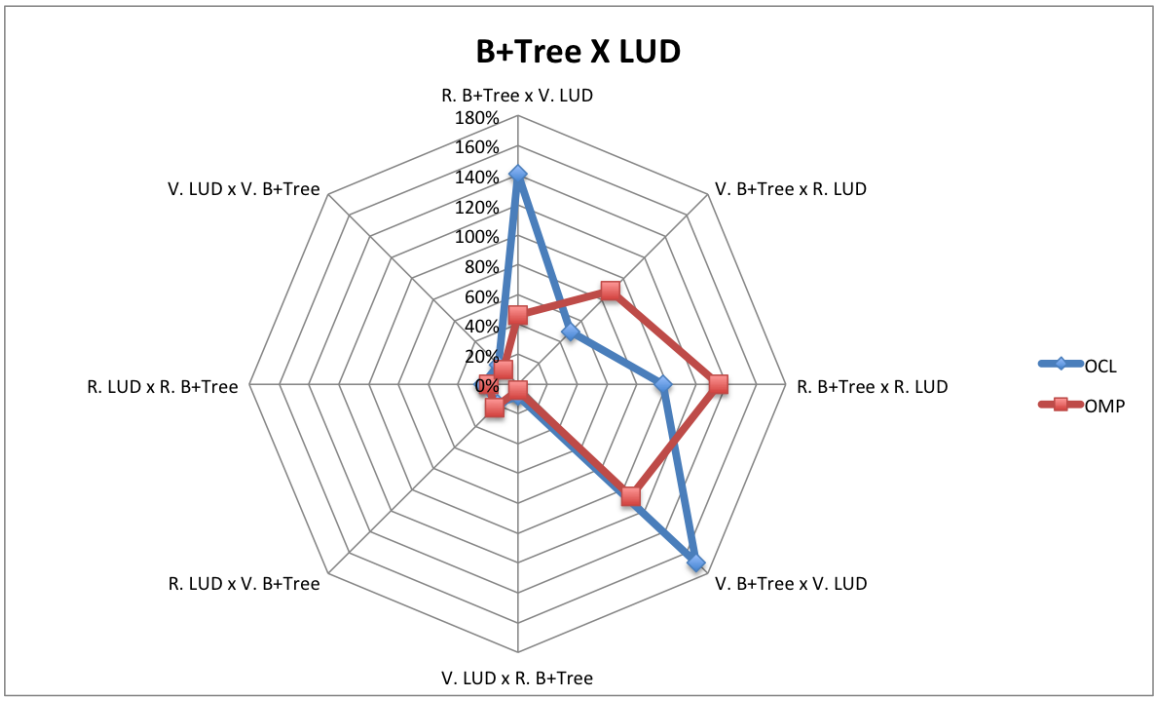

Figura 5.21: Valor de perda de desempenho do algoritmo B+Tree ou LUD executado em um ambiente real ou virtual, causada pela concorrência do algoritmo LUD ou B+Tree em um ambiente real ou virtual.

concorrente com outro algoritmo LUD ou B+Tree em ambiente virtual ou real. Na figura a comparação ocorre conforme a ordem de aparecimento do texto: OpenCL, seguido de OpenMP. Exemplo: R.B+Tree x V.LUD o real está sendo executado com biblioteca OpenCL e o virtual com OpenMP.

Nas figuras é possível perceber que a melhor combinação, quando é necessário compartilhar os recursos com os algoritmos B+Tree e LUD, implementados com bibliotecas OpenCL e OpenMP, ocorre com perda média de 6%, quando LUD é executado em ambiente virtual implementado com biblioteca OpenCL, concorrentemente com B+Tree sendo executado em ambiente real implementado com biblioteca OpenMP (8% e 4% de perda respectivamente), sendo esta a combinação que também mostrou a maior estabilidade do ambiente, com distância entre pontos de 4%.

O pior caso ocorreu com a combinação com perda média de 138% na execução de B+Tree em ambiente virtual implementado com biblioteca OpenCL, concorrentemente

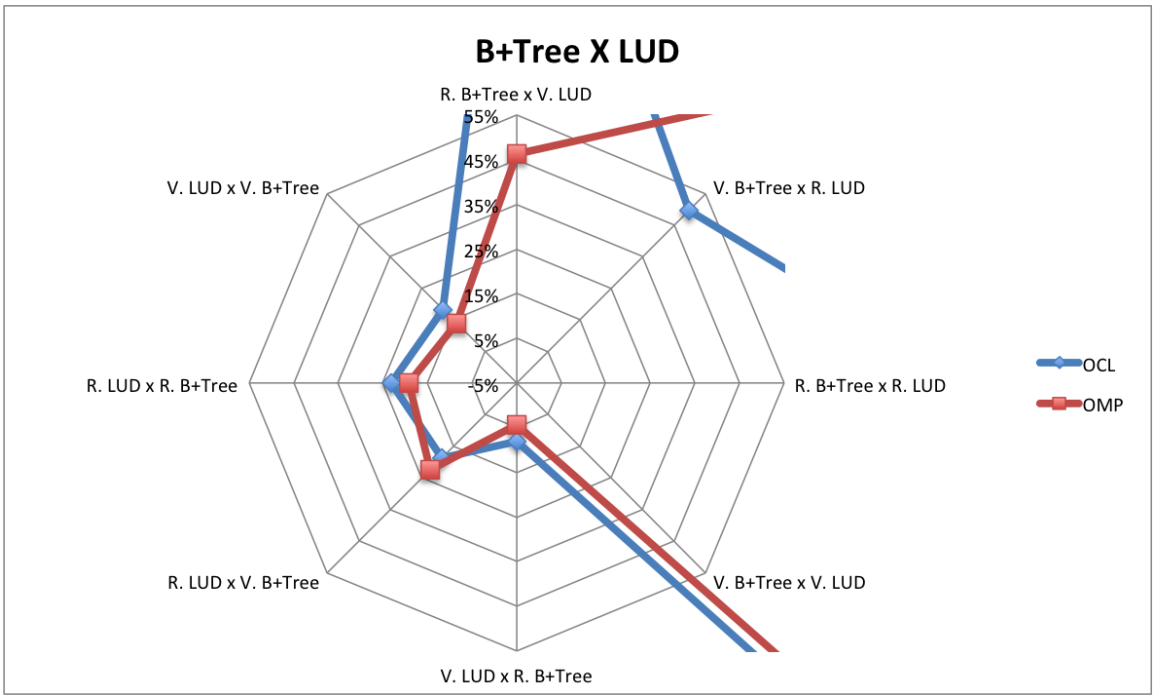

Figura 5.22: Detalhe do valor de perda de desempenho do algoritmo B+Tree ou LUD executado em um ambiente real ou virtual, causada pela concorrência do algoritmo LUD ou B+Tree em um ambiente real ou virtual.

com o algoritmo LUD sendo executado em ambiente virtual com biblioteca OpenMP (170% e 107% de perda respectivamente). Já a combinação que obteve a maior instabilidade, com distância entre pontos de 95%, foi a execução de B+Tree em ambiente real implementado com biblioteca OpenCL, concorrentemente com LUD sendo executado em ambiente virtual implementado com biblioteca OpenMP (141% e 46% de perda respectivamente).

Outro ponto de interesse relacionado à este teste ocorre com a comparação entre o algoritmo LUD executado em qualquer ambiente (real ou virtual) implementado com biblioteca OpenCL, concorrentemente com algoritmo B+Tree executando em qualquer ambiente (real ou virtual) implementado com biblioteca OpenMP. Nestes testes, além da menor perda de desempenho com relação às outras combinações, houve uma maior estabilidade do ambiente com distância de 4% entre todas as combinações, o que não ocorreu com a inversão da biblioteca, ou seja, B+Tree implementado com OpenCL e LUD implementado com OpenMP. O que permite avaliar que o tipo de biblioteca também interfere

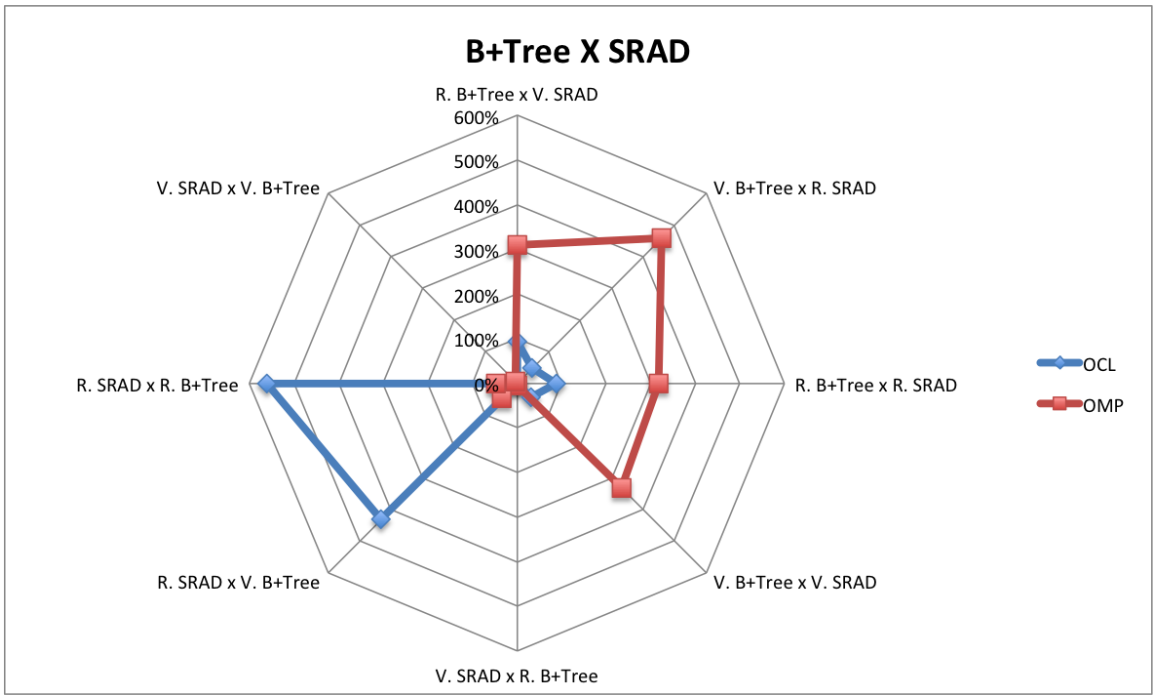

Figura 5.23: Valor de perda de desempenho do algoritmo B+Tree ou SRAD executado em um ambiente real ou virtual, causada pela concorrência do algoritmo SRAD ou B+Tree em um ambiente real ou virtual.

no desempenho das aplicações.

5.2.10 Avaliação Heterogênea do Algoritmo B+Tree X SRAD

As Figuras 5.23 e em detalhe 5.24 mostram os resultados obtidos com algoritmo B+Tree ou SRAD como linhas de base, executados em um ambiente real ou virtual com execução concorrente com outro algoritmo SRAD ou B+Tree em ambiente virtual ou real. Na figura a comparação ocorre conforme a ordem de aparecimento do texto: OpenCL, seguido de OpenMP. Exemplo: R.B+Tree x V.SRAD o real está sendo executado com biblioteca OpenCL e o virtual com OpenMP.

Nas figuras é possível perceber que as duas melhores combinações, quando é necessário compartilhar os recursos com os algoritmos B+Tree e SRAD, implementados com bibliotecas OpenCL e OpenMP, ocorrem com perda média de 2%, quando SRAD é executado em ambiente virtual implementado com biblioteca OpenCL, concorrentemente com B+Tree

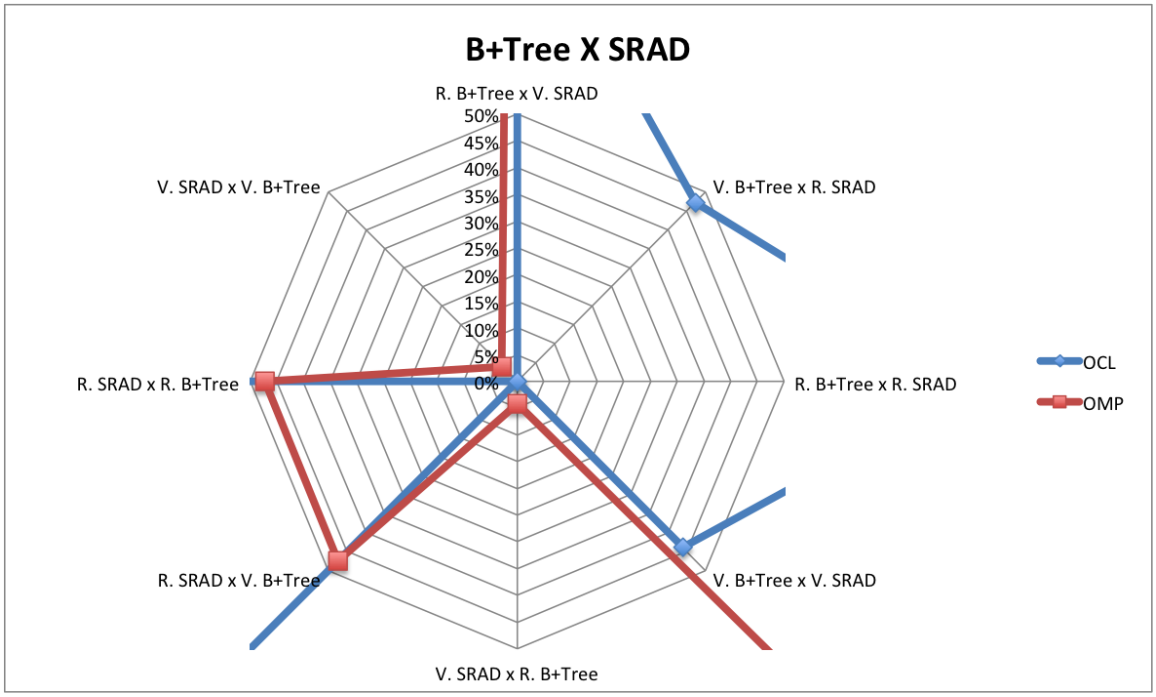

Figura 5.24: Detalhe do valor de perda de desempenho do algoritmo B+Tree ou SRAD executado em um ambiente real ou virtual, causada pela concorrência do algoritmo SRAD ou B+Tree em um ambiente real ou virtual.

sendo executado em ambiente real implementado com biblioteca OpenMP (0% e 4% de perda respectivamente) e quando SRAD é executado em ambiente virtual implementado com biblioteca OpenCL, concorrentemente com B+Tree sendo executado em ambiente virtual implementado com biblioteca OpenMP (0% e 4% de perda respectivamente), sendo estas as combinações que também mostraram as maiores estabilidades dos ambientes, com distância entre pontos de 4% para os dois casos, ou seja, qualquer destas combinações poderiam ser usadas no compartilhamento de ambiente.

O pior caso ocorreu com a combinação com perda média de 304% na execução de SRAD em ambiente real implementado com biblioteca OpenCL, concorrentemente com o algoritmo B+Tree sendo executado em ambiente real com biblioteca OpenMP (561% e 47% de perda respectivamente). Sendo esta também a combinação que obteve a maior instabilidade, com distância entre pontos de 514%.

Outro ponto de interesse relacionado à este teste ocorre com a comparação entre o

algoritmo SRAD executado em ambiente virtual implementado com biblioteca OpenCL, concorrentemente com algoritmo B+Tree executando em qualquer ambiente (real ou virtual) implementado com biblioteca OpenMP. Nestes testes, além da menor perda de desempenho com relação às outras combinações, houve uma maior estabilidade do ambiente com distância de 4% entre estas combinações, o que não ocorreu com a inversão da biblioteca, ou seja, B+Tree implementado com OpenCL e SRAD implementado com OpenMP. O que permite avaliar que o tipo de biblioteca também interfere no desempenho das aplicações.

5.3 Conclusão dos Testes de Concorrência

A Tabela 5.3 mostra as melhores combinações entre os quatro algoritmos escolhidos e as bibliotecas de implementação OpenCL e OpenMP de forma homogênea. A análise da concorrência homogênea buscou neste trabalho, apenas apresentar os resultados comparativos par a par entre os quatro tipos de algoritmos baseados nos tipos de Dwarfs executados simultaneamente, mas uma análise a posteriori mostrou que a perda de desempenho quando há adição de mais um desses quatro tipos de algoritmos no ambiente de testes, o comportamento em relação às perdas mantém um padrão constante, ou seja, a perda de desempenho é escalada nas mesmas proporções do número de adição de ambientes concorrentes. A Tabela 5.3 mostra os algoritmos base nas colunas e os concorrentes nas linhas.

