UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Guilherme Neri Andrade

Um Arcabouço Escalável, Eficiente e Adaptativo para Busca Aproximada em Conteúdo Multimídia

Belo Horizonte
2022

Guilherme Neri Andrade

Um Arcabouço Escalável, Eficiente e Adaptativo para Busca Aproximada em Conteúdo Multimídia

Versão Final

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.

Orientador: Renato Antônio Celso Ferreira
Coorientadores: George Luiz Medeiros Teodoro e
Leonardo Chaves Dutra da Rocha

Belo Horizonte
2022

Andrade, Guilherme Neri.

A554a Um arcabouço escalável, eficiente e adaptativo para busca aproximada em conteúdo multimídia: [recurso eletrônico] / Guilherme Neri Andrade – 2022.
1 recurso online (118 f. il, color.) : pdf.

Orientador: Renato Antônio Celso Ferreira.
Coorientadores: George Luiz Medeiros Teodoro e Leonardo Chaves Dutra da Rocha.

Tese (Doutorado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f. 113 -118

1. Computação – Teses. 2. Computação de Alto Desempenho – Teses. 3. Sistemas Distribuídos – Teses. 4. Recuperação da Informação - Teses. 5. Sistemas multimídia. I. Ferreira, Renato Antônio Celso. II. Teodoro, George Luiz Medeiros. III. Rocha, Leonardo Chaves Dutra da. IV. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. V. Título.

CDU 519.6*34(043)

Ficha catalográfica elaborada pela bibliotecária Irenquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais - ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇAO

FOLHA DE APROVAÇÃO

Um Arcabouço Escalável, Eficiente e Adaptativo para Busca Aproximada em Conteúdo Multimídia

GUILHERME NERI ANDRADE

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. RENATO ANTÔNIO CELSO FERREIRA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. LEONARDO CHAVES DUTRA DA ROCHA - Coorientador
Departamento de Ciência da Computação - UFSJ

PROF. GEORGE LUIZ MEDEIROS TEODORO - Coorientador
Departamento de Ciência da Computação - UFMG

PROF. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - UFMG

PROFA. LÚCIA MARIA DE ASSUMPÇÃO DRUMMOND
Departamento de Ciência da Computação - UFF

Documento assinado digitalmente
gov.br
MARIO ANTONIO RIBEIRO DANTAS
Data: 23/04/2025 17:45:38-0300
Verifique em https://validar.iti.gov.br

Prof. Mario Antonio ribeiro Dantas
Departamento de Ciênncia da Computação – UFJF

Documento assinado digitalmente
gov.br
RICARDO RIBEIRO DOS SANTOS
Data: 23/04/2025 18:59:13-0300
Verifique em https://validar.iti.gov.br

Prof. Ricardo Ribeiro dos Santos
Faculdade de Computação – UFMS

Belo Horizonte, 18 de março de 2022.

Dedico aos meus pais, à Isa e à Ciência.

Agradecimentos

Meu processo de doutoramento foi longo. Foram sete anos em que estive matriculado na Universidade Federal de Minas Gerais como doutorando em Ciência da Computação. Nos anos iniciais, começando em 2015, cumpri o cronograma do ciclo de qualificação como esperado: passei nas temidas prova teóricas (Projeto e Análise de Algoritmos; Redes de Computadores e Sistemas Operacionais; Teoria da Computação; Arquitetura/Organização de computadores e Sistemas Lógicos), passei nas disciplinas obrigatórias, terminei o estágio em docência. Estive presente, como sempre, nos projetos de pesquisa relacionados ao contexto de Sistemas e Computação de Alto desempenho, já havia publicado artigos e tinha uma bolsa de estudos fornecida pela LG Eletrocnics. Estava a frente de um grupo de pesquisa que, em conjunto da LG e professores da UFMG, começamos e finalizamos um projeto de pesquisa relevante. Nesse ponto, início de 2017, ainda com 4 semestres de crédito, parecia estar confortável para finalizar o doutorado nos sonhados 4 anos de curso. Mas a ciência é desafiadora não só tecnicamente, ele te confronta em muitos aspectos, que vão muito além do profissional, única e exclusivamente porque a ciência é incerta.

Em 2017, comecei o caminho para encontrar minha tese. Me dediquei nos próximos dois anos a tornar todas as minhas contribuições acadêmicas anteriores em uma tese de doutorado. Eu claramente havia uma solução para um problema, mas qual era ele? Eu não consegui responder. Essa busca, na qual minha solução corria atrás de um problema, por um tris não me fez abandonar toda minha trajetória acadêmica. Não havia motivação. Na busca pelo problema eu perdi, também, a solução. O sentimento de não ter encontrado a tese é bem ruim. A ciência me mostrou, nesse ponto, que não tem como planejar a incerteza. Me afastei do doutorado, não havia energia, nem vontade. Novamente me aproximei em 2020. Já havia perdido muito dos sentimentos que me aproximavam da vida acadêmica, e tive que tirar algumas forças para recomeçar. Recomecei.