Tabela 5.1: Tabela comparativa mostrando a melhor combinação em ambientes simultâneos Homogêneos.

X	**R Kmeans**	**V Kmeans**	**R SRAD**	**V SRAD**	**R B+Tree**	**V B+Tree**
Real LUD						OpenMP
Virtual LUD			OpenCL	OpenCL		
Real Kmeans			OpenCL			OpenMP
Virtual Kmeans			OpenCL		OpenMP	OpenMP
Real SRAD			OpenCL			OpenMP
Virtual SRAD			OpenCL			
Real B+Tree	OpenMP	OpenMP				
Virtual B+Tree	OpenMP	OpenMP	OpenCL		OpenMP	

Foi possível extrair dos testes que em quase todos os casos SRAD teve um desempe-

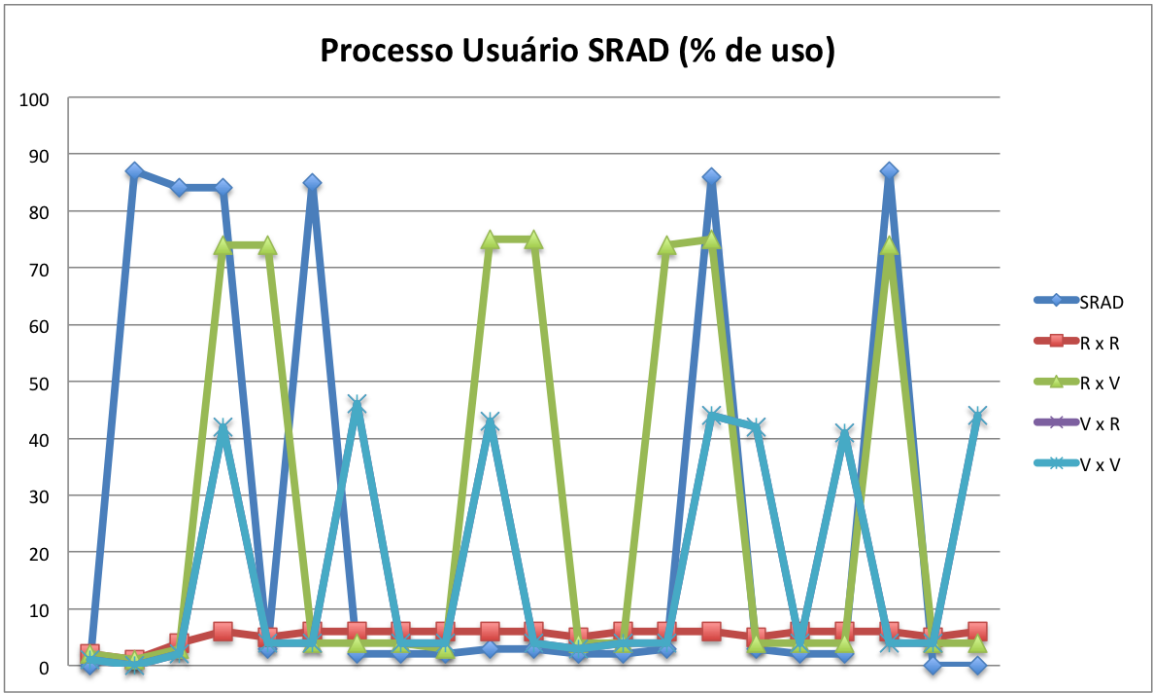

Figura 5.25: Análise do desempenho do algoritmo SRAD em relação aos processos de usuário.

nho muito pior comparado com outros algoritmos, o que sugere que este tipo de algoritmo não deve concorrer pelos mesmos recursos com outros algoritmos. O desempenho deste algoritmo foi estudado para se entender o motivo desta diferença na concorrência com outros algoritmos, principalmente com respeito à execução em ambientes reais, que teve os piores resultados em termos de perda de desempenho.

A Figura 5.25 apresenta o desempenho do algoritmo SRAD em relação aos processos de usuário. É possível verificar neste gráfico que no caso da execução entre dois processos reais, a porcentagem de uso se mostra estável e baixa.

A Figura 5.26 apresenta o desempenho do algoritmo SRAD em relação aos processos do sistema. É possível, da mesma forma, verificar neste gráfico que no caso da execução entre dois processos reais, a porcentagem de uso se mostra também estável e baixa. Os resultados da primeira e segunda amostras apresentadas no gráfico representam o momento da geração da matriz, daí os seus valores diferenciados dos demais.

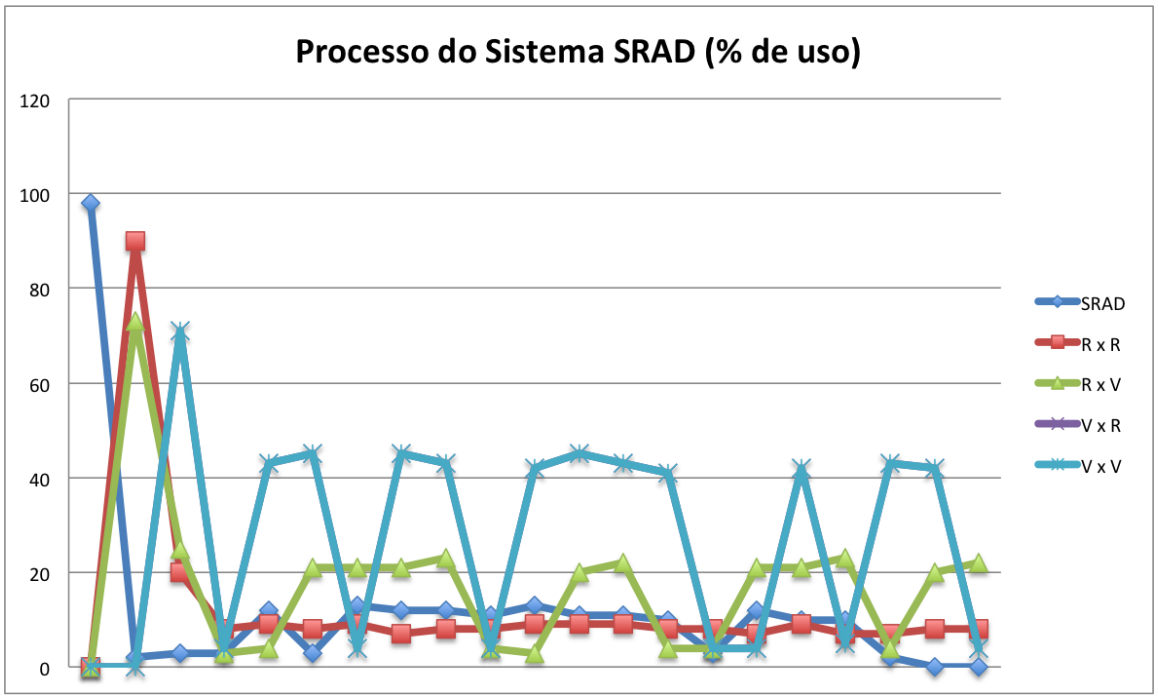

Figura 5.26: Análise do desempenho do algoritmo SRAD em relação aos processos de sistema.

A Figura 5.27 apresenta o desempenho do algoritmo SRAD em relação aos processos de leitura de dados. É possível verificar neste gráfico que no caso da execução entre dois processos reais, a quantidade de leitura de dados (em Kb) é muito maior que nas outras comparações.

A Figura 5.28 apresenta o desempenho do algoritmo SRAD em relação aos processos de escrita de dados. É possível verificar neste gráfico que no caso da execução entre dois processos reais, a porcentagem de escrita de dados (em Kb) é muito maior que nas outras comparações.

Destes dados, sabendo que houve uma maior perda de desempenho do algoritmo SRAD quando há concorrência com outros algoritmos, especialmente em relação à execução em ambiente real, verificou-se analisando as figuras, que no ambiente real o processo compete com outros processos do sistema, incluindo o processo da máquina virtual, dado a isso, há necessidade de salvar o contexto toda vez que outro processo solicitar uso da CPU. Isso não ocorre tanto com a máquina virtual, porque ela aloca memória e não compartilha

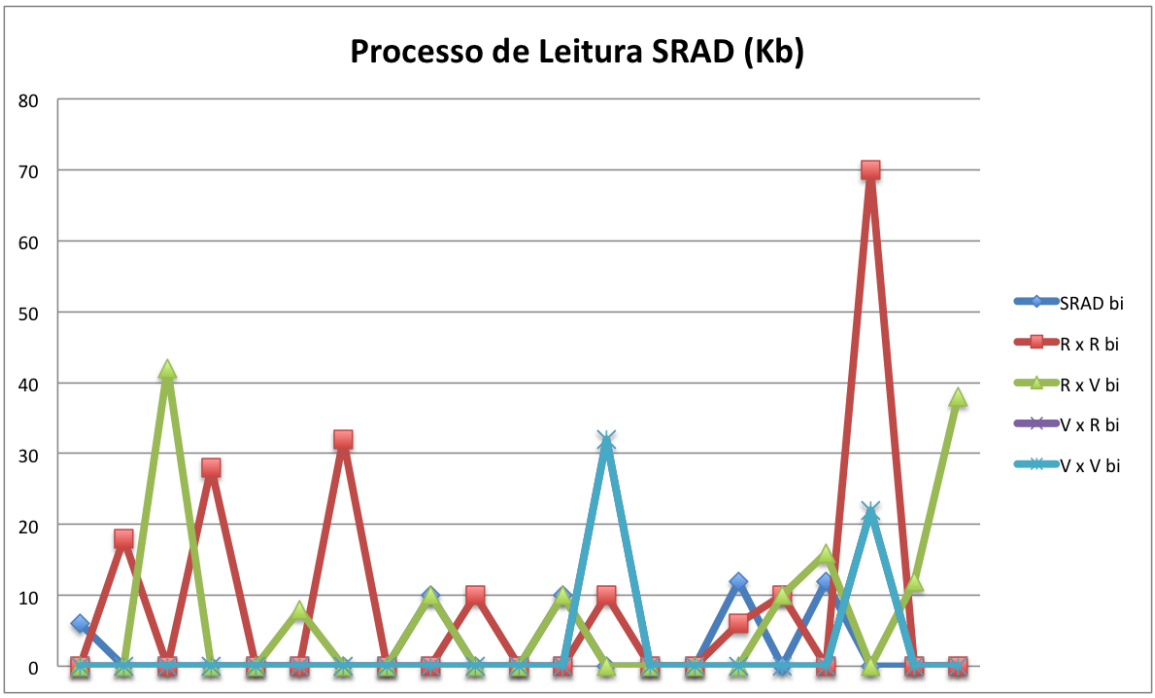

Figura 5.27: Análise do desempenho do algoritmo SRAD em relação aos processos de leitura de dados.

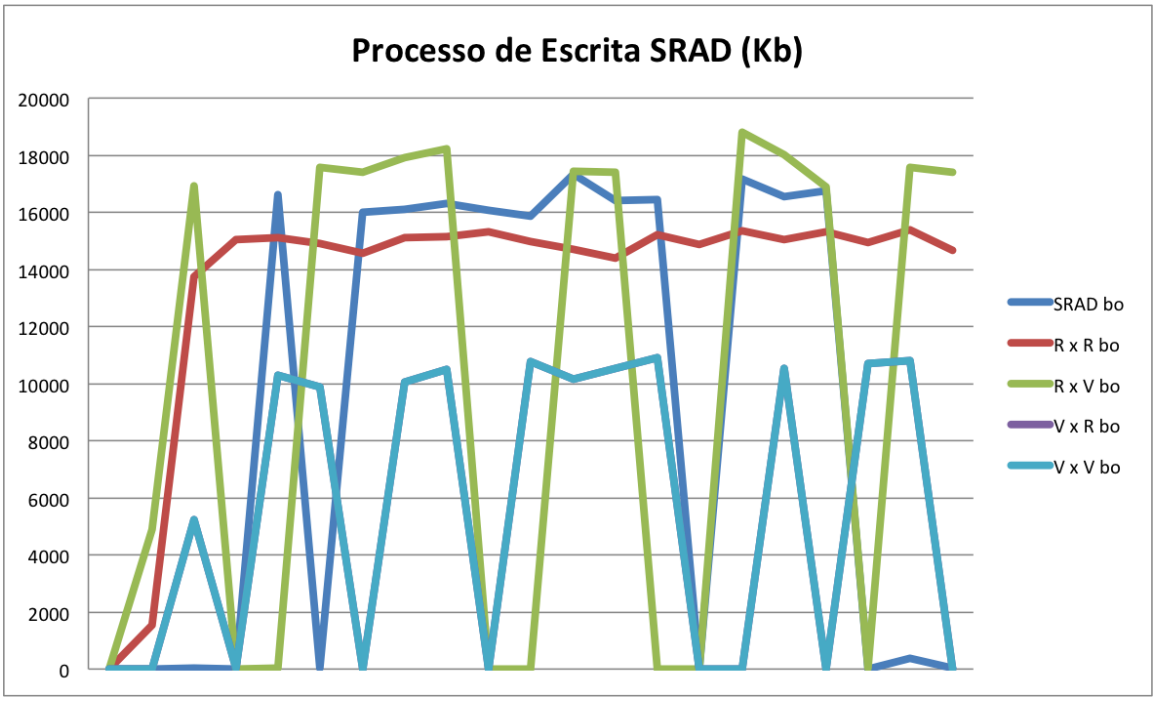

Figura 5.28: Análise do desempenho do algoritmo SRAD em relação aos processos de escrita de dados.

com outros processos da máquina real.

Analisando os processos de leitura e escrita do SRAD verifica-se também que há um

tempo de escrita alto e de leitura instável (com picos), em todos os casos, e muito maior na concorrência real x real.

A análise apresentada acima nos dá os motivos pelos quais a concorrência do SRAD com qualquer outro algoritmo acarretou as maiores perdas de desempenho.

5.4 Comparação dos Testes Heterogêneos

Esta seção apresenta os comparativos entre os testes heterogêneos, com comparações entre bibliotecas e algoritmos versus bibliotecas.

5.4.1 Avaliação das Bibliotecas Utilizadas

As Figuras 5.29, 5.30, 5.31 e 5.32 apresentam a comparação entre os algoritmos executados com cada combinação entre as bibliotecas (OpenCL e OpenMP) e a perda média. Em cada figura o nome da primeira coluna é a implementação com OpenCL e a segunda coluna é a implementação com OpenMP, por exemplo, na coluna R. Kmeans x V. LUD, R. Kmeans foi implementado com OpenCL e V. LUD com biblioteca OpenMP.

5.4.2 Impacto Causado e Sofrido Pelo Algoritmo Kmeans em Ambiente Real

A Figura 5.33 apresenta o impacto causado e sofrido pelo algoritmo Kmeans em ambiente real sobre outros algoritmos em ambiente real ou virtual. Devido à diferença entre tempos de execução de cada algoritmo, foi aplicada a normalização de valores em uma escala entre 0 e 9 como apresentado na Seção 4.6 e pelas Fórmulas 4.1 e 4.2. Com estes valores normalizados é possível comparar algoritmos com tempos de grandezas diferentes.

Na figura é possível verificar que o maior impacto causado pelo algoritmo Kmeans

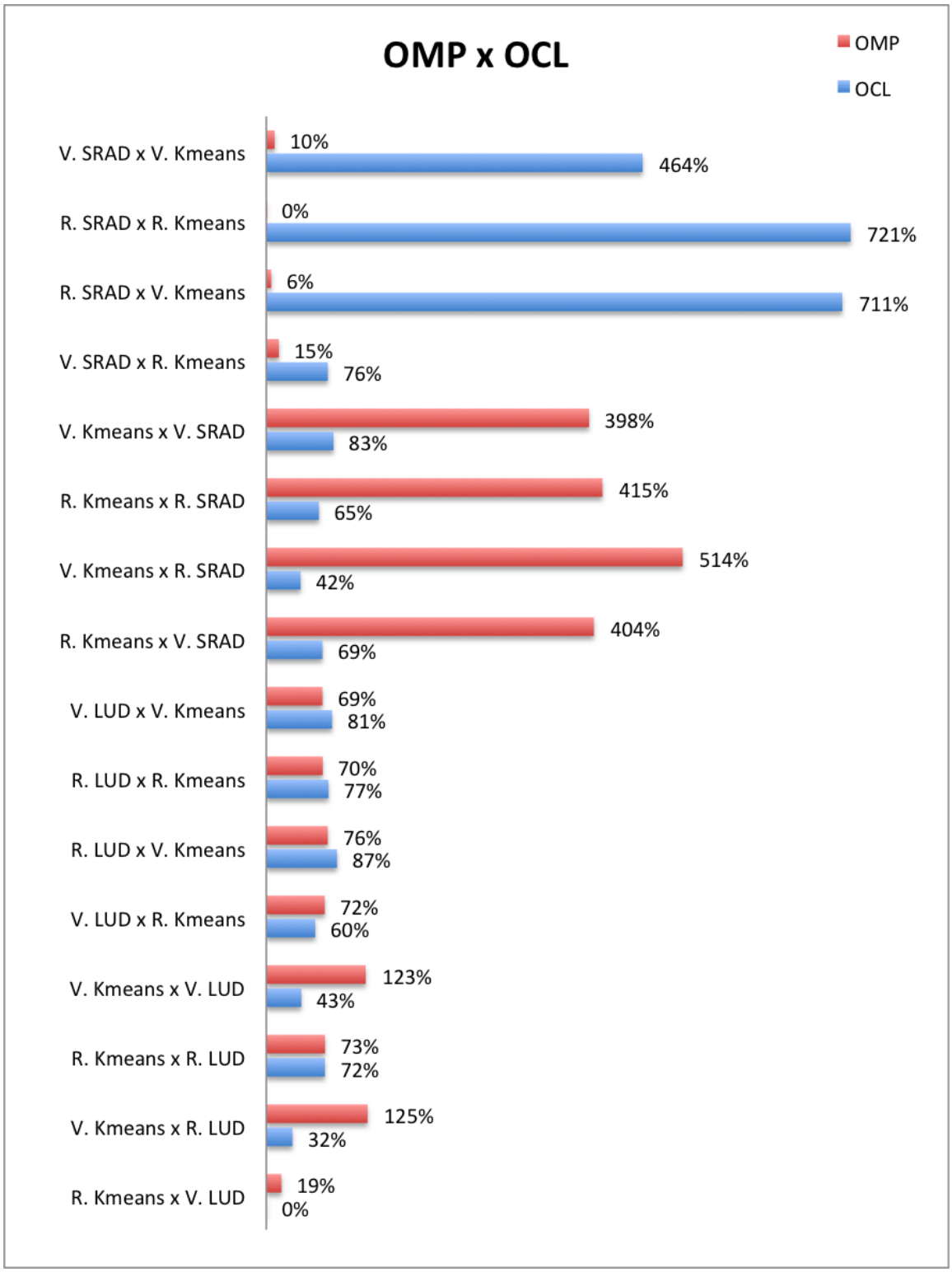

Figura 5.29: Comparativo de Desempenho Entre OpenMP e OpenCL (Gráfico 1 de 4).

sendo executado em ambiente real acontece quando este algoritmo implementado com biblioteca OpenMP concorre com B+Tree executando em ambiente real implementado com biblioteca OpenCL, sendo que Kmeans teve perda de desempenho normalizado 3,3

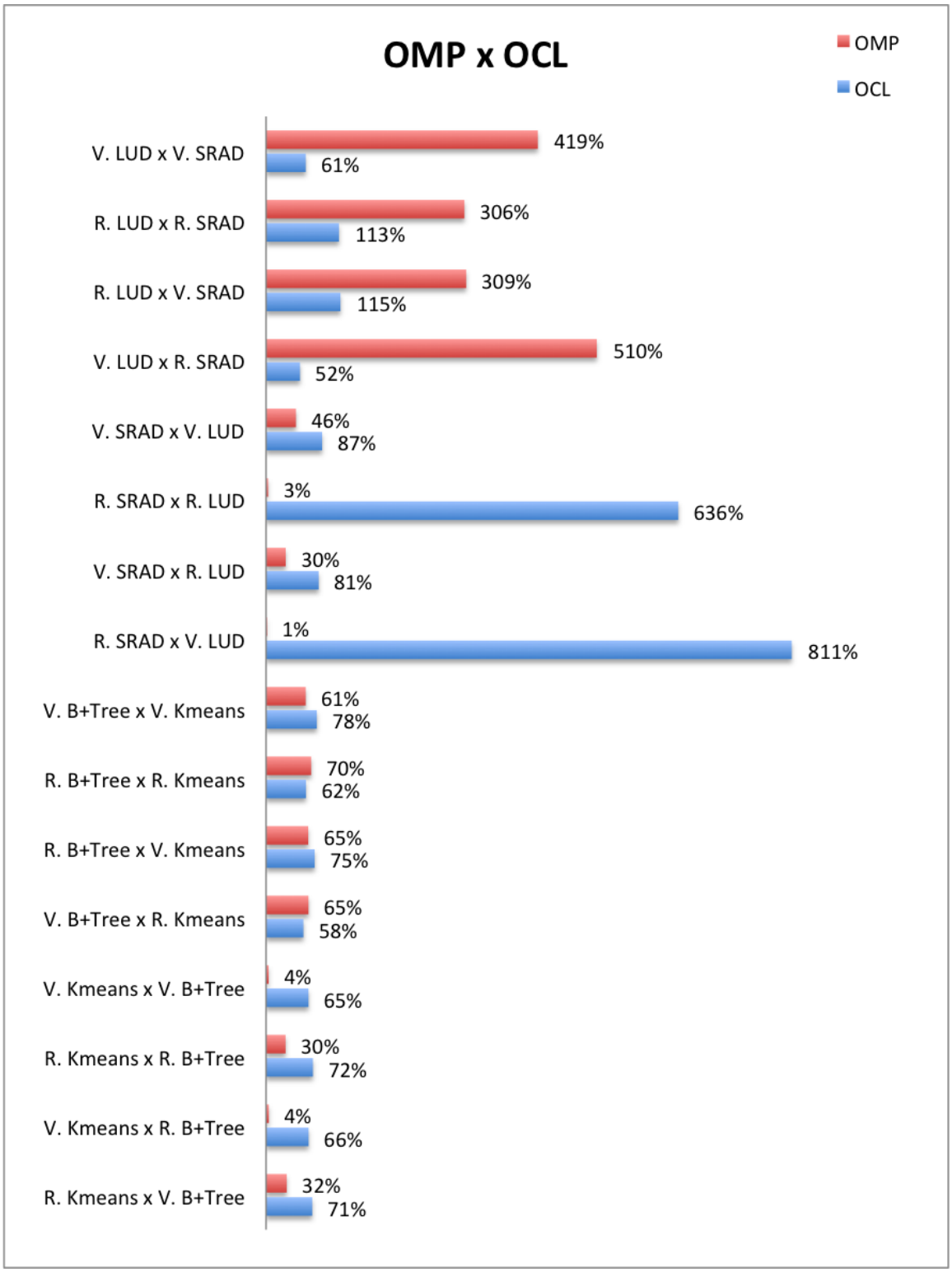

Figura 5.30: Comparativo de Desempenho Entre OpenMP e OpenCL (Gráfico 2 de 4).

e B+Tree teve perda de desempenho normalizado 7,65.

Verifica-se ainda, com respeito ao impacto causado pelo algoritmo Kmeans em ambi-

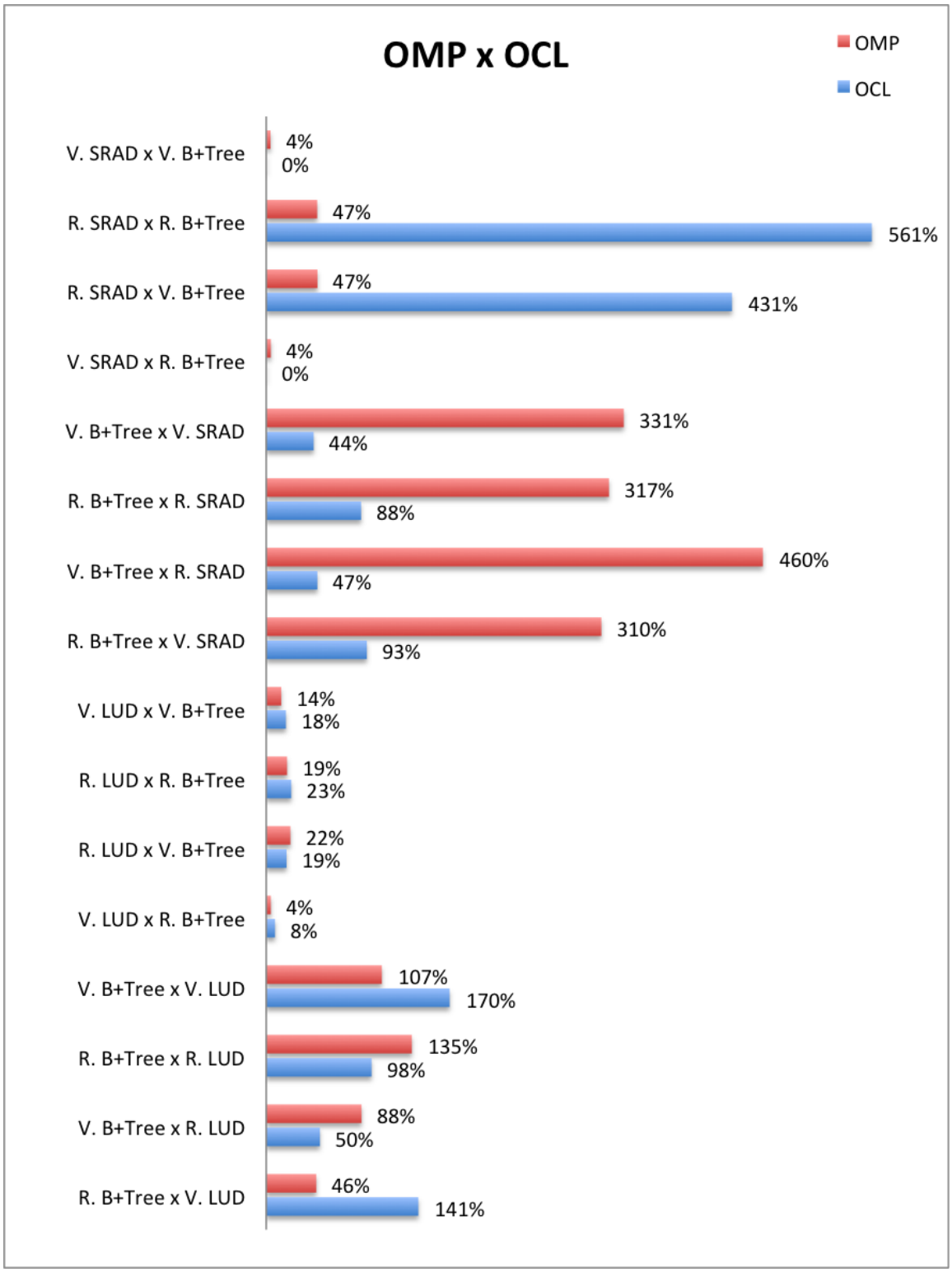

Figura 5.31: Comparativo de Desempenho Entre OpenMP e OpenCL (Gráfico 3 de 4).

ente real que, quando implementado com biblioteca OpenCL, acarreta perda de desempenho normalizado igual a 7,65 no algoritmo B+Tree sendo executado em ambiente virtual, implementado com biblioteca OpenMP, sendo que o algoritmo Kmeans sofre perda de

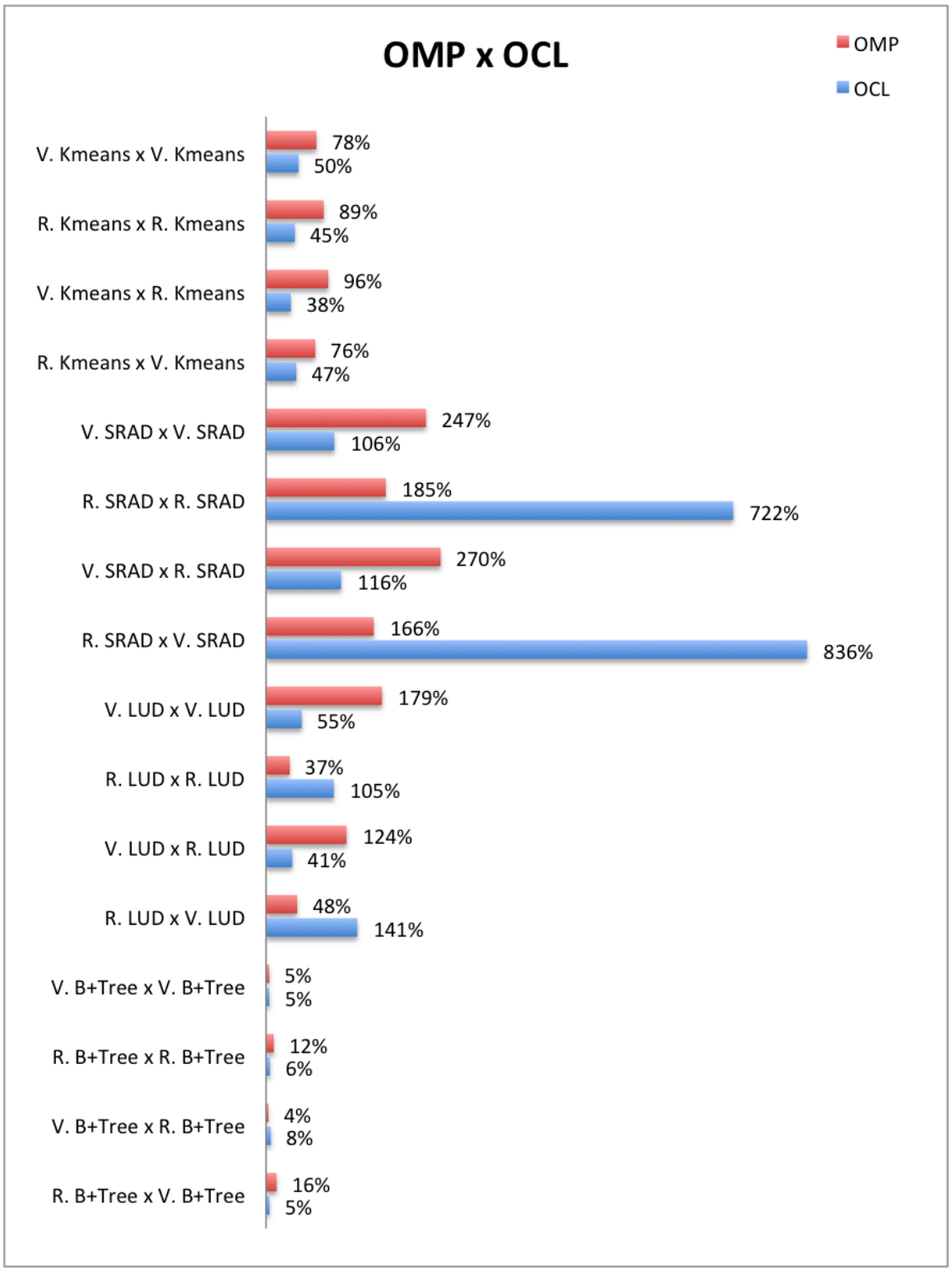

Figura 5.32: Comparativo de Desempenho Entre OpenMP e OpenCL (Gráfico 4 de 4).

desempenho normalizada igual a 5,85.