O recomeço coincidiu com um momento mundial delicado: a pandemia. Iniciei em um outra jornada acadêmica, bem mais intensa. Com uma dedicação que eu não via em mim desde alguns anos atrás: codifiquei com muita vontade, propus e testei coisas novas, formei um novo grupo de pesquisa, perdi fins de semana, tive gastrite, tive ansiedade, mas ao final de 2021 eu tive uma tese. E essa tese representava pra mim muito mais que contribuições científicas: era o resultado de uma teimosia (quase que masoquista) em mostrar pra mim mesmo que não tinha como controlar tudo e que eu deveria sempre respeitar meus momentos. Em 2022 eu defendi a tese.

Esse é um resumo bem rápido do que foram esses 7 anos. Porém é muito mais

denso e profundo do que consegui escrever nesses 3 parágrafos. Quando me vi tendo de recomeçar algo que eu sentia já ter perdido e não fazer mais sentido para a minha vida, eu enfrentei labirintos psicológicos que não consigo descrever. Minha ansiedade era cotidiana, a inquietação e impaciência também. E voltando a um comentário do segundo parágrafo: "e tive que tirar algumas forças para recomeçar...", eu sei exatamente de onde vieram essas forças.

O meu recomeço acadêmico foi seguido de um grande e importante começo. Com a Isa dividi tudo: rotina, pandemia, trabalhos, diversão, medos, euforias, descanso, sonhos. Percebo que com os dias compartilhados nos conectamos de uma forma muito profunda, senti dela toda a motivação, incentivo e inspiração para continuar meu trabalho acadêmico. Ela, mais que ninguém, presenciou minha ansiedade, sentiu comigo, esteve presente em todos os momentos que desanimei e me ajudou a reerguer explicitamente com palavras de amor e acolhimento, mas principalmente silenciosamente quando demostrava admiração pelo meu trabalho e pelo meu esforço. Essa admiração que a Isa expressa por mim, foi meu combustível cotidiano. As vezes quando eu não conseguia perceber o quão longe eu tinha chegado, ou o quão importante foram as coisas que eu havia feito, ela mais que prontamente se enchia de orgulho para me mostrar tudo isso. É uma inspiração e privilégio ter a Isa como o amor da minha vida e a conclusão dessa tese é um resultado do bem que ela me faz.

Meus pais são a força que me sustenta em todos os meus caminhos, não foi diferente no doutorado. Durante meu recomeço estiveram distantes fisicamente, entretanto, presentes em cada situação. Deles tiro a referência de caráter, honestidade, e trabalho duro, me inspiram como pessoa e me incentivam em qualquer que seja a minha decisão. A grande herança que me deram foi a clareza em entender que com o estudo eu poderia alcançar o mundo e, para isso, nunca mediram esforços em me ajudar na minha formação, mesmo tendo que fazer sacrificios. Pai e Mãe, vocês são um grande motivo de eu ter seguido em frente, de não ter desanimado e principalmente de ter muito alegria em compartilhar o orgulho que essa vitória representa. Vocês me acompanharam sempre no meu caminho e esse título é nosso!

Todos os meus familiares tiveram grande importância também e agradeço todos que comemoraram essa conquista comigo. Aos meus avós Vô Lan (em memória) e Vó Melena, Vô Felipe (em memória) e Vó Nenem (em memória), por serem a base, por serem suporte e referência. Aos meus tios e tias, por acreditarem na minha capacidade e expressarem tanto carinho. Aos meus primos e primas por serem irmãos e grandes amigos. Agradeço aos meus amigos que mesmo distantes compartilharam comigo a trajetória do meu doutorado e vibraram com minha vitória. E também, aos amigos do grupo de pesquisa da UFMG, que me ajudaram na construção da tese, com comentários e trabalho em conjunto.

Agradeço ao meu co-orientador George Teodoro que, muito gentilmente, me apresentou o caminho acadêmico em meu recomeço e me guiou ao longo desse novo contexto

de pesquisa. Agradeço também ao meu co-orientador Leonardo Rocha, que acompanhou os meus passos como cientista da computação desde a graduação até o fim do doutorado, com ele aprendi grande parte do que sei sobre a pesquisa científica. Por fim, não menos importante, ao meu orientador Renato Ferreira, que ao longo de 9 anos me orientou com muita serenidade e competência. Muito obrigado por ter me conduzido academicamente e por me ajudado a abrir importantes caminhos profissionais.

"*Os benefícios da ciência não são para os cientistas, e sim para a humanidade.*"
(Louis Pasteur)

Resumo

Aplicações Content-based multimedia retrieval (CBMR) vem se tornando cada vez mais populares em diversos serviços *online* que manipulam grandes volumes de dados e estão sujeitos à intensos fluxos de consultas e inserções de novos objetos (descritores multimídia). Embora sejam aplicações complexas, encontrar os objetos mais próximos à uma consulta é tipicamente a sua operação mais onerosa. Para resolver esse problema, diversos trabalhos recentes propõe a paralelização em memória distribuída da busca aproximada por vizinhos mais próximos (ANNS). Observou-se que as soluções destaque presentes na literatura empregam estratégias de paralelização que conduzem à uma distribuição do conjunto de dados propícia à criar gargalos de processamento e também mostram-se subótimas ao uso de recursos computacionais, presentes em cada nó de processamento, para tratar os fluxos intensos e variáveis de consultas e inserções de novos descritores. Assim, neste trabalho de tese foram propostas estratégias que superam as limitações apontadas e estendem a capacidade da paralelização de abordagens ANNS em memória distribuída para lidar eficientemente com demandas variáveis de consultas em grandes volumes crescentes de objetos multimídia.