Já com respeito ao maior impacto sofrido, ou seja, impacto causado por outro algo-

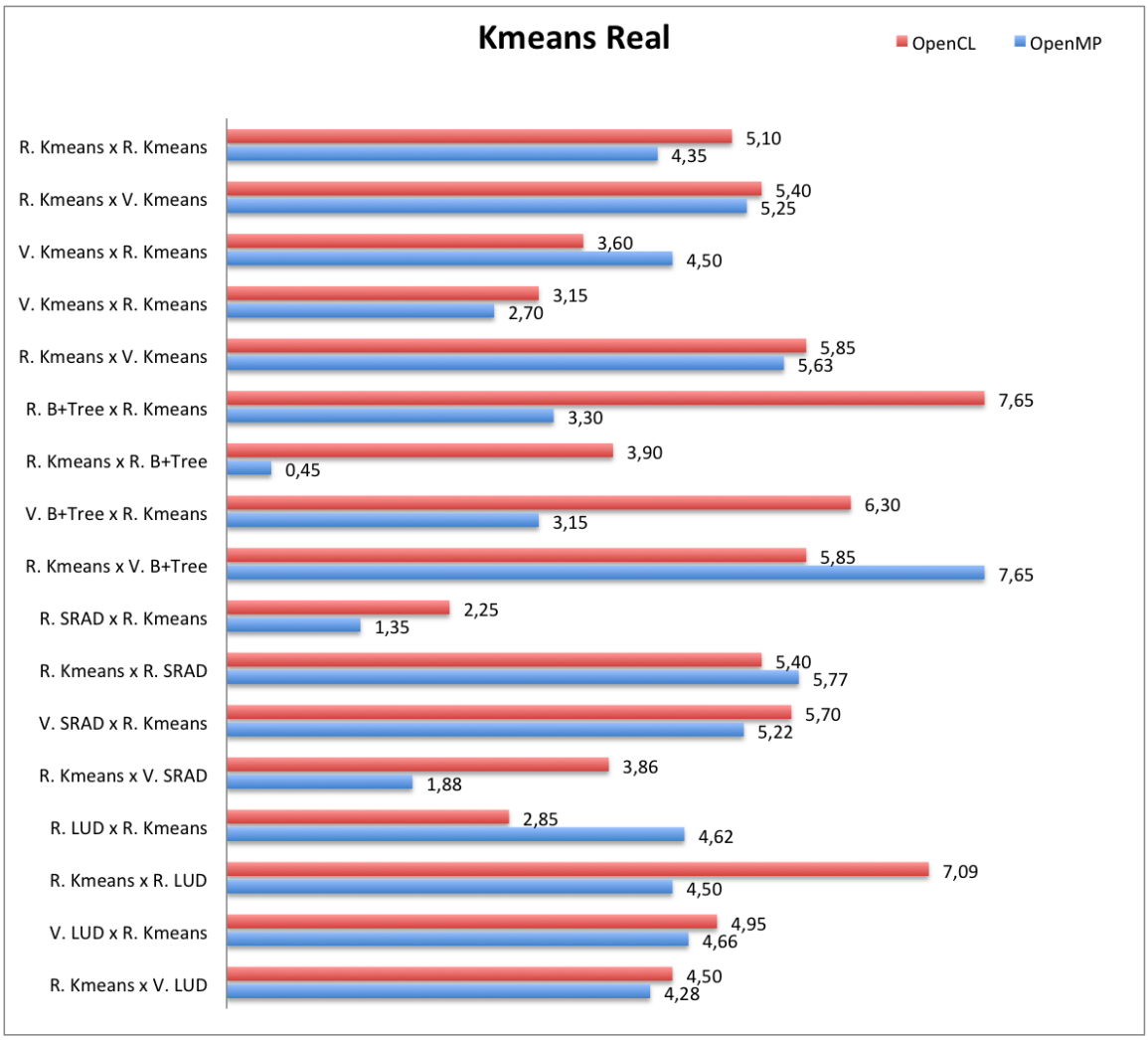

Figura 5.33: Impacto sofrido e causado pelo algoritmo Kmeans real.

ritmo, ocorre quando Kmeans sendo executado em ambiente real implementado com biblioteca OpenCL concorre pelos mesmos recursos com LUD sendo executado em ambiente real implementado com biblioteca OpenMP, sendo que Kmeans tem perda de desempenho normalizada igual a 7,09 e LUD 4,50.

5.4.3 Impacto Causado e Sofrido Pelo Algoritmo Kmeans em Ambiente Virtual

A Figura 5.34 apresenta o impacto causado e sofrido pelo algoritmo Kmeans em ambiente virtual sobre outros algoritmos em ambiente real ou virtual. Devido à diferença entre

tempos de execução de cada algoritmo, foi aplicada a normalização de valores em uma escala entre 0 e 9 como apresentado na Seção 4.6 e pelas Fórmulas 4.1 e 4.2. Com estes valores normalizados é possível comparar algoritmos com tempos de grandezas diferentes.

Na figura é possível verificar que o maior impacto causado pelo algoritmo Kmeans sendo executado em ambiente virtual acontece quando este algoritmo implementado com biblioteca OpenCL concorre com SRAD executando em ambiente virtual implementado com biblioteca OpenMP, sendo que Kmeans teve perda de desempenho normalizado 3,6 e SRAD teve perda de desempenho normalizado 8,1.

Já com respeito ao maior impacto sofrido, ou seja, impacto causado por outro algoritmo, ocorre quando Kmeans sendo executado em ambiente virtual implementado com biblioteca OpenMP concorre pelos mesmos recursos com SRAD sendo executado em ambiente real implementado com biblioteca OpenCL, sendo que Kmeans tem perda de desempenho normalizada igual a 7,65 e SRAD 5,4.

5.4.4 Impacto Causado e Sofrido Pelo Algoritmo LUD em Ambiente Real

A Figura 5.35 apresenta o impacto causado e sofrido pelo algoritmo LUD em ambiente real sobre outros algoritmos em ambiente real ou virtual. Devido à diferença entre tempos de execução de cada algoritmo, foi aplicada a normalização de valores em uma escala entre 0 e 9 como apresentado na Seção 4.6 e pelas Fórmulas 4.1 e 4.2. Com estes valores normalizados é possível comparar algoritmos com tempos de grandezas diferentes.

Na figura é possível verificar que o maior impacto causado pelo algoritmo LUD sendo executado em ambiente real acontece quando este algoritmo implementado com biblioteca OpenMP concorre com o algoritmo Kmeans executando em ambiente real implementado com biblioteca OpenCL, sendo que LUD teve perda de desempenho normalizada 4,5 e

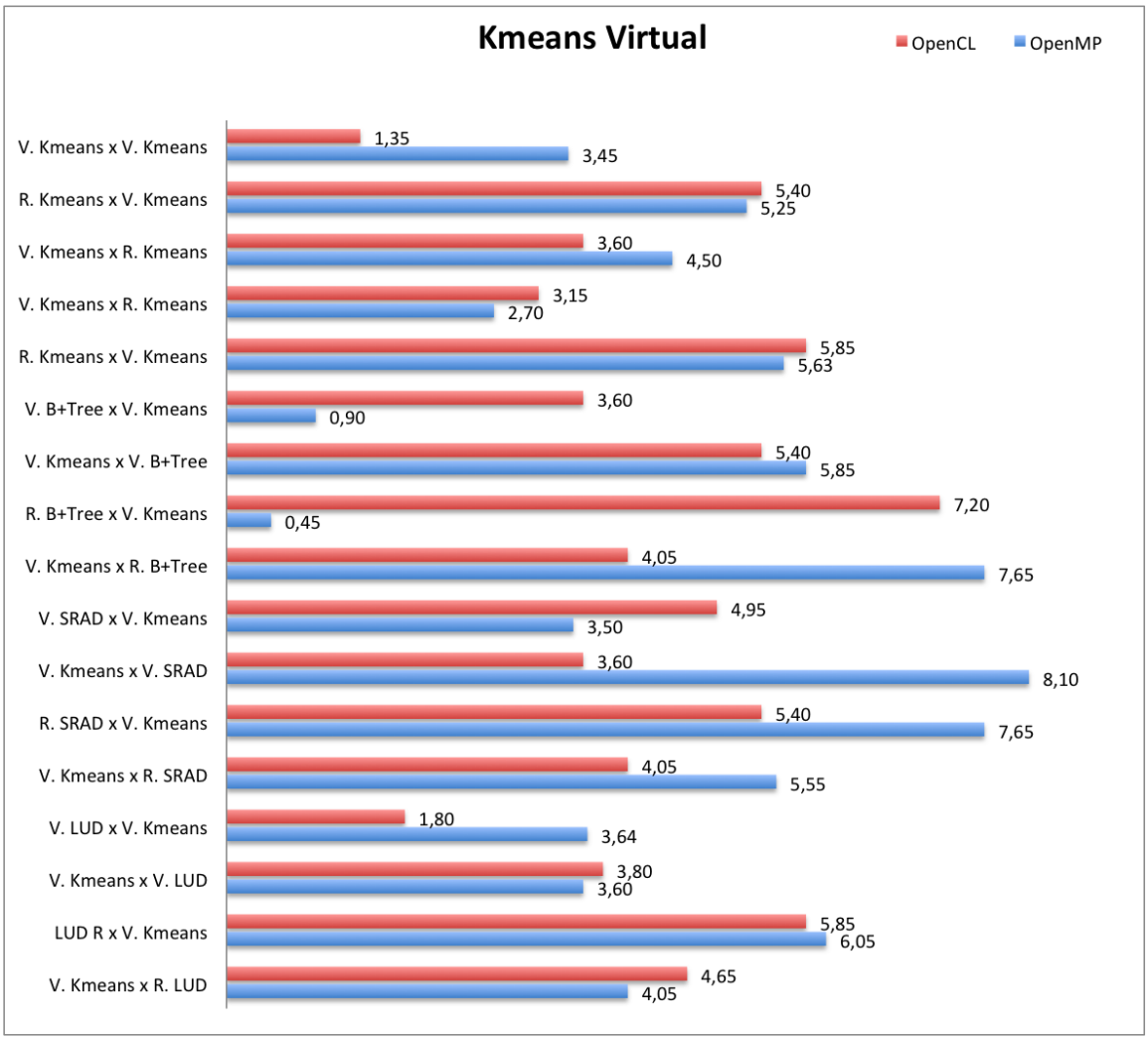

Figura 5.34: Impacto sofrido e causado pelo algoritmo Kmeans virtual.

Kmeans 7,09.

Já com respeito ao maior impacto sofrido, ou seja, impacto causado por outro algoritmo, ocorre quando LUD sendo executado em ambiente real implementado com biblioteca OpenCL concorre pelos mesmos recursos com LUD sendo executado em ambiente real implementado com biblioteca OpenMP, sendo que LUD implementado com OpenCL tem perda de desempenho normalizada igual a 6,75 e LUD implementado com OpenMP teve perda de desempenho normalizada igual a 4,07.

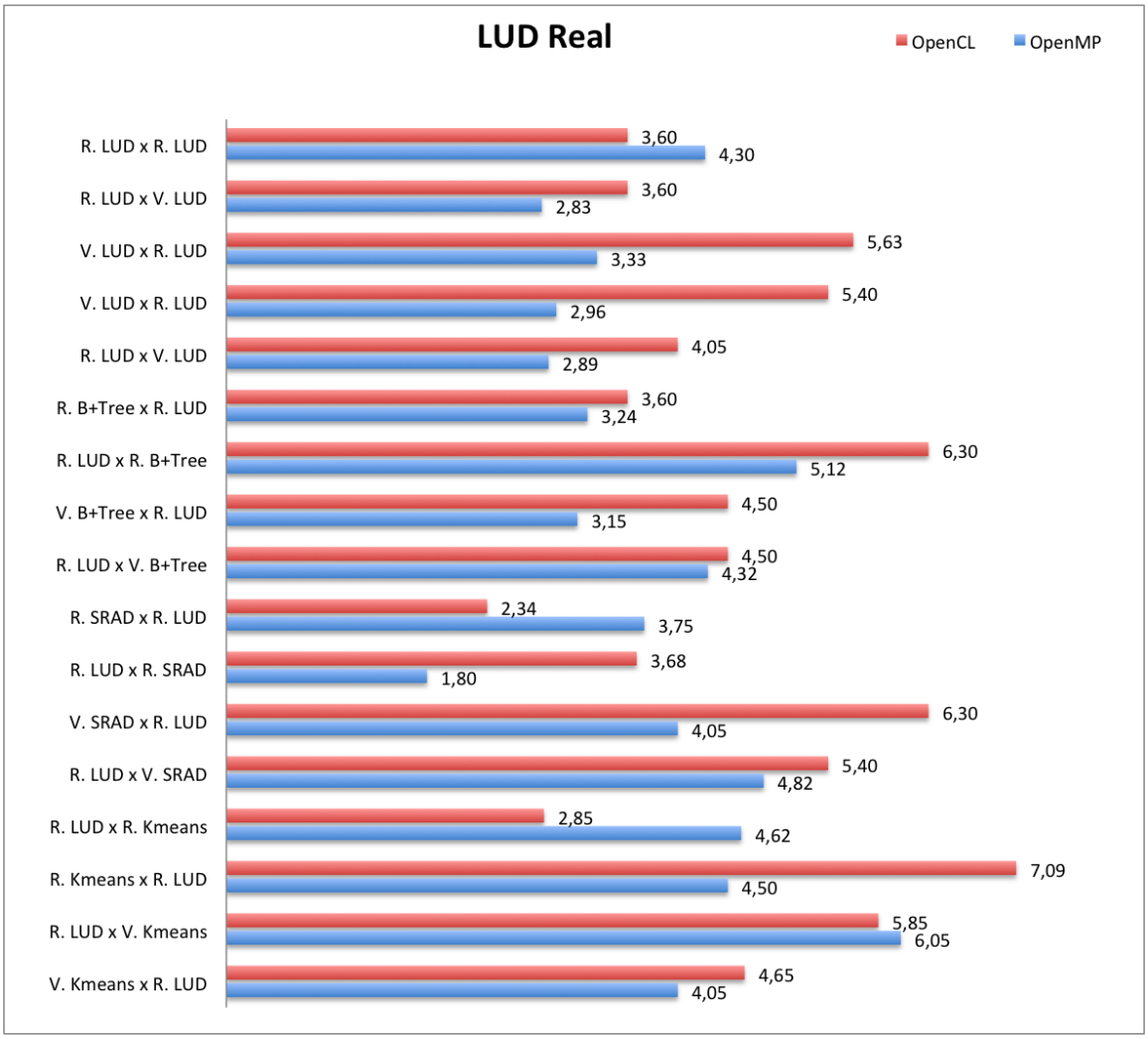

Figura 5.35: Impacto sofrido e causado pelo algoritmo LUD real.

5.4.5 Impacto Causado e Sofrido Pelo Algoritmo LUD em Ambiente Virtual

A Figura 5.36 apresenta o impacto causado e sofrido pelo algoritmo LUD em ambiente virtual sobre outros algoritmos em ambiente real ou virtual. Devido à diferença entre tempos de execução de cada algoritmo, foi aplicada a normalização de valores em uma escala entre 0 e 9 como apresentado na Seção 4.6 e pelas Fórmulas 4.1 e 4.2. Com estes valores normalizados é possível comparar algoritmos com tempos de grandezas diferentes.

Na figura é possível verificar que o maior impacto causado pelo algoritmo LUD sendo executado em ambiente virtual acontece quando este algoritmo implementado com bibli-

oteca OpenCL concorre com o algoritmo B+Tree executando em ambiente virtual implementado com biblioteca OpenMP, sendo que LUD teve perda de desempenho normalizada 4,32 e B+Tree 6,30.

Já com respeito ao maior impacto sofrido, ou seja, impacto causado por outro algoritmo, ocorre quando LUD sendo executado em ambiente virtual implementado com biblioteca OpenCL concorre pelos mesmos recursos com LUD sendo executado em ambiente real implementado com biblioteca OpenMP, sendo que LUD implementado com OpenCL executando em ambiente virtual tem perda de desempenho normalizada igual a 5,63 e LUD implementado com OpenMP executando em ambiente real teve perda de desempenho normalizada igual a 3,33.

5.4.6 Impacto Causado e Sofrido Pelo Algoritmo SRAD em Ambiente Real

A Figura 5.37 apresenta o impacto causado e sofrido pelo algoritmo SRAD em ambiente real sobre outros algoritmos em ambiente real ou virtual. Devido à diferença entre tempos de execução de cada algoritmo, foi aplicada a normalização de valores em uma escala entre 0 e 10 como apresentado na Seção 4.6 e pelas Fórmulas 4.1 e 4.2. Com estes valores normalizados é possível comparar algoritmos com tempos de grandezas diferentes.

Na figura é possível verificar que o maior impacto causado pelo algoritmo SRAD sendo executado em ambiente real acontece quando este algoritmo implementado com biblioteca OpenCL concorre com o algoritmo Kmeans executando em ambiente virtual implementado com biblioteca OpenMP, sendo que SRAD teve perda de desempenho normalizada 5,40 e Kmeans 7,65.

Já com respeito ao maior impacto sofrido, ou seja, impacto causado por outro algoritmo, ocorre quando SRAD sendo executado em ambiente real implementado com

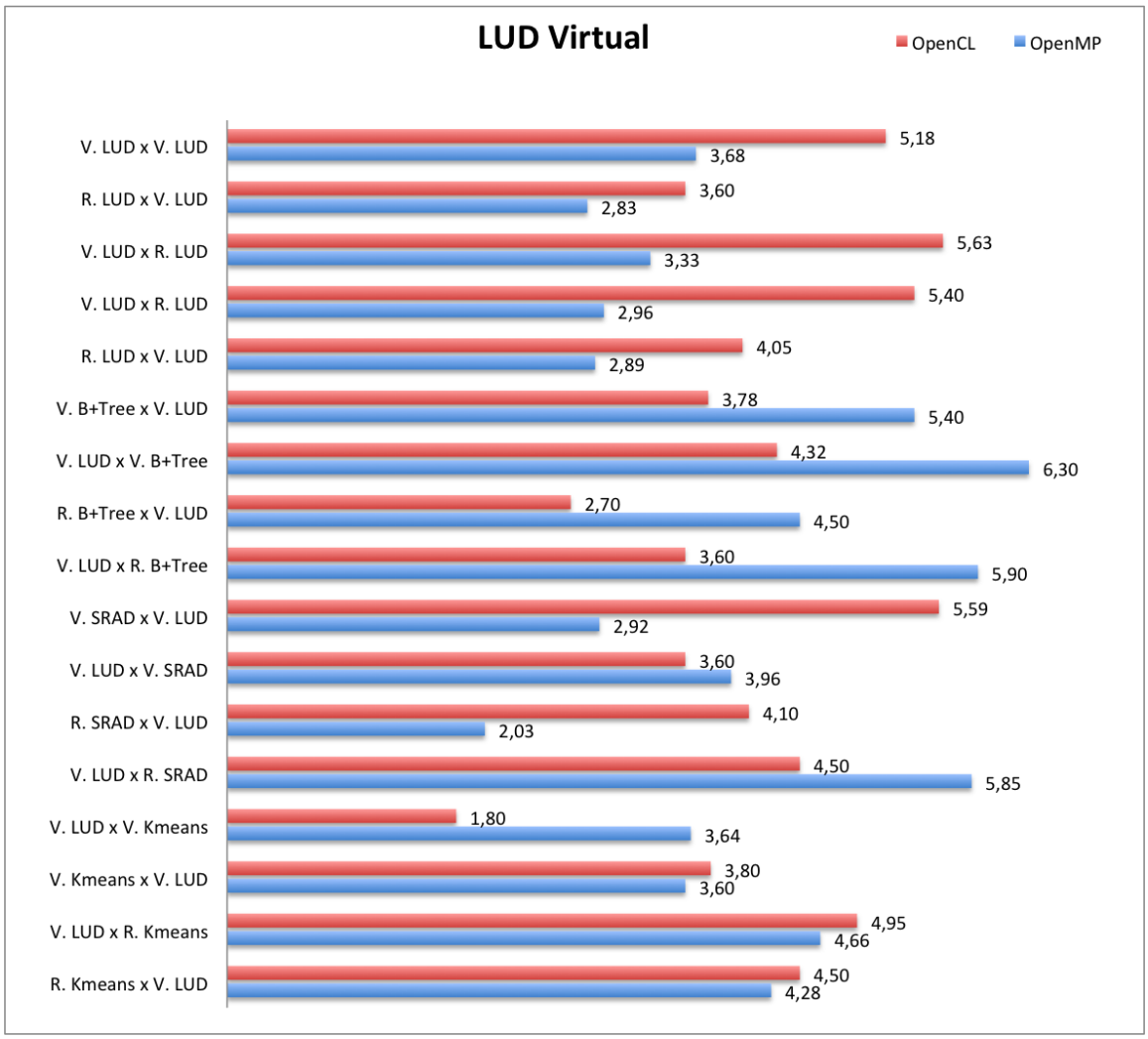

Figura 5.36: Impacto sofrido e causado pelo algoritmo LUD virtual.

biblioteca OpenMP concorre pelos mesmos recursos com B+Tree sendo executado em ambiente real implementado com biblioteca OpenCL, sendo que SRAD teve perda de desempenho normalizada igual a 8,55 e B+Tree teve perda de desempenho normalizada igual a 4,50.

5.4.7 Impacto Causado e Sofrido Pelo Algoritmo SRAD em Ambiente Virtual

A Figura 5.38 apresenta o impacto causado e sofrido pelo algoritmo SRAD em ambiente virtual sobre outros algoritmos em ambiente real ou virtual. Devido à diferença entre

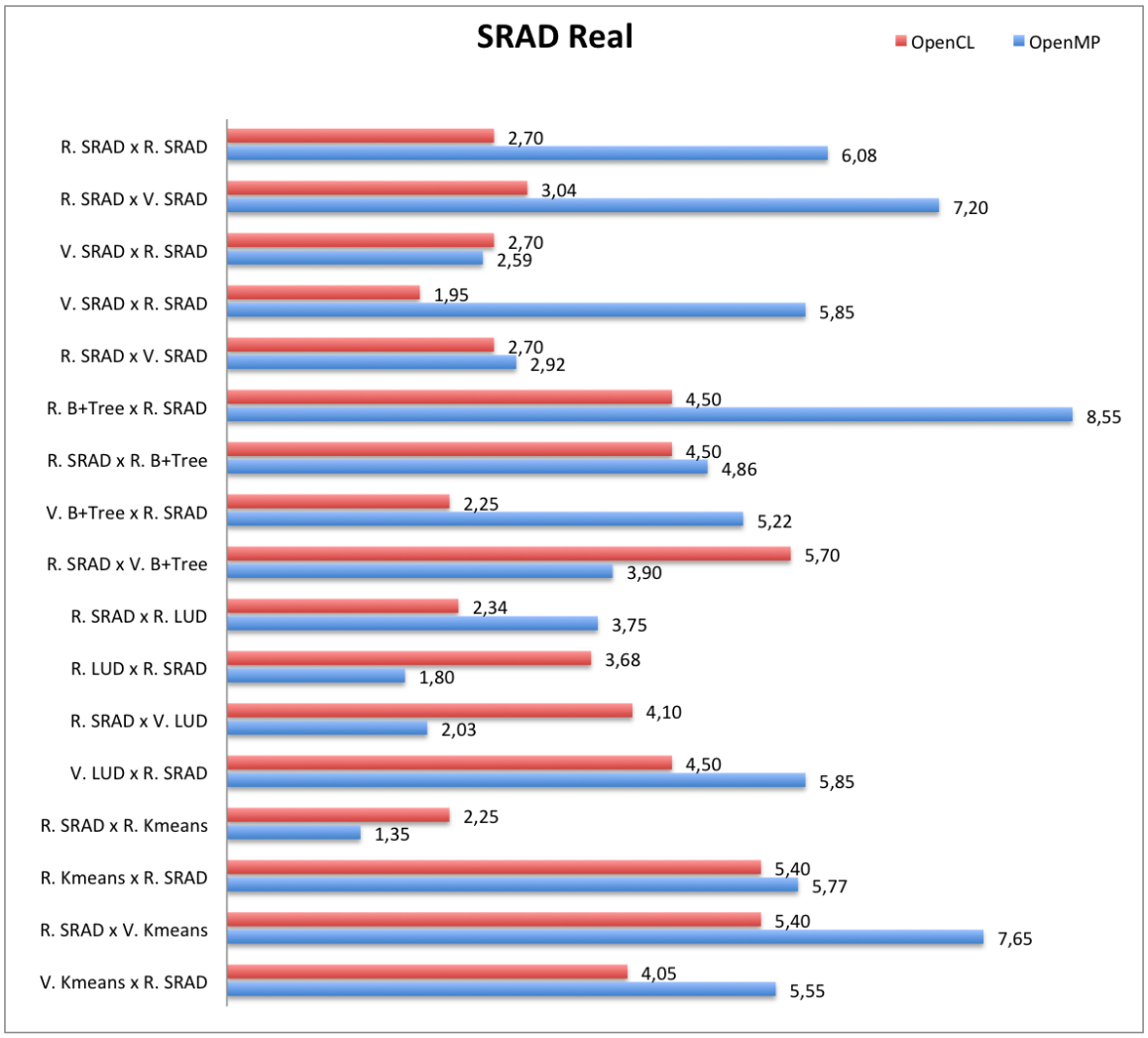

Figura 5.37: Impacto sofrido e causado pelo algoritmo SRAD real.

tempos de execução de cada algoritmo, foi aplicada a normalização de valores em uma escala entre 0 e 10 como apresentado na Seção 4.6 e pelas Fórmulas 4.1 e 4.2. Com estes valores normalizados é possível comparar algoritmos com tempos de grandezas diferentes.

Na figura é possível verificar que o maior impacto causado pelo algoritmo SRAD sendo executado em ambiente virtual acontece quando este algoritmo implementado com biblioteca OpenCL concorre com o mesmo algoritmo SRAD executando em ambiente real implementado com biblioteca OpenMP, sendo que SRAD implementado com OpenCL executando em ambiente virtual teve perda de desempenho normalizada 1,95 e SRAD implementado com OpenMP executando em ambiente real teve perda de desempenho normalizada 5,85.

Já com respeito ao maior impacto sofrido, ou seja, impacto causado por outro algoritmo, ocorre quando SRAD sendo executado em ambiente virtual implementado com biblioteca OpenMP concorre pelos mesmos recursos com Kmeans sendo executado em ambiente virtual implementado com biblioteca OpenCL, sendo que SRAD teve perda de desempenho normalizada igual a 8,10 e Kmeans teve perda de desempenho normalizada igual a 3,60.

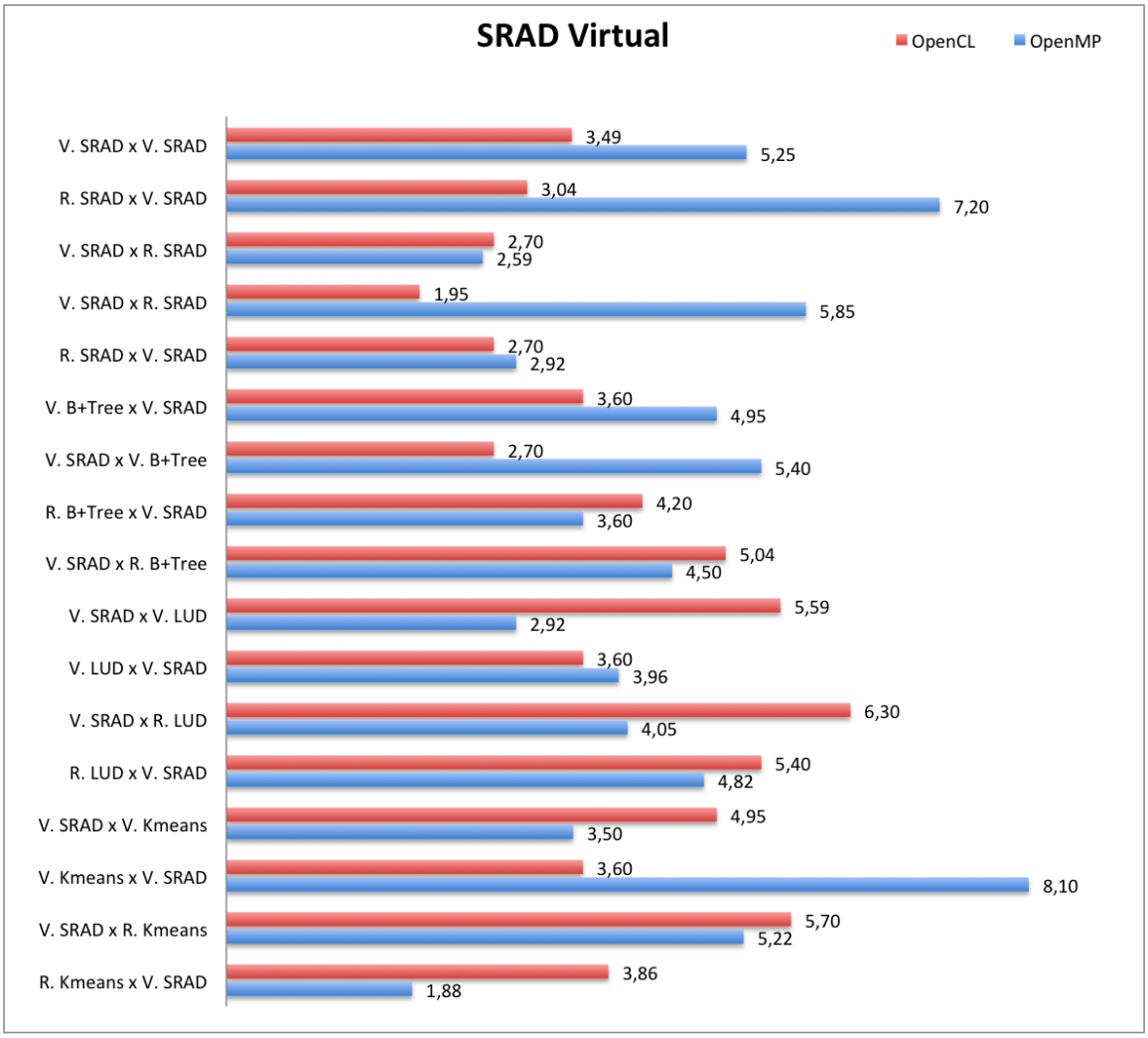

Figura 5.38: Impacto sofrido e causado pelo algoritmo SRAD virtual.

5.4.8 Impacto Causado e Sofrido Pelo Algoritmo B+Tree em Ambiente Real

A Figura 5.39 apresenta o impacto causado e sofrido pelo algoritmo B+Tree em ambiente real sobre outros algoritmos em ambiente real ou virtual. Devido à diferença entre tempos de execução de cada algoritmo, foi aplicada a normalização de valores em uma escala entre 0 e 10 como apresentado na Seção 4.6 e pelas Fórmulas 4.1 e 4.2. Com estes valores normalizados é possível comparar algoritmos com tempos de grandezas diferentes.

Na figura é possível verificar que o maior impacto causado pelo algoritmo B+Tree sendo executado em ambiente real acontece quando este algoritmo implementado com biblioteca OpenCL concorre com o algoritmo SRAD executando em ambiente real implementado com biblioteca OpenMP, sendo que SRAD teve perda de desempenho normalizada 8,55 e B+Tree teve perda de desempenho normalizada 4,50.

Já com respeito ao maior impacto sofrido, ou seja, impacto causado por outro algoritmo, ocorre em duas combinações, quando B+Tree sendo executado em ambiente real implementado com biblioteca OpenCL concorre pelos mesmos recursos com Kmeans sendo executado em ambiente real implementado com biblioteca OpenMP, sendo que B+Tree teve perda de desempenho normalizada igual a 7,65 e Kmeans teve perda de desempenho normalizada igual a 3,30. A outra combinação com maior perda sofrida também com 7,65 foi causada pelo algoritmo Kmeans em ambiente virtual, sendo que este teve perda normalizada de 4,05 e foi implementado biblioteca OpenCL.

5.4.9 Impacto Causado e Sofrido Pelo Algoritmo B+Tree em Ambiente Virtual

A Figura 5.40 apresenta o impacto causado e sofrido pelo algoritmo B+Tree em ambiente virtual sobre outros algoritmos em ambiente real ou virtual. Devido à diferença entre

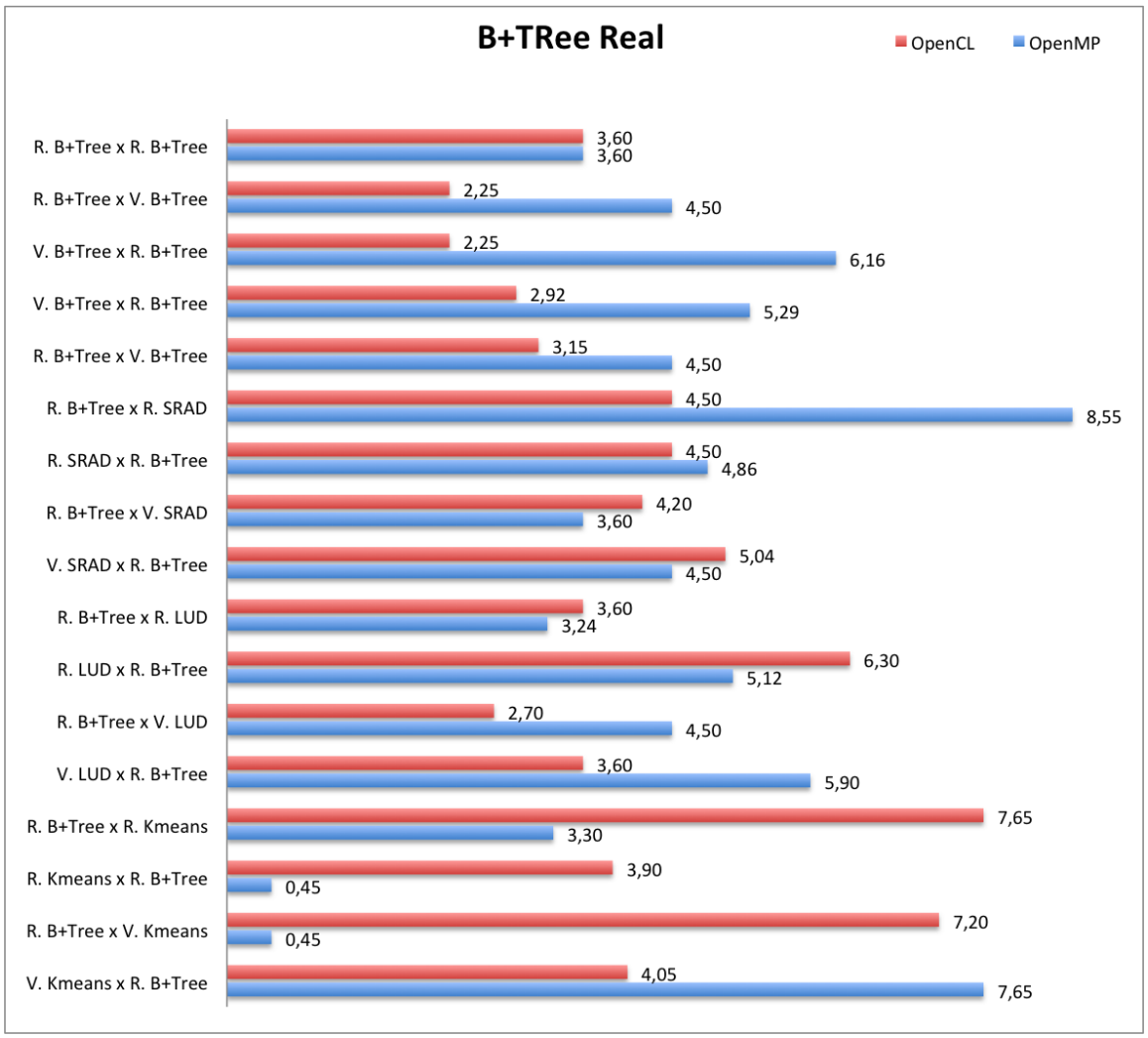

Figura 5.39: Impacto sofrido e causado pelo algoritmo B+Tree real.

tempos de execução de cada algoritmo, foi aplicada a normalização de valores em uma escala entre 0 e 10 como apresentado na Seção 4.6 e pelas Fórmulas 4.1 e 4.2. Com estes valores normalizados é possível comparar algoritmos com tempos de grandezas diferentes.