Palavras-chave: computação de alto desempenho; sistemas; sistemas distribuídos; busca por similaridade.

Abstract

Content-based multimedia retrieval (CBMR) applications are becoming increasingly popular in several *online* services which handle large volumes of data and are submitted to high query and objects insertions (multimedia descriptors) rate. Although the complexity of these applications is in different aspects, find the objects closest to a query one, is undoubtedly one of the most costly operations. To solve this problem, several recent work proposes the parallelization in distributed memory of Approximate nearest neighbors (ANN) search. It was observed that the highlighted solutions present in the literature employ parallelization strategies that lead to a dataset partitioning creating processing bottlenecks and also show up suboptimal to the use of computational resources, present in each processing node, to handle the high variables queries and insertions streans. Thus, in this thesis work, strategies were proposed that overcome the limitations pointed out and extend the ability of ANNS distributed memory parallelization approaches to efficiently handle intense and variable query demands in large and growing volumes of multimedia objects.

Keywords: high performance computing; systems; distributed systems; similarity search.

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

CBMR	*Content-Based Multimedia Retrieval*
QP	*Query Processor*: Processador de Consultas
Coord	Coordenador
QS	*Query Streamer*: Produtor de Consultas
IS	*Insertion Streamer*: Produtor de novos descritores
DES	*Data Equal Split*
BES	*Bucket Equal Split*
SABES	*Spatial-Aware Bucket Equal Split*
IVFADC	*Inverted File Index for Asymmetric Distance Calculation*
IVF	I*nverted File Index*
ANNS	*Approximated Nearest Neighbors Search*
KNN	*k-Nearest Neighbors*
QOT	*Query Outer Threads*: *Threads* externas para busca local
QIT	*Query Inner Thread*: *Threads* internas para busca local
IOT	*Insertion Outer Threads*: Threads externas para inserção de novos descritores
QSR	*Query Stream Rate*: Taxa de chegada de Consultas
ISR	*Insertion Stream Rate*: Taxa de chegada de novos descritores
QLF	*Query Load Factor*: Taxa de chegada de consultas
ILF	*Insertion Load Factor*: Taxa de chegada de novos descritores
IQ	*Insertion Queue*: Fila de tarefas para inserção de novos descritores
QQ	*Query Queue*: Fila de tarefas para consultas

Sumário

Capítulo 1

Introdução

Com o avanço das tecnologias recentes, várias aplicações ganharam destaque nos últimos anos, permitindo a produção e o consumo em larga escala de diversos tipos de conteúdo multimídia. Um exemplo significativo desse fenômeno são as redes sociais, onde é possível compartilhar textos, imagens e vídeos, interagir com pessoas e conteúdos variados. Isso tem possibilitado que sistemas de software extraiam e compreendam padrões não triviais de comportamento e preferências de milhares de usuários. Além disso, a Internet das Coisas desempenha um papel relevante nesse contexto. A interconexão de objetos do dia a dia, equipados com sensores como câmeras e microfones, tem permitido a coleta e transmissão de dados, dando origem a serviços autônomos e transparentes em diversas esferas da sociedade.

Consequentemente, a geração massiva de dados multimídia e a demanda por processamentos cada vez mais eficientes em diversos campos da tecnologia e da ciência da computação têm apresentado desafios significativos. Isso tem impulsionado o desenvolvimento e aprimoramento de aplicações que se baseiam na manipulação desse tipo de informação, consolidando áreas de interesse como Recuperação de Informação Baseada em Conteúdo Multimídia (CBMR) [2]. Dentro desse conjunto de aplicações, que abrange motores de busca de imagens, marcação de fotos em redes sociais, identificação de sons em tempo real, entre outras, há uma série de etapas que envolvem a coleta, transformação e extração de informações em volumes de dados cada vez maiores. E entre essas manipulações de objetos multimídia, um dos passos mais críticos e desafiadores consiste em encontrar objetos semelhantes a uma consulta específica.

A Busca por Similaridade (NNS)[62] é uma operação fundamental não apenas em aplicações CBMR, mas também em diversos cenários desafiadores na ciência da computação. Nesse problema de busca, o objetivo é encontrar o objeto mais semelhante a um objeto de consulta específico, onde a similaridade entre eles é determinada pela proximidade no espaço métrico no qual estão inseridos. Uma generalização direta desse problema, que é mais adequada para aplicações CBMR, é encontrar um conjunto de k objetos mais próximos (conhecido como o problema dos k-vizinhos mais próximos, ou k-NN)[62, 42].