Na figura é possível verificar que o maior impacto causado pelo algoritmo B+Tree sendo executado em ambiente virtual acontece quando este algoritmo implementado com biblioteca OpenCL concorre com o mesmo algoritmo B+Tree executando em ambiente real implementado com biblioteca OpenMP, sendo que SRAD implementado com OpenCL executando em ambiente virtual teve perda de desempenho normalizada 2,25 e B+Tree implementado com OpenMP executando em ambiente real teve perda de desempenho normalizada 6,16.

Já com respeito ao maior impacto sofrido, ou seja, impacto causado por outro algoritmo, ocorre quando B+Tree sendo executado em ambiente virtual implementado com biblioteca OpenMP concorre pelos mesmos recursos com Kmeans sendo executado em ambiente real implementado com biblioteca OpenCL, sendo que B+Tree teve perda de desempenho normalizada igual a 7,65 e Kmeans teve perda de desempenho normalizada igual a 5,85.

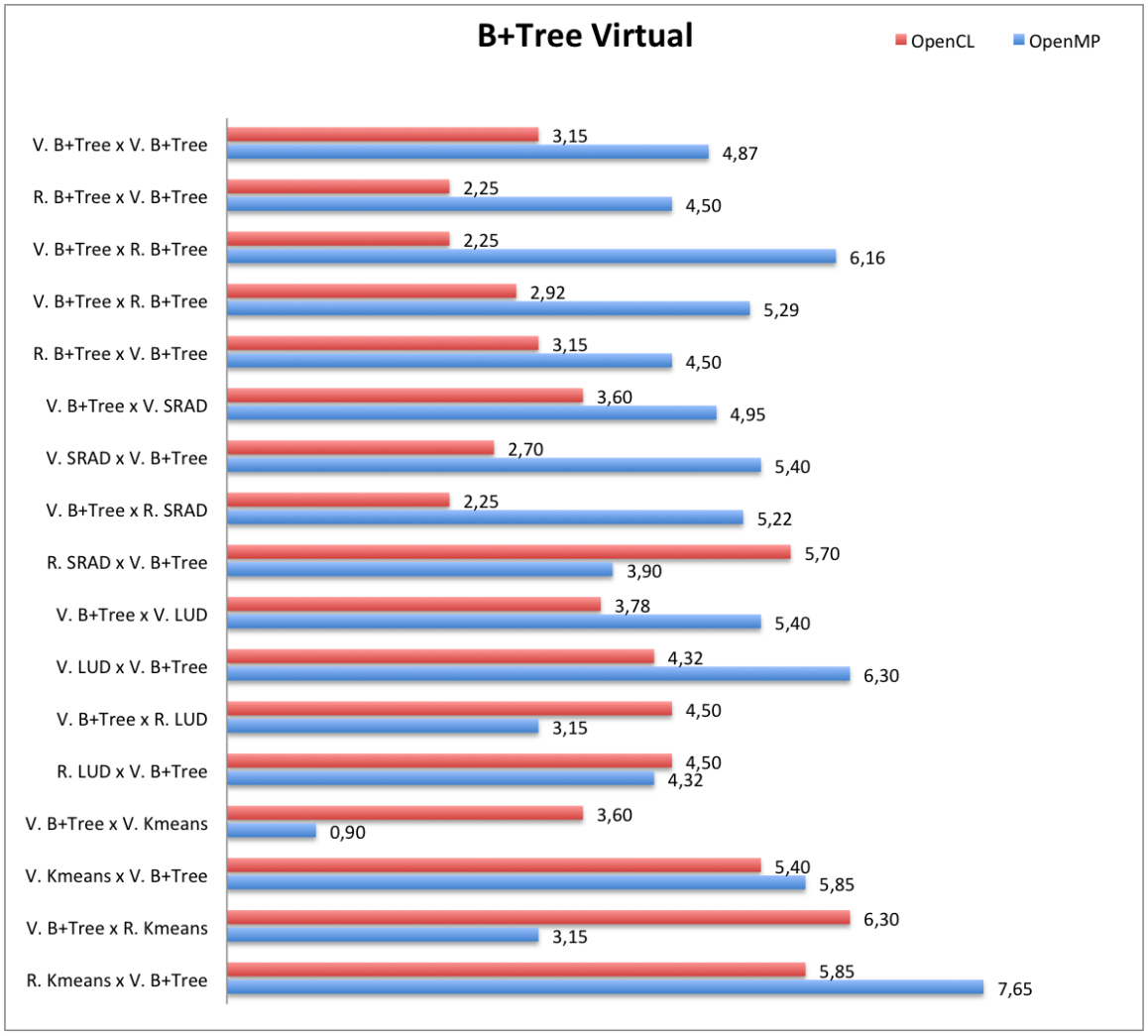

Figura 5.40: Impacto sofrido e causado pelo algoritmo B+Tree virtual.

5.5 Consolidação dos Resultados

Esta seção apresenta a consolidação dos resultados obtidos de todos os testes de concorrência. As figuras foram divididas da seguinte forma:

- **Avaliação Homogênea dos Resultados com OpenCL -** Porcentagem comparativa dos impactos sofridos em cada algoritmo implementado com biblioteca OpenCL pela concorrência com outro algoritmo implementado com biblioteca OpenCL;
- **Avaliação Homogênea dos Resultados com OpenMP -** Porcentagem comparativa dos impactos sofridos em cada algoritmo implementado com biblioteca OpenMP pela concorrência com outro algoritmo implementado com biblioteca OpenMP;
- **Avaliação Heterogênea dos Resultados com OpenCL e OpenMP Concorrentes -** Porcentagem comparativa dos impactos sofridos em cada algoritmo implementado com biblioteca OpenCL ou OpenMP pela concorrência com outro algoritmo implementado com biblioteca OpenMP ou OpenCL.

A representação permite de forma visual analisar os maiores e menores impactos com a concorrência de cada algoritmo, além de permitir o uso das tabelas por algoritmos de escalonamento. Os resultados são apresentados a seguir.

5.5.1 Avaliação Homogênea dos Resultados com OpenCL

A Figura 5.41 consolida os impactos causados pela concorrência entre algoritmos homogêneos usando biblioteca OpenCL.

Na figura é possível verificar quais os algoritmos que podem conviver em um mesmo recurso físico e quais não podem. A representação desta figura traz duas visualizações importantes:

1. A porcentagem de perda do algoritmo implementado com biblioteca OpenCL, concorrendo com outro algoritmo implementado com biblioteca OpenCL, onde as linhas

são o algoritmo de base e as colunas da matriz são os algoritmos concorrentes (impactantes);

2. A convenção utilizada é que quanto mais escuro o tom da célula na planilha, pior foi a combinação, ou seja, são das combinações que devem ser evitadas e por conseguinte as mais claras as combinações que podem coexistir.

Caso Homogêneo OpenCL								
X	Real LUD	Virtual LUD	Real Kmeans	Virtual Kmeans	Real SRAD	Virtual SRAD	Real B+Tree	Virtual B+Tree
Real LUD	85% / 85%	40% / 85%	57% / 95%	58% / 97%	112% / 70%	20% / 81%	106% / 85%	96% / 64%
Virtual LUD	85% / 40%	80% / 80%	50% / 97%	59% / 105%	758% / 11%	132% / 21%	72% / 89%	85% / 99%
Real Kmeans	95% / 57%	97% / 50%	65% / 65%	46% / 63%	342% / 12%	48% / 50%	95% / 44%	84% / 46%
Virtual Kmeans	97% / 58%	105% / 59%	63% / 46%	61% / 61%	246% / 8%	131% / 22%	77% / 44%	74% / 48%
Real SRAD	70% / 112%	11% / 758%	12% / 342%	8% / 246%	70% / 70%	14% / 108%	51% / 110%	5% / 169%
Virtual SRAD	81% / 20%	21% / 132%	50% / 48%	22% / 131%	108% / 14%	32% / 32%	57% / 32%	15% / 78%
Real B+Tree	85% / 106%	89% / 72%	44% / 95%	44% / 77%	110% / 51%	32% / 57%	99% / 99%	115% / 64%
Virtual B+Tree	64% / 96%	99% / 85%	46% / 84%	48% / 74%	169% / 5%	78% / 15%	64% / 115%	102% / 102%

Figura 5.41: Tabela Comparativa do Impacto Causado com Biblioteca OpenCL e Algoritmo Homogêneo.

Análise dos resultados:

1. A menor perda de desempenho sofrida por um algoritmo foi com o algoritmo B+Tree executado em ambiente virtual (perda de 5%), quando executado concorrentemente com SRAD em ambiente real, entretanto, a perda do algoritmo SRAD é de 169% para a mesma combinação;

2. Deve-se evitar a concorrência do algoritmo SRAD executando em ambiente real com qualquer outro algoritmo/ambiente, pois as perdas de desempenho variaram de 70% à 758%;

3. A melhor combinação possível e também mais estável (menor variação entre pontos) acontece com o uso do SRAD em ambiente virtual concorrendo com outro algoritmo SRAD em ambiente virtual (perda de 32% para ambos);

5.5.2 Avaliação Homogênea dos Resultados com OpenMP

A Figura 5.42 Apresenta o impacto causado pela concorrência entre algoritmos homogêneos usando biblioteca OpenMP. Na figura é possível verificar quais os algoritmos que podem conviver em um mesmo recurso físico e quais não podem. A representação desta figura traz duas visualizações importantes:

1. A porcentagem de perda do algoritmo implementado com biblioteca OpenMP, concorrendo com outro algoritmo implementado com biblioteca OpenMP, onde as linhas são o algoritmo de base e as colunas da matriz são os algoritmos concorrentes (impactantes);

2. A convenção utilizada é que quanto mais escuro o tom da célula na planilha, pior foi a combinação, ou seja, são das combinações que devem ser evitadas e por conseguinte as mais claras as combinações que podem coexistir.

Caso Homogêneo OpenMP

X	Real LUD	Virtual LUD	Real Kmeans	Virtual Kmeans	Real SRAD	Virtual SRAD	Real B+Tree	Virtual B+Tree
Real LUD	107% / 107%	147% / 139%	36% / 103%	25% / 137%	117% / 80%	113% / 143%	37% / 41%	9% / 16%
Virtual LUD	139% / 147%	176% / 176%	52% / 116%	40% / 211%	153% / 107%	123% / 124%	49% / 70%	13% / 66%
Real Kmeans	103% / 36%	116% / 52%	53% / 53%	48% / 49%	115% / 49%	109% / 54%	4% / 8%	5% / 4%
Virtual Kmeans	137% / 25%	211% / 40%	49% / 48%	49% / 49%	149% / 33%	129% / 45%	4% / 4%	4% / 5%
Real SRAD	80% / 117%	107% / 153%	49% / 115%	33% / 149%	129% / 129%	115% / 156%	50% / 104%	12% / 61%
Virtual SRAD	143% / 113%	124% / 123%	54% / 109%	45% / 129%	156% / 115%	130% / 130%	51% / 96%	16% / 60%
Real B+Tree	41% / 37%	70% / 49%	8% / 4%	4% / 4%	104% / 50%	96% / 51%	102% / 102%	6% / 14%
Virtual B+Tree	16% / 9%	66% / 13%	4% / 5%	5% / 4%	61% / 12%	60% / 16%	14% / 6%	16% / 16%

Figura 5.42: Tabela Comparativa do Impacto Causado com Biblioteca OpenMP e Algoritmo Homogêneo.

Análise dos resultados:

1. A pior perda de desempenho foi de 211% no algoritmo LUD em ambiente virtual concorrentemente com o algoritmo Kmeans em ambiente virtual com perda de 40%;

2. Apesar da estabilidade do ambiente, a concorrência do algoritmo LUD em ambiente virtual com outro algoritmo LUD em ambiente virtual se mostrou a que mais causou perda para ambos (176% de perda de desempenho para os dois).

3. As melhores combinações são:

   (a) B+Tree em ambiente real (perda de 4%) concorrendo com Kmeans em ambiente virtual (perda de 4%);

   (b) B+Tree em ambiente virtual (perda de 5%) concorrendo com Kmeans em ambiente real (perda de 4%);

   (c) B+Tree em ambiente virtual (perda de 4%) concorrendo com Kmeans em ambiente virtual (perda de 4%);

   (d) B+Tree em ambiente real (perda de 4%) concorrendo com Kmeans em ambiente real (perda de 8%);

5.5.3 Avaliação Heterogênea dos Resultados

A Figura 5.43 Apresenta o impacto causado pela concorrência entre algoritmos heterogêneos usando biblioteca OpenCL e OpenMP. Na figura é possível verificar quais os algoritmos que podem conviver em um mesmo recurso físico e quais não podem. A representação desta figura traz duas visualizações importantes:

1. A porcentagem de perda do algoritmo implementado com biblioteca OpenCL ou OpenMP, concorrendo com outro algoritmo implementado com biblioteca OpenMP ou OpenCL;

2. A convenção utilizada é que quanto mais escuro o tom da célula na planilha, pior foi a combinação, ou seja, são das combinações que devem ser evitadas e por conseguinte as mais claras as combinações que podem coexistir.

Análise dos resultados:

Caso Heterogêneo OpenCL x OpenMP									
X		OpenMP							
		Real LUD	Virtual LUD	Real Kmeans	Virtual Kmeans	Real SRAD	Virtual SRAD	Real B+Tree	Virtual B+Tree
OpenCL	Real LUD	37% / 105%	48% / 141%	70% / 77%	76% / 87%	306% / 113%	309% / 115%	19% / 23%	22% / 19%
	Virtual LUD	124% / 41%	179% / 55%	72% / 60%	69% / 81%	510% / 52%	419% / 61%	4% / 8%	14% / 18%
	Real Kmeans	73% / 72%	19% / 0%	89% / 45%	76% / 47%	415% / 65%	404% / 69%	30% / 72%	32% / 71%
	Virtual Kmeans	125% / 32%	123% / 43%	96% / 38%	78% / 50%	514% / 42%	398% / 83%	4% / 66%	4% / 65%
	Real SRAD	3% / 636%	1% / 811%	0% / 721%	6% / 711%	185% / 722%	166% / 836%	47% / 561%	47% / 431%
	Virtual SRAD	30% / 81%	46% / 87%	15% / 76%	10% / 464%	270% / 116%	247% / 106%	4% / 0%	4% / 0%
	Real B+Tree	135% / 98%	46% / 141%	70% / 62%	65% / 75%	317% / 88%	310% / 93%	12% / 6%	16% / 5%
	Virtual B+Tree	88% / 50%	107% / 170%	65% / 58%	61% / 78%	460% / 47%	331% / 44%	12% / 8%	5% / 5%

Figura 5.43: Tabela Comparativa do Impacto Causado nos Algoritmos Implementados com Biblioteca OpenCL pela Concorrência com Algoritmos Implementados com Biblioteca OpenMP.

1. Todas as combinações de SRAD em ambiente real implementadas com biblioteca OpenCL, concorrendo com algoritmos implementados com bibliotecas OpenMP devem ser evitadas, já que a menor perda acarretada a este algoritmo foi de 431% concorrendo com B+Tree executando em ambiente virtual e a maior perda foi de 836% concorrendo com SRAD em ambiente virtual ou LUD em ambiente real;

2. Todas as combinações de SRAD em ambiente real ou virtual implementadas com biblioteca OpenMP, concorrendo com algoritmos implementados com bibliotecas OpenCL devem ser evitadas, já que a menor perda acarretada a este algoritmo foi de 166% concorrendo com SRAD executando em ambiente real e a maior perda foi de 514% concorrendo com Kmeans em ambiente virtual;

3. Os algoritmos que tiveram menor perda foram:

   (a) B+Tree em ambiente real ou virtual implementados com biblioteca OpenMP concorrentemente com SRAD executando em ambiente virtual implementado com biblioteca OpenCL (sem perda para o algoritmo SRAD e perda de 4% para os algoritmos B+Tree em ambiente real ou virtual);

(b) B+Tree em ambiente virtual implementado com biblioteca OpenMP concorrentemente com B+Tree em ambiente virtual implementado com biblioteca OpenCL (perda de 5% para os dois algoritmos);

(c) LUD em ambiente virtual implementado com OpenCL concorrentemente com B+Tree em executando em ambiente real implementado com biblioteca OpenMP (perda de 8% e 4% respectivamente).

Outras combinações podem coexistir, de acordo com as políticas de acesso e controle do ambiente, avaliando-se principalmente a estabilidade do ambiente (menor distância entre pontos) e a qualidade (menor perda média). Cabendo ao administrador do ambiente, ou ao algoritmo de escalonamento, decidir a melhor combinação à partir dos dados gerados.

CAPÍTULO 6

CONCLUSÕES

Este trabalho apresentou uma análise sobre os efeitos da concorrência entre classes algoritmos utilizados em aplicações, executados em ambientes virtuais e reais. Nesta análise foi possível observar a diversidade de efeitos causados por estas combinações, associadas aos efeitos causados pelas bibliotecas utilizadas na implementação dos mesmos. O modelo proposto de avaliação destes efeitos (Afinidade) possibilita que os dados coletados se tornem a base de informações e uma fonte de consulta para averiguar se determinada aplicação está se comportando como deveria, ou se está sendo degradada ou degradando o ambiente quando da execução com algum outro algoritmo concorrentemente.

Este estudo torna-se importante na medida que o uso de ambientes virtuais tem se tornado cada vez mais comum, como forma de otimizar o uso de recursos reais (consolidação) ou em que a computação em nuvem é apresentada como uma solução para a obtenção de recursos adicionais sob demanda. Os resultados obtidos têm a intenção de fornecer subsídios para a melhor combinação entre as várias aplicações que compartilham desses recursos.

A abordagem da análise do ambiente real, mesmo considerando o foco do trabalho no uso de nuvens computacionais, e neste caso, a degradação causada em ambientes virtualizados, se mostrou necessária já que no ambiente real executando algoritmos concorrentemente, uma degradação causada por esta concorrência degradaria todo o ambiente e consequentemente os ambientes virtuais hospedados nestes ambientes reais.

Um exemplo desta degradação que os ambientes reais podiam sofrer ou causar seria a execução de um servidor web, por exemplo, que concorrendo com outros servidores vir-

tualizados poderiam degradar estes e consequentemente todo o ambiente, bastando para a análise, identificar a qual classe de algoritmos o servidor web estaria associado.

Os resultados dos experimentos mostraram que, se há real necessidade de compartilhamento de recursos e, consequentemente há concorrência pelo recurso físico, alguns tipos de aplicações podem coexistir sem significativa degradação do meio, permitindo este compartilhamento, enquanto que outras combinações de aplicações devem ser evitadas.

Como uma extensão do presente trabalho, os resultados e as conclusões aqui apresentadas, podem ser utilizadas para desenvolvimento de aplicações e escalonadores para os ambientes testados, afim de minimizar a perda de desempenho resultante da concorrência.

Para avaliar o ambiente, a abordagem adotada foi o uso do conceito de classes de Dwarfs. Os quatro algoritmos utilizados para os testes foram escolhidos porque cobrem a maioria dos tipos de aplicações existentes, sendo estas científicas ou comerciais.

Os resultados apresentados, além de medir o grau de perda de desempenho durante o compartilhamento de recursos, também permitiu verificar a influência da combinação entre os algoritmos, assim como a ordem em que é feita a execução das aplicações. Por exemplo, no caso da aplicação LUD implementada em OpenMP, quando executada em um ambiente real e a instanciação de um ambiente virtual com o mesmo algoritmo simultaneamente, acarreta perda média de 136 % para a aplicação no ambiente real e 144 % para o ambiente virtual, mas no caso de um ambiente virtual rodando com a mesma aplicação, quando outro aplicativo foi instanciado no ambiente real, a perda foi de 147 % (144 %, antes) e 139 % (136 % antes), respectivamente.

Outra contribuição importante foi o efeito causado pelo tipo de biblioteca paralela utilizada para implementar os algoritmos. Foram avaliadas as bibliotecas OpenMP e OpenCL que mostraram variação do tempo de execução e influência quando há concorrência com

outros algoritmos. Um resultado a ser destacado nesta análise foi o caso dos testes que avaliaram o algoritmo SRAD em um ambiente real, sendo executado simultaneamente com o algoritmo LUD encapsulado em um ambiente virtualizado utilizando bibliotecas OpenCL, cuja perda de desempenho foi de 758 % para algoritmo SRAD, enquanto o mesmo teste, agora com a utilização de bibliotecas OpenMP causou uma perda de 153 % para SRAD.

A Tabela 5.3 apresentou uma síntese dos resultados para o caso da combinação homogênea, que mostram os tipos de algoritmos que podem ser executados simultaneamente com menor perda de desempenho e o melhor tipo de biblioteca paralela a ser usada por cada algoritmo, como apresentado na Seção 4.5 e no Capítulo 5.

Os resultados obtidos permitiram propor então o conceito de afinidade, o qual foi caracterizado pelo o grau de compatibilidade entre as classes de aplicações, cuja execução concorrente no mesmo ambiente de computação resultaria em uma perda mínima para estas aplicações e o próprio ambiente, e a sua utilização em apoio ao compartilhamento de recursos virtuais ou reais.

As Figuras 5.41, 5.42 e 5.43 apresentam um resumo de todos os resultados obtidos nos testes realizados. Os valores lá apresentados consolidam todas as perdas nas combinações testadas, indicando os tipos de concorrência que podem coexistir em um mesmo recurso e quais os que devem ser evitados. Com os dados disponibilizados por estas tabelas, um escalonador de tarefas pode escolher as melhores combinações, caso seja necessário compartilhar recursos e quais das combinações devem ser evitadas para que não haja prejuízo do desempenho das aplicações sendo executadas, bem como do ambiente real ou virtualizado.

O modelo de pontuação de recurso também é considerado uma inovação, já que até este momento não foi encontrado nenhum mecanismo de pontuação que permitisse avaliar

estes algoritmos e linguagens. Ele procura equalizar estes valores de ordens de grandeza diferentes, como o caso dos tempos de execução dos algoritmos. Tem por propriedade avaliar todo o conjunto de algoritmos testados por meio de pontuações máximas e mínimas de forma pré-definida, possibilitando uma análise quantitativa dos impactos resultantes da concorrência.

O modelo de pontuação permite criar métricas de desempenho capazes de avaliar a infraestrutura computacional existente de forma a verificar o seu desempenho em função das aplicações nele alocadas, seja em ambientes reais ou virtuais. Estas métricas podem ser utilizadas de forma a flexibilizar o uso desse ambiente. Isto é possível, na medida em que se conhece o ambiente e os efeitos da concorrência, e neste sentido é possível delimitar a escolha do melhor ambiente para executar o algoritmo de forma concorrente, optando entre o melhor tempo de resposta (menor influência de outro algoritmo) ou maior estabilidade (quando o tempo de resposta não é crítico), por exemplo.

Quanto às hipóteses e aos questionamentos formulados, foi possível verificar que o compartilhamento de recursos pode causar perdas na execução das aplicações nele hospedadas e o grau de degradação observado é dependente da combinação dos tipos de algoritmos das aplicações executadas concorrentemente com graus bem diversificados.

O uso de classes de aplicações se mostrou uma abordagem adequada para a determinação do grau de degradação causado pela concorrência de aplicações.

Verificou-se que há classes de aplicações que podem coexistir em um mesmo ambiente causando pouca degradação e estas classes poderiam compartilhar o mesmo recurso, quando fosse necessário a existência de concorrência, e há classes de aplicações que causam grande degradação do ambiente, e neste caso devem ser evitadas.

Mostrou-se que o tipo de biblioteca utilizada na implementação dos algoritmos (OpenCL

e OpenMP) afeta a combinação de classes que podem concorrer e classes que não podem concorrer.

A avaliação da média de perda de desempenho, quando há concorrência heterogênea se mostrou uma boa abordagem para definir que combinações de bibliotecas podem competir por um mesmo recurso e quais não podem competir. Na mesma avaliação, através do cálculo da distância entre pontos, permitiu-se definir que ambientes/algoritmos/bibliotecas possuem a maior estabilidade do ambiente, e quais aquelas combinações que possuem a maior instabilidade no ambiente.

Na avaliação heterogênea (OpenCL e OpenMP) percebeu-se em todos os algoritmos que a combinação do tipo de biblioteca utilizada acarreta variação significativa de perda de desempenho do ambiente.

As tabelas comparativas de desempenho concorrente apresentadas na Seção 5.5 permitem a análise visual dos resultados, já que é possível avaliar a distinção entre os maiores impactos da concorrência (tonalidades mais escuras) e os menores impactos da concorrência (tonalidades mais claras). Possibilitando então que sejam avaliadas rapidamente as melhores combinações, bem como as combinações que não devem coexistir.

Em resumo, nesta tese, avaliou-se que o grau de degradação causado pela concorrência no uso dos recursos é diferenciado e dependente dos tipos de aplicações combinadas neste recurso, e que este grau pode ser avaliado com o uso do conceito de classes de aplicações. À proposta de avaliação deste grau deu-se o nome de “Afinidade”. Com este tipo de abordagem é possível mensurar os impactos das combinações pretendidas e sua utilização para melhor alocar as aplicações nos recursos existentes.

6.1 Considerações Finais

Neste trabalho pôde-se definir dois tipos de Afinidade, o primeiro com respeito aos critérios quantitativos, onde se analisou as menores perdas de desempenho, o segundo com respeito à maior estabilidade da execução e do ambiente. Destes dois tipos de Afinidade é possível propor algoritmos que podem definir quais as melhores combinações de recursos disponíveis.

Como proposta da criação de escalonadores, um algoritmo trabalharia com duas matrizes para cada servidor, uma com o grau de degradação causado, apresentado nas Figuras 5.41, 5.42 e 5.43. Na segunda matriz haveria os algoritmos sendo executados. Como, no caso dos algoritmos testados, houve um aumento em progressão aritmética, por exemplo, se com um algoritmo, concorrentemente, outro fosse executado e avaliou-se 10 % de perda, com mais um algoritmo concorrente a perda seria de 20 %.

O escalonador verificaria quais os servidores "reais"que poderiam executar um algoritmo sem degradar o ambiente, já que há informações dos tipos de algoritmos que não poderiam compartilhar o meio, e da mesma forma, quais os que poderiam concorrer pelos mesmos recursos.

6.2 Trabalhos Futuros

Como proposta de trabalhos futuros sugere-se os testes com todos os outros algoritmos do pacote Rodínia. Além disso, dos dados extraídos até o momento (mais de 7000 testes) é possível fazer uma consolidação dos resultados e extrair informações relevantes à tomada de decisão para escalonadores.

Há intenção ainda de comparar e executar os mesmos algoritmos em GPUs para verificar a afinidade de aplicações neste tipo de arquitetura. A arquitetura GPGPU torna-se uma avaliação interessante na medida que é possível realizar os testes de concorrência, tanto no modo GPU, ou a combinação do uso CPU x GPU.

O uso da Fórmula 4.3 de normalização com pesos pode ser aplicada à escolha do melhor tipo de ambiente para executar determinado tipo de aplicação, contribuindo assim para formulação de novos escalonadores.

BIBLIOGRAFIA

[1] *Motor: A Virtual Machine for High Performance Computing*, 2006.

[2] *Engineering Statistics 4E Student Study Edition with Student Solutions Manual Package*. John Wiley & Sons, Incorporated, 2009.

[3] *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012, Ottawa, Canada, May 13-16, 2012*. IEEE, 2012.

[4] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, e Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, abril de 2010.

[5] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, e Katherine A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Relatório Técnico UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec de 2006.

[6] Maha JEBALIA Sami TABBANE Asma BEN LETAIFA, Amel HAJI. State of the art and research challenges of new services architecture technologies: Virtualization, soa and cloud computing. *International Journal of Grid and Distributed Computing*, 3:69–88, 2010.

[7] Lee Badger, Tim Grance, Robert Patt-Corner, e Jeff Voas. DRAFT Cloud Computing Synopsis and Recommendations. Relatório técnico, U.S. Department of Commerce Gary Locke, Secretary National Institute of Standards and Technology Patrick D. Gallagher, Director.

[8] Daniel Beimborn, Thomas Miletzki, e Stefan Wenzel. Platform as a service (paas). *Business & Information Systems Engineering*, 3(6):381–384, 2011.

[9] Berkley. A dwarf is an algorithmic method that captures a pattern of computation and communication, 2013. [Online; accessed 06-Outubro-2013].

[10] Paolo Bientinesi, Roman Iakymchuk, e Jeff Napper. *HPC on Competitive Cloud Resources*. Springer US, 2010.

[11] L. Susan Blackford, Jaeyoung Choi, Andrew J. Cleary, James Demmel, Inderjit S. Dhillon, Jack Dongarra, Sven Hammarling, Greg Henry, Antoine Petitet, Ken Stanley, David W. Walker, e R. Clinton Whaley. ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance. *Proceedings of the 1996 ACM/IEEE conference on Supercomputing*, páginas 5. IEEE, 1996.

[12] Rajkumar Buyya, Chee Shin Yeo, e Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. *Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, HPCC '08, páginas 5–13, Washington, DC, USA, 2008. IEEE Computer Society.

[13] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, e Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, junho de 2009.

[14] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, e Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, janeiro de 2011.

[15] Fabrice ; MEHAUT Jean-François ; POLANCZYK Rafael Vanoni CARISSIMI, A. S. ; DUPROS. Aspectos de programação paralela em máquinas numa. *Workshops, competitions and minicourses.* Sociedade Brasileira de Computação (SBC), 2007.

[16] CERN. Helix nebula the science cloud: A catalyst for change in europe. http://cds.cern.ch/record/1537032/files/HelixNebula-2013-002.pdf, 2013.

[17] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, e Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. *IISWC*, páginas 44–54. IEEE, 2009.

[18] Xingchen Chu, Krishna Nadiminti, Chao Jin, Srikumar Venugopal, e Rajkumar Buyya. Aneka: Next-generation enterprise grid platform for e-science and e-business. *Applications, Proceedings of the 3 rd IEEE International Conference and Grid Computing*, páginas 10–13. IEEE CS Press, 2007.

[19] Inc. All rights reserved. Citrix Systems. Site do xen, 2012. [Online; accessed 06-Outubro-2012].

[20] Phillip Colella. Defining software requirements for scientific computing. DARPA HPCS presentation, 2004.

[21] Éfrem R. A. Moraes João H. C. Pimentel Rodrigo Q. Freitas Cássio A. Melo, Daniel F. Arcoverde. Software como serviço: Um modelo de negócio emergente, 2007. [Online; accessed 02-Julho-2014].

[22] Wesam Dawoud, Ibrahim Takouna, e Christoph Meinel. Elastic vm for cloud resources provisioning optimization. Ajith Abraham, Jaime Lloret Mauri, John F. Buford, Junichi Suzuki, e Sabu M. Thampi, editors, *ACC (1)*, volume 190 of *Communications in Computer and Information Science*, páginas 431–445. Springer, 2011.

[23] Elder de Macedo Rodrigues. Realocação de recursos em ambientes virtualizados. Mestrado em ciência da computação, 2009.

[24] Tyler Dwyer, Alexandra Fedorova, Sergey Blagodurov, Mark Roth, Fabien Gaud, e Jian Pei. A practical method for estimating performance degradation on multicore processors, and its application to hpc workloads. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*,

SC Í2, páginas 83:1–83:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[25] EC2. Página do projeto amazon elastic compute cloud (amazon ec2), 2012. [Online; accessed 03-Outubro-2012].