A solução exata para o problema k-NN, chamada de busca exaustiva, é impraticável nos cenários atuais das aplicações CBMR. Portanto, uma série de soluções foram

desenvolvidas para enfrentar os desafios da busca em grandes volumes de dados de alta dimensão. A Busca Aproximada de Vizinhos Mais Próximos (ANNS) é baseada na ideia de que é possível alcançar uma precisão próxima do resultado ótimo com um desempenho computacionalmente viável. Além disso, ela oferece a capacidade de ajustar o equilíbrio entre a precisão desejada e a eficiência, permitindo encontrar um compromisso adequado entre ambos. Embora as estratégias ANNS tenham obtido resultados notáveis ao possibilitar a busca em volumes relevantes de dados de alta dimensão, essas soluções foram projetadas para execução sequencial em uma única máquina, com o conjunto de dados alocado na memória principal.

No entanto, as aplicações de busca multimídia em cenários CBMR reais lidam com volumes consideráveis de dados e fluxos intensos de consultas e inserções de novos objetos na base de busca. Essas características ultrapassam a capacidade de uma única máquina para armazenar e executar o processo de busca de forma eficiente. Diante disso, surge a necessidade de desenvolver soluções paralelas e em memória distribuída para abordar esses desafios.

1.1 Objetivos

As soluções em memória distribuída para ANNS devem ser capazes de lidar com as complexas características inerentes aos cenários online de aplicações CBMR, visando acelerar o processo de busca por objetos similares a uma consulta em um conjunto de dados dinâmico. Além de abordar questões relacionadas à escalabilidade da solução paralela, as abordagens de alto desempenho para esse problema precisam enfrentar o desafio de lidar com bases de dados multimídia que estão sendo consultadas intensivamente e que, simultaneamente, têm um fluxo intenso de novos objetos sendo produzidos e precisando ser indexados para expandir o conjunto de busca disponível. Esses fluxos de dados concorrentes possuem intensidades variáveis ao longo da execução da aplicação, alternando entre momentos de alta intensidade de consultas e momentos de alta intensidade de inserções, por exemplo.

A maneira como o conjunto de dados é distribuído entre os nós de processamento da máquina de memória distribuída é uma característica-chave para a eficiência da solução ANNS. Uma consulta precisa percorrer todos os dados relevantes para encontrar os objetos mais similares, o que significa que a estratégia de particionamento da base de dados determina quantos e quais nós de processamento precisam ser acessados durante uma busca. Muitas abordagens presentes na literatura [48, 6, 38, 34, 39, 51, 3] exploram uma divisão igualitária dos dados. Embora essa abordagem seja popular e amplamente

utilizada, ela enfrenta gargalos significativos de comunicação e custos computacionais devido ao início e término dos processos no sistema distribuído. Isso ocorre porque, para cada consulta, todos os nós de processamento precisam ser acessados. Essa limitação impacta a escalabilidade e o desempenho paralelo da estratégia distribuída de ANNS.

Considerando que objetos multimídia similares estão próximos no espaço, existe a oportunidade de explorar particionamentos da base de dados que direcionem o acesso a um conjunto reduzido de nós de processamento durante uma consulta. Ao utilizar um subconjunto específico de recursos para uma busca, é possível aumentar as possibilidades de execuções paralelas de consultas e também reduzir os custos associados à comunicação e execução da busca nos nós de processamento. Surpreendentemente, essa abordagem tem sido pouco explorada na literatura.

A estratégia de particionamento dos dados direciona o acesso e a comunicação entre os nós de processamento. No entanto, cada nó precisa gerenciar fluxos variáveis de consultas e novos objetos multimídia que precisam ser armazenados. Lidar eficientemente com a simultaneidade entre consultas e inserções é um desafio significativo. Além disso, a variabilidade na intensidade dos fluxos torna ainda mais complexa a proposição de uma solução paralela e em memória distribuída para ANNS em aplicações CBMR.

Nesse contexto e considerando que os nós de processamento são componentes com múltiplos recursos paralelos, a adaptação dinâmica desse paralelismo interno se torna uma abordagem atrativa para se ajustar às diferentes intensidades dos fluxos de entrada. Alguns trabalhos exploram o ajuste dinâmico dos níveis de paralelismo para acelerar o processo de busca aproximada quando há apenas fluxo de consultas [51, 4]. Além disso, estudos em diversos contextos de aplicações [9, 17, 54] demonstram que a adaptação dinâmica de recursos computacionais em cenários com variações na intensidade dos fluxos de dados pode significativamente aprimorar o desempenho dos sistemas. Isso acontece porque configurações estáticas tendem a ser subótimas para alocar recursos de maneira eficaz. Portanto, para que uma solução ANNS em memória distribuída, no contexto de aplicações CBMR, seja bem-sucedida em lidar com múltiplos fluxos de dados dinâmicos, é fundamental se adaptar às diferentes intensidades da aplicação.

Dessa forma, tendo em mente os desafios mencionados para o tema de *busca aproximada por objetos multimídia similares em grandes volumes de dados dinâmicos, por meio de estratégias que organizam espacialmente os itens e tiram proveito de máquinas com memória distribuída, considerando fluxos variáveis de consultas e novos dados como entrada*, os objetivos desta tese buscam propor e discutir soluções para responder à pergunta: *O particionamento do conjunto de objetos multimídia correlacionando as características espaciais dos dados e a adaptação dinâmica de recursos paralelos em máquinas com memória distribuída melhora a eficiência computacional para a execução de ANNS em volumes massivos de dados?*.

Para explorar as hipóteses em evidência, as premissas relevantes deste trabalho

serão:

1. Abordagens para ANNS que se baseiam na organização dos objetos de acordo com a proximidade espacial demonstram uma dinâmica de armazenamento e busca semelhantes. Isso abre a possibilidade de conceber e desenvolver uma arquitetura paralela e em memória distribuída que seja independente da estratégia específica de ANNS a ser implementada. Essa abordagem agnóstica à estratégia ANNS permitiria uma implementação mais flexível e adaptável, capaz de acomodar diferentes abordagens de busca aproximada por objetos similares, ao mesmo tempo em que otimiza a eficiência de armazenamento e consulta de descritores.

2. A distribuição do conjunto de dados entre os nós de processamento, levando em consideração as características de proximidade espacial dos objetos, tem o potencial de diminuir a quantidade de recursos computacionais necessários em cada processo de busca. Essa diminuição pode amplificar a utilização do paralelismo disponível na máquina de memória distribuída, resultando em um aumento significativo no desempenho da solução ANNS.

3. Configurações estáticas de paralelismo, aplicadas internamente a cada nó de processamento para o processamento de consultas e inserções, podem ser subótimas devido à variabilidade na intensidade dos fluxos de entrada. Cada nó de processamento precisa ter a capacidade de ajustar dinamicamente seus recursos de processamento internos para encontrar a configuração adequada que corresponda à intensidade dos fluxos de consultas e inserções em um dado momento. Isso é essencial para otimizar a utilização dos recursos disponíveis e melhorar o desempenho do sistema em tempo real, conforme as condições de entrada variam.

1.2 Contribuições

De maneira abrangente, o modelo de paralelização em memória distribuída para ANNS é independente da estratégia específica de ANN usada localmente nos nós de processamento para buscar objetos similares nas partições distribuídas. Nos trabalhos relacionados, é possível encontrar avaliações utilizando abordagens específicas para busca aproximada, como IVFADC [51], LSH [49] e FLANN [39], que são algumas das estratégias ANNS mais comuns.

Esses algoritmos de ANNS de última geração frequentemente se baseiam na indexação dos objetos multimídia em diversas estruturas de dados, que vão desde Índices

Invertidos [51] até Hashes [49]. Apesar das diferentes estruturas de dados, a intuição por trás da organização dos dados é semelhante: objetos próximos no espaço são agrupados em conjuntos, formando "buckets"de dados. Essa estratégia visa reduzir a quantidade de comparações necessárias para encontrar objetos similares a uma consulta, já que apenas "buckets"mais próximos serão acessados. Dessa forma, a primeira contribuição deste trabalho de tese é a proposta e implementação de uma arquitetura paralela e em memória distribuída, que inclui componentes capazes de dar suporte ao desenvolvimento e à implementação de algoritmos ANNS que organizam o conjunto de objetos em "buckets". Além disso, essa arquitetura genérica possibilita simplificar a implementação de diferentes abordagens para o particionamento de dados, personalizando métodos para o roteamento de dados, além de permitir a execução simultânea de fluxos de consultas e inserções, bem como a implementação de mecanismos para a adaptação dinâmica dos recursos computacionais necessários para processar esses fluxos de entrada.

Além disso, considerando a arquitetura em memória distribuída proposta para ANNS, este trabalho de tese explora e desenvolve um conjunto de estratégias para o particionamento de dados e realiza comparações entre elas e as abordagens existentes encontradas na literatura. As abordagens propostas, conhecidas como SABES e SABES++, particionam o conjunto de dados com base em suas características espaciais, visando a redução da quantidade de recursos computacionais necessários para executar uma consulta. Além disso, são investigados métodos para manter o equilíbrio da carga de objetos em cada nó de processamento. Essas contribuições são combinadas com uma análise abrangente e comparação de várias estratégias de particionamento de dados disponíveis na literatura, discutindo os comportamentos de eficiência associados a cada uma.

Para lidar com os requisitos de tempo real das aplicações CBMR, onde a base de dados cresce ao longo do ciclo de vida da aplicação, foi proposta uma organização do conjunto de objetos em cada nó de processamento, considerando não apenas a proximidade espacial, mas também a proximidade temporal do momento em que foram indexados. Essa proposta amplia a capacidade dos "buckets"utilizados pelas abordagens ANNS, permitindo que consultas e inserções ocorram em partições de tempo e espaço simultaneamente, com baixo custo computacional para a sincronização dessas operações.

E, ainda relacionado aos aspectos dinâmicos da busca aproximada em aplicações CBMR, outra contribuição significativa deste trabalho é a proposta de uma abordagem adaptativa para a gestão dinâmica de recursos paralelos em um nó de processamento, denominada MS-ADAPT. Nessa contribuição, foram desenvolvidos métodos para identificar a variabilidade da intensidade dos fluxos de entrada, de modo a determinar a melhor configuração de níveis de paralelismo interno para cada nó de processamento. Essa abordagem foi extensivamente comparada com estratégias de configurações estáticas e o impacto no ganho de eficiência foi avaliado. Os resultados demonstraram que o MS-ADAPT atinge ou supera o desempenho das melhores estratégias estáticas em diferentes cenários de variação

de intensidade dos fluxos.

Todas as contribuições foram avaliadas por meio de testes de escalabilidade nos quais diferentes estratégias de particionamento de dados e métodos de gestão de fluxos concorrentes foram variados. Nos cenários de teste, foram replicados comportamentos semelhantes aos das aplicações CBMR, e os experimentos foram conduzidos utilizando até 160 nós de computação e um conjunto de dados contendo 344 bilhões de objetos multimídia. Os resultados revelaram que as soluções propostas exibem eficiência superlinear, ou seja, à medida que o número de nós utilizados aumenta, as melhorias de desempenho também aumentam.

Por fim, as contribuições deste trabalho de tese foram concebidas para cumprir o objetivo de pesquisa: propor uma solução paralela e em memória distribuída para abordagens ANNS que enfrentam desafios de grande volume de dados altamente dimensionais e dinâmicos. Isso é alcançado por meio da gestão eficiente de fluxos concorrentes e paralelos, da adaptação dinâmica dos recursos computacionais disponíveis, além da organização do conjunto de dados com base em características espaciais e temporais, a fim de otimizar o desempenho paralelo do sistema como um todo.

1.3 Organização do Texto

No Capítulo 2 é responsável por apresentar o contexto dos problemas abordados por este trabalho de tese. Em um primeiro momento, é apresentado as definições e principais soluções referentes aos problema de busca aproximada em grandes volumes de dados. São detalhados os mecanismos e objetivos das estratégias estado da arte para ANNS, bem como aponta-se as limitações e desafios em aberto. Em sequencia, o capítulo discute as abordagens de alto desempenho para ANNS presentes na literatura, definições importantes e os compromissos a serem otimizados, separando as abordagens e discussões em dois grupos: i) ANNS em Conjunto de dados Estáticos; ii) ANNS em Conjunto de dados Dinâmicos.

No Capítulo 3 endereça a primeira contribuição deste trabalho de tese. Nele, inicialmente, apresenta-se detalhes da arquitetura em memória distribuída proposta para suportar as funcionalidades que serão exploras em sequencia. A arquitetura apresentada permite a implantação de diferentes estratégias ANNS baseadas na organização de dados em *buckets*, bem como a implementação de diferentes estratégias de particionamento de dados e a gestão adaptativa de fluxos de dados concorrentes. Assim, em sequência no Capítulo 3, apresenta-se as estratégias de particionamento de dados SABES e SABES++ que são propostas deste trabalho. Por fim, essas estratégias foram amplamente avalia-

das, comparando-as com estratégias linha de base presentes na literatura bem como são apresentados testes de escalabilidade da solução paralela e em memória distribuída.

No Capítulo 4 são apresentados os detalhes da arquitetura proposta que suportam a característica dinâmica das aplicações CBMR, onde fluxos de consultas são executados de forma concorrente à fluxos de inserção de novos objetos na base de dados. Nesse momento, todos os componentes e mecanismos propostos que expandem a capacidade da arquitetura paralela e em memória distribuída são detalhados, bem como a proposta de estratégia (MS-ADAPT) para adaptação dinâmica dos recursos paralelos em cada nó de processamento do sistema. Essa abordagem foi avaliada comparando-se com soluções estáticas e novamente testes de escalabilidade foram conduzidos para observar o impacto de eficiência resultado das contribuições.

Por fim, o Capítulo 5 conclui o trabalho de tese, apresentando as observações e resultados dos estudos conduzidos bem como as possibilidades e oportunidades de pesquisa futura.

Capítulo 2

Revisão sobre Busca por Objetos Similares

O propósito deste capítulo é contextualizar os problemas de busca que este trabalho aborda, oferecendo definições, explorando complexidades, identificando desafios inerentes e destacando contribuições relevantes presentes na literatura. As formalizações apresentadas aqui estabelecem o embasamento teórico necessário para compreender as abordagens investigadas e propostas nos próximos capítulos.

Para começar, na Seção 2.1, aborda-se o amplo problema da Busca por Similaridade, delineando as nuances da derivação que é central neste estudo: a Busca pelos k-vizinhos mais próximos (Busca kNN). Em seguida, dada a inviabilidade de aplicar métodos exatos a este problema, a seção introduz as principais heurísticas aproximadas para a Busca kNN (ANNS) presentes na literatura. Aqui, são detalhados os mecanismos utilizados para acelerar a busca e também para reduzir a dimensionalidade do espaço métrico, onde a precisão do resultado da busca é trocada por ganhos de desempenho.

Contudo, as soluções propostas para ANNS são tipicamente desenvolvidas para execução sequencial em uma única máquina, o que limita sua aplicabilidade em cenários caracterizados por grandes volumes de objetos, como é o caso das aplicações CBMR, que também são abordadas neste estudo. Portanto, a Seção 2.3 explora soluções paralelas e distribuídas para viabilizar a execução de estratégias ANNS em conjuntos de dados volumosos com alta dimensionalidade. Esta seção inicia contextualizando esse cenário, delineando os objetivos a serem otimizados e as características que distinguem duas categorias de abordagens: i) ANNS em Conjuntos de Dados Estáticos; ii) ANNS em Conjuntos de Dados Dinâmicos. Cada uma dessas categorias é examinada separadamente nas Subseções 2.3.2 e 2.3.3, onde são apresentadas suas particularidades, componentes essenciais e principais contribuições da literatura.

O capítulo conclui apontando as oportunidades de pesquisa que emergem dessas análises e, por fim, destaca os aspectos que fundamentam as contribuições deste trabalho.

2.1 A Busca por Similaridade e Busca kNN Aproximada (ANNS)

Busca por similaridade é um termo amplo que descreve um conjunto de algoritmos que, em essência, tentam resolver o problema de encontrar objetos mais próximos à um objeto consulta em um conjunto finito de dados representado em um espaço métrico. Um espaço métrico é tal que:

Definição 1. *Seja $S = (M, m)$ um espaço métrico formado por $M \neq \emptyset$ e uma métrica $m : M \times M \rightarrow \mathbb{R}$ tal que $\forall (x, y) \in M \times M, m(x, y) \in R$ na qual podemos chamar de distância de x a y. Assim $m(x, y)$ deve satisfazer: (i) $m(x, x) = 0$; (ii) Se $x \neq y$ então $m(x, y) > 0$; (iii) $m(x, y) = m(y, x)$; (iv) $m(x, z) \leq m(x, y) + m(y, z)$*

O problema da busca por similaridade pode ser definido formalmente como:

Definição 2. *Seja $S = (\Omega, dist)$ um espaço métrico d-dimensional, com domínio Ω tal que $dist : \Omega \times \Omega \rightarrow \mathbb{R}^d$, e $X \subseteq \Omega$ um conjunto de objetos nesse espaço, queremos encontrar o(s) objeto(s) mais próximo, de acordo com dist, à um objeto de consulta $q \in \Omega$. Um objeto p no espaço d-dimensional é tal que $p = p_1, p_2, ..., p_i, ..., p_d$.*

Nesse contexto, a similaridade entre um par de objetos é dada pela distância espacial entre eles, de forma que valores pequenos de distância correspondem à um alto grau de similaridade. Assim, é possível derivar essa similaridade por meio de diferentes métricas de distância, escolhendo aquela que mais se adéqua ao contexto do conjunto de objetos. Distância Euclidiana (L_2) [14], Manhattan (L_1) [46], Similaridade de Cossenos [59] e Chebyshev [13], são as métricas mais comuns. Assim, podemos completar a definição anterior com:

Definição 3. *Dado um conjunto de objetos $X = p_1, p_2, ...p_n$ no espaço métrico S e um objeto consulta q encontrar o(s) objeto(s) mais próximos $NN(q, X)$ respeitando a métrica de distância $dist : S \times S \rightarrow \mathbb{R}$ é tal que: $dist(q, X) = argmin_{p \in X} dist(q, p)$.*

O problema da Busca por Similaridade pode ser enunciada diferentemente variando o objetivo da consulta: i) *Consulta por Alcance*: o objetivo é encontrar todos os objetos em X tal que a similaridade para q não ultrapasse um valor definido; ii) *Consulta pelos k-vizinhos mais próximos (kNN)*: o objetivo é encontrar um conjunto de k objetos em X mais próximos à q.

A Busca pelos k-vizinhos mais próximos (Busca kNN) representa a maneira mais amplamente usada de consulta por similaridade [62, 42], dado que é possível controlar com facilidade a cardinalidade do resultado esperado. Com o advento da geração massiva

de dados, esse tipo de busca ganhou importância, tornando-se fundamental em diversas aplicações modernas onde o foco é a manipulação de grandes coleções de objetos, como em aplicações CBMR. Pela definição 2 podemos formalizando o problema da Busca kNN como:

Definição 4. *Seja S um espaço d-dimensional, com domínio ω tal que $d : \omega \times \omega \to \mathbb{R}^d$, e $X \subseteq \omega$ um conjunto de objetos nesse espaço, queremos encontrar os k objetos mais próximos, em d, à um objeto de consulta $q \in \omega$. Assim, $KNN(q, X, k) = K$, onde o conjunto K deva satisfazer as condições: $|K| = k, K \subseteq X$ e, indo além, $\forall p \in K, x \in (X - K), dist(q, p) \leq dist(q, x)$. [43]*

A solução exata para o problema da Busca kNN, conhecida como busca exaustiva, e apresentada no Algoritmo 1, demanda que o objeto de consulta seja comparado com todos os objeto do conjunto X. Essa abordagem tem uma complexidade de tempo de execução próxima a $O(k|X| + klog(k))$.

Algorithm 1: Busca Exaustiva kNN

```
ß Data: Conjunto X e objeto consulta q
  Result: Conjunto K com k objetos mais próximos
K ← ∅
foreach p ∈ X do
    distances.append(dist(p, q));
sort(distances, modo = crescente)
K ← distances[0..k]
```

Apesar de proporcionar uma solução exata ao problema, e mesmo considerando que várias estruturas de dados foram proposta para acelerar a resolução de consultas [20, 8, 39], buscar exaustivamente os k vizinhos mais próximos é impraticável em grandes volumes de objetos e espaço de altas dimensões [28, 57].

Portanto, para contornar essa inviabilidade, as heurísticas de busca aproximada podem promover a troca de desempenho pela assertividade do conjunto dos k-vizinhos mais próximos encontrado. Nesse conjunto de soluções, conhecidos por Busca Aproximada dos k-vizinhos mais próximos (ANNS), existe a possibilidade de reduzir o tempo de busca em troca de uma perda na qualidade da resposta (quando comparada com a solução exata).

O custo computacional de uma Busca kNN, é essencialmente dado pela i) quantidade de distâncias calculadas entre pares de objetos e; ii) pelo tempo gasto em cada um desses cálculos. O objetivo geral da Busca kNN Aproximada é, reduzir a quantidade de distâncias computadas para acelerar o tempo total de pesquisa. Entretanto, esse conjunto de soluções trás a troca entre os compromissos de eficiência e eficácia do resultado da busca: ao passo que, quanto menos objetos forem comparados ao objeto consulta, mais rápida será a busca e consequentemente mais distantes estaremos da solução exata.

Conforme argumentado no trabalho [42], existem pontos favoráveis às heurísticas de ANNS que as tornam aceitáveis e amplamente usadas: Há uma diferença entre a percepção do usuário de qual seria o objeto mais próximo à uma consulta; e efetivamente o objeto mais próximo segundo a métrica de distância espacial escolhida. Nesse sentido, é preferível receber uma resposta não tão precisa do ponto de vista métrico, entretanto em um tempo consideravelmente menor. Em casos práticos, o tempo de resposta se torna mais importante e essencial do que a precisão do resultado.

Sendo assim, existe uma grande variedade de abordagens para ANNS na literatura, e segundo a taxonomia apresentada no trabalho [42], elas podem ser classificadas segundo diferentes aspectos como: i) o tipo de dados onde se aplicam; ii) a estratégia de aproximação envolvida para acelerar a busca; iii) a garantia da qualidade da resposta e; iv) o tipo de interação do usuário.

Conforme supracitado, a quantidade de distâncias calculadas entre os pares de objetos e os custos computacionais associados, balanceiam os compromissos de tempo total de busca e qualidade da resposta. Assim, a estratégia de aproximação escolhida (aspecto ii) guiará como a abordagem ANNS irá acelerar a busca, podendo ser por meio da:

- Mudança e aproximação do espaço métrico e da forma como as distâncias são computadas [55]. São soluções menos comuns, onde o objetivo é aplicar a busca exaustiva em um contexto espacial mais simples, onde o custo computacional do cálculo das distância é menor.

- Redução da quantidade de cálculos de distância. Nesse contexto, é possível encontrar soluções que podam o conjunto de objetos a serem comparados [5]; ou que finalizam a busca antecipadamente quando algum valor aceitável foi alcançado [63].

As estratégias mais relevantes para ANN presentes na literatura [24, 39, 30], se beneficiam da poda do conjunto de objetos à serem comparados com o objeto consulta. Assim, considerando a relevância e aplicabilidade dessas soluções, nas subseções seguintes exploraremos os aspectos desse conjunto de abordagens para Busca Aproximada dos k-vizinhos mais próximos.

2.1.1 Poda do Conjunto de Busca

De maneira geral, e partindo da definição 2, a similaridade entre os objetos do conjunto X é dada pela distância espacial entre eles, logo, objetos similares tendem a estar próximos uns aos outros no espaço métrico S. Com essa observação, podemos dizer

que no conjunto de dados $X \subseteq S$, existem regiões espaciais contendo um conjunto de objetos similares. Tais regiões, podem ser formalmente definidas como:

Definição 5. *Regiões* $X_n \subseteq X$ *são tais que* $(X_1 \cup X_2 \cup ... \cup X_n) = X | n \in 1..|X|$ *sendo que* $\forall x, y \in X_n ev \in (X - X_n), dist(x, y) \leq dist(x, v)$.

Dessa maneira, a estratégia de podar o conjunto de busca acontece exatamente quando a solução ANNS restringe a comparação do objeto consulta à subconjuntos de objetos que estão em algumas regiões, mais especificamente, as regiões mais próximas do objeto consulta. A escolha de todas as regiões para a busca resume a solução do problema ao método exaustivo.

A escolha da forma como essas regiões vão ser construídas e como os objetos da base vão ser armazenados é fundamental e difere entre as estratégias ANNS. Muitos trabalhos propõem o uso de estruturas de dados para consolidar essa organização e tirar proveito das facilidades de acesso aos subconjuntos espaciais. Dentre essas estruturas estão kd-tree, k-means tree, ball trees, e cover tree [20, 8, 39]. Por meio desses mecanismos, as soluções ANNS já apresentam considerável aceleração no tempo de busca quando comparadas com a solução exaustiva.

Indo além, em alguns outros trabalhos, as estratégias de ANNs se beneficiam de técnicas de agrupamento dos objetos (como o algoritmo k-means [45]) para encontrar as regiões e elencar objetos representativos em cada uma delas. A partir de então, é proposto o uso de estruturas de indexação, como listas invertidas [30], onde os objetos de uma região serão armazenados e associados ao respectivo objeto representativo. Outras proposta mapeiam as regiões em entradas de Tabelas Hashs [24], onde objetos