[26] Jaliya Ekanayake e Geoffrey Fox. High performance parallel computing with clouds and cloud technologies. Dimiter R. Avresky, Michel Diaz, Arndt Bode, Bruno Ciciani, e Eliezer Dekel, editors, *CloudComp*, volume 34 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, páginas 20–38. Springer, 2009.

[27] Yaakoub El-Khamra, Hyunjoo Kim, Shantenu Jha, e Manish Parashar. Exploring the performance fluctuations of hpc workloads on clouds. *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, páginas 383–387, Washington, DC, USA, 2010. IEEE Computer Society.

[28] Vegard Engen, Juri Papay, Stephen C. Phillips, e Michael Boniface. Predicting application performance for multi-vendor clouds using dwarf benchmarks. *Proceedings of the 13th international conference on Web Information Systems Engineering*, WISE'12, páginas 659–665, Berlin, Heidelberg, 2012. Springer-Verlag.

[29] Eucalyptus. Página do projeto eucalyptus, 2012. [Online; accessed 03-Outubro-2012].

[30] Luis Carlos E. de Bona Fabio L. Licht, Bruno R. Schulze. QUALIDADE DE SERVIÇO EM NUVENS COMPUTACIONAIS: UMA AVALIAÇÃO DE RECURSOS EM AMBIENTES DISTRIBUÍDOS. *Journal of Engineering of Catholic University of Petrópolis*, 8(1):1–13, 2013.

[31] Dror G. Feitelson e Larry Rudolph. Towards convergence in job schedulers for parallel supercomputers. *Proceedings of the Workshop on Job Scheduling Strategies*

for Parallel Processing, IPPS '96, páginas 1–26, London, UK, UK, 1996. Springer-Verlag.

[32] Matteo Frigo, Steven, e G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, páginas 216–231, 2005.

[33] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, e Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. *Proceedings, 11th European PVM/MPI Users' Group Meeting*, páginas 97–104, Budapest, Hungary, September de 2004.

[34] Khronos Group. The open standard for parallel programming of heterogeneous systems, 2014. [Online; accessed 02-Junho-2014].

[35] IBM. Alta disponibilidade, 2011. [Online; accessed 11-Junho-2012].

[36] Keith R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J. Wasserman, e Nicholas J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, páginas 159–168, Washington, DC, USA, 2010. IEEE Computer Society.

[37] Kaustubh R. Joshi, Guy Bunker, Farnam Jahanian, Aad P. A. van Moorsel, e Joseph Weinman. Dependability in the cloud: Challenges and opportunities. *DSN*, páginas 103–104. IEEE, 2009.

[38] Alex Kaiser, Samuel Williams, Kamesh Madduri, Khaled Ibrahim, David Bailey, James Demmel, e Erich Strohmaier. TORCH Computational Reference Kernels: A Testbed for Computer Science Research. Relatório Técnico UCB/EECS-2010-144, EECS Department, University of California, Berkeley, Dec de 2010.

[39] Erich L. Kaltofen. The seven dwarfs of symbolic computation*, 2010.

[40] Melanie Kambadur, Tipp Moseley, Rick Hank, e Martha A. Kim. Measuring interference between live datacenter applications. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC Í2, páginas 51:1–51:12, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[41] Kate Keahey, Patrick Armstrong, John Bresnahan, David LaBissoniere, e Pierre Riteau. Infrastructure outsourcing in multicloud environment. *Proceedings of the Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit*, San Jose, CA (USA), 2012.

[42] Iraklis A. Klampanosa e Joemon M. Joseb. Searching in peer-to-peer networks. *Computer Science Review*.

[43] Ian Korf, Mark Yandell, e Joseph Bedell. *BLAST*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.

[44] C12G Labs. Site do projeto opennebula de nuvens computacionais, 2012. [Online; accessed 06-Outubro-2012].

[45] C.H.N. Lahoz. *O processo de elicitação de metas de dependabilidade para sistemas computacionais críticos: estudo de caso aplicado a área espacial*. 2009.

[46] Craig A. Lee. A perspective on scientific cloud computing. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, páginas 451–459, New York, NY, USA, 2010. ACM.

[47] Amazon Web Services LLC. Site do amazon ec2, 2012. [Online; accessed 06-Outubro-2012].

[48] Heiko Ludwig, Alexander Keller, Asit Dan, Richard King, e Richard Franck. A service level agreement language for dynamic electronic services. *Electronic Commerce Research*, 3:43–59, 2003. 10.1023/A:1021525310424.

[49] Dale L. Lunsford, East B. Blvd, e Long Beach. Virtualization Technologies in Information Systems Education. 20(3):339–349.

[50] Laion F. Manfroi, Mariza Ferro, Andre Yokoyama, Antonio Roberto Mury, e Bruno Schulze. A walking dwarf on the clouds. *IEEE/ACM 6th International Conference on Utility and Cloud Computing*, páginas 399–404, Dresden, Germany, 2013. International Workshop on Clouds and (eScience) Applications Management (CloudAM2013), IEEE Computer Society.

[51] Sudha Mani e Shrisha Rao. Operating cost aware scheduling model for distributed servers based on global power pricing policies. *Proceedings of the Fourth Annual ACM Bangalore Conference*, COMPUTE '11, páginas 12:1–12:8, New York, NY, USA, 2011. ACM.

[52] K A Manjula e P Karthikeyan. Cloud computing ? a paradigm shift. *Global Journal of Computer Science and Technology*, 10(6), 2010.

[53] Paul Marshall, Kate Keahey, e Tim Freeman. Elastic site: Using clouds to elastically extend site resources. *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, páginas 43–52, Washington, DC, USA, 2010. IEEE Computer Society.

[54] Mark F. Mergen, Volkmar Uhlig, Orran Krieger, e Jimi Xenidis. Virtualization for high-performance computing. *SIGOPS Oper. Syst. Rev.*, 40(2):8–11, abril de 2006.

[55] Microsoft. Plataforma microsoft azure, 2011. [Online; accessed 11-Junho-2012].

[56] Hamid R. Motahari-Nezhad, Bryan Stephenson, e Sharad Singha. Outsourcing Business to Cloud Computing Services: Opportunities and Challenges. *IEEE IT Professional, Special Issue on Cloud Computing*, 11(2), setembro de 2009.

[57] A. Munshi, B. Gaster, T.G. Mattson, e D. Ginsburg. *OpenCL Programming Guide*. OpenGL. Pearson Education, 2011.

[58] Visakh Muraleedharan. Hawk-i HPC Cloud Benchmark Tool. Msc in high performance computing, University of Edinburgh, Edinburgh, August de 2012.

[59] Iulian Neamtiu. Elastic executions from inelastic programs. *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, páginas 178–183, New York, NY, USA, 2011. ACM.

[60] K. K. Nguyen, M. Lemay, e M. Cheriet. Enabling Infrastructure as a Service (IaaS) on IP Networks: From Distributed to Virtualized Control Plane. *IEEE Communications Magazine*, (PP):1+, março de 2012.

[61] Nimbus. Página do projeto nimbus, 2012. [Online; accessed 03-Outubro-2012].

[62] Nitu. Configurability in saas (software as a service) applications. *Proceedings of the 2Nd India Software Engineering Conference*, ISEC '09, páginas 19–26, New York, NY, USA, 2009. ACM.

[63] U.S. of Department of Energy. Advanced scientific computing research (ascr), 2013. [Online; accessed 09-Outubro-2013].

[64] U.S. of Department of Energy. Department of energy (doe), 2013. [Online; accessed 09-Outubro-2013].

[65] OpenMP Architecture Review Board. *OpenMP Application Program Interface*, 3.0 edition, maio de 2008.

[66] Stephen C. Phillips, Vegard Engen, e Juri Papay. Snow White Clouds and the Seven Dwarfs. *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, páginas 738–745, Washington, DC, USA, 2011. IEEE Computer Society.

[67] Dinesh Rajan, Anthony Canino, Jesus A. Izaguirre, e Douglas Thain. Converting a high performance application to an elastic cloud application. *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and*

Science, CLOUDCOM '11, páginas 383–390, Washington, DC, USA, 2011. IEEE Computer Society.

[68] Aarthi Raveendran, Tekin Bicer, e Gagan Agrawal. A framework for elastic execution of existing mpi programs. *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '11, páginas 940–947, Washington, DC, USA, 2011. IEEE Computer Society.

[69] Vignesh T. Ravi, Michela Becchi, Gagan Agrawal, e Srimat Chakradhar. Supporting gpu sharing in cloud environments with a transparent runtime consolidation framework. *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, páginas 217–228, New York, NY, USA, 2011. ACM.

[70] Paulo Antonio Leal Rego, Emanuel Ferreira Coutinho, Danielo Goncalves Gomes, e Jose Neuman de Souza. Faircpu: Architecture for allocation of virtual machines using processing features. *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing*, UCC '11, páginas 371–376, Washington, DC, USA, 2011. IEEE Computer Society.

[71] Christiane Pousa Ribeiro, Jean-Francois Mehaut, Alexandre Carissimi, Marcio Castro, e Luiz Gustavo Fernandes. Memory affinity for hierarchical shared memory multiprocessors. *Proceedings of the 2009 21st International Symposium on Computer Architecture and High Performance Computing*, SBAC-PAD '09, páginas 59–66, Washington, DC, USA, 2009. IEEE Computer Society.

[72] Jörg Schad, Jens Dittrich, e Jorge-Arnulfo Quiané-Ruiz. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proc. VLDB Endow.*, 3(1-2):460–471, setembro de 2010.

[73] Julia Schroeter, Sebastian Cech, Sebastian Götz, Claas Wilke, e Uwe Assmann. Towards modeling a variable architecture for multi-tenant saas-applications. *Pro-*

ceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, VaMoS 12, páginas 111–120, New York, NY, USA, 2012. ACM.

[74] Peter Sempolinski e Douglas Thain. A comparison and critique of eucalyptus, opennebula and nimbus. *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, páginas 417–426, Washington, DC, USA, 2010. IEEE Computer Society.

[75] Indiana University (Main Open MPI server). Site do projeto openmpi, 2012. [Online; accessed 06-Outubro-2012].

[76] Amazon Web Services. Summary of the amazon ec2 and amazon rds service disruption in the us east region, 2013. [Online; accessed 09-Outubro-2013].

[77] Upendra Sharma, Prashant Shenoy, Sambit Sahu, e Anees Shaikh. A cost-aware elasticity provisioning system for the cloud. *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ICDCS '11, páginas 559–570, Washington, DC, USA, 2011. IEEE Computer Society.

[78] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, e John Wilkes. Cloudscale: elastic resource scaling for multitenant cloud systems. *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, páginas 5:1–5:14, New York, NY, USA, 2011. ACM.

[79] Shu sheng CAO, Wei YIN, e Xing yi CHEN. A robust cluster-based dynamic-supernode scheme for hybrid peer-to-peer network. *The Journal of China Universities of Posts and Telecommunications*, 14, Supplement 1(0):21 – 26, 2007.

[80] J.B. Silva, M. Vin'icius, e R. Netto. Confiabilidade nos servicos web: Um estudo sobre as técnicas para implementação de dependabilidade. *Proceedings of 37a Info-Brasil TI & Telecom*, 2011.

[81] Yogesh Simmhan, Catharine van Ingen, Girish Subramanian, e Jie Li. Bridging the gap between desktop and the cloud for escience applications. *Proceedings of*

the 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD '10, páginas 474–481, Washington, DC, USA, 2010. IEEE Computer Society.

[82] Bradley Simmons, Angela McCloskey, e Hanan Lutfiyya. Dynamic provisioning of resources in data centers. *Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, ICAS '07, páginas 40–, Washington, DC, USA, 2007. IEEE Computer Society.

[83] Joshua E. Simons e Jeffrey Buell. Virtualizing high performance computing. *SIGOPS Oper. Syst. Rev.*, 44(4):136–145, dezembro de 2010.

[84] Kevin Skadron. Rodinia:accelerating compute-intensive applications with accelerators, 2014. [Online; accessed 02-Junho-2014].

[85] D. Skinner e W. Kramer. Understanding the causes of performance variability in hpc workloads. *2013 IEEE International Symposium on Workload Characterization (IISWC)*, 0:137–149, 2005.

[86] Jason Sonnek e Abhishek Chandra. Virtual putty. *Proceedings of the 2009 conference on Hot topics in cloud computing*, HotCloud'09, Berkeley, CA, USA, 2009. USENIX Association.

[87] Ahmed A. Soror, Umar Farooq Minhas, Ashraf Aboulnaga, Kenneth Salem, Peter Kokosielis, e Sunil Kamath. Automatic virtual machine configuration for database workloads. *ACM Trans. Database Syst.*, 35(1):7:1–7:47, fevereiro de 2008.

[88] Leonardo O.; Machado Javam C. Sousa, Flavio R. C.; Moreira. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. *ERCEMAPI. Todos os direitos reservados a EDUFPI.*, 2009.

[89] Paul Springer. Berkeley's Dwarfs on CUDA. Relatório técnico, RWTH Aachen University, 2011. Seminar Project.

[90] Jeffrey M. Squyres. *The Archicture of Open Source Applications*, volume ii, capítulo 15. Self published, April de 2012.

[91] John A. Stratton, Christopher Rodrigrues, I-Jui Sung, Nady Obeid, Liwen Chang, Geng Liu, e Wen-Mei W. Hwu. Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing. Relatório Técnico IMPACT-12-01, University of Illinois at Urbana-Champaign, Urbana, março de 2012.

[92] A.S. Tanenbaum. *Modern Operating Systems*. GOAL Series. Prentice Hall, 2008.

[93] CA Technologies. *The Complete Guide To Monitoring Virtualized Environments*. CA Technologies, 2013.

[94] Juan M. Tirado, Daniel Higuero, Florin Isaila, Jesús Carretero, e Adriana Iamnitchi. Affinity p2p: A self-organizing content-based locality-aware collaborative peer-to-peer network. *Computer Networks*, 54(12):2056 – 2070, 2010. ¡ce:title¿P2P Technologies for Emerging Wide-Area Collaborative Services and Applications¡/ce:title¿.

[95] Radu Tudoran, Alexandru Costan, Gabriel Antoniu, e Luc Bougé. A performance evaluation of azure and nimbus clouds for scientific applications. *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*, CloudCP '12, páginas 4:1–4:6, New York, NY, USA, 2012. ACM.

[96] Energy Department U.S. The magellan report on cloud computing for science. http://magellan.alcf.anl.gov/, 2011.

[97] Luis M. Vaquero, Luis RoderoMerino, e Rajkumar Buyya. Dynamically scaling applications in the cloud. *SIGCOMM Comput. Commun. Rev.*, 41(1):45–52.

[98] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, e Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, dezembro de 2008.

[99] Nedeljko Vasić, Dejan Novaković, Svetozar Miučin, Dejan Kostić, e Ricardo Bianchini. Dejavu: accelerating resource allocation in virtualized environments. *SIGARCH Comput. Archit. News*, 40(1):423–436, março de 2012.

[100] Christian Vecchiola, Suraj Pandey, e Rajkumar Buyya. High-performance cloud computing: A view of scientific applications. *Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, ISPAN '09, páginas 4–16, Washington, DC, USA, 2009. IEEE Computer Society.

[101] Smita Vijayakumar, Qian Zhu, e Gagan Agrawal. Dynamic resource provisioning for data streaming applications in a cloud environment. *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, páginas 441–448, Washington, DC, USA, 2010. IEEE Computer Society.

[102] Lizhe Wang, Jie Tao, Marcel Kunze, Alvaro Canales Castellanos, David Kramer, e Wolfgang Karl. Scientific cloud computing: Early definition and experience. *Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, HPCC '08, páginas 825–830, Washington, DC, USA, 2008. IEEE Computer Society.

[103] Jared Wilkening, Wilke1 Andreas, Desai Narayan, e Meyer Folker. Using Clouds for Metagenomics: A Case Study.

[104] Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, e James Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, SC '07, páginas 38:1–38:12, New York, NY, USA, 2007. ACM.

[105] IT Word. Pane no cloud da amazon, 2011. [Online; accessed 11-Junho-2012].

[106] Katherine Yelick, Susan Coghlan, Brent Draney, Lavanya Ramakrishnan, Adam Scovel, Iwona Sakrejda, Anping Liu, Scott Campbell, Piotr T Zbiegiel, Tina Declerck, e et al. The magellan report on cloud computing for science the magellan report on cloud computing for science. *Energy*, páginas 170, 2011.

[107] Qi Zhang, Lu Cheng, e Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1:7–18, 2010. 10.1007/s13174-010-0007-6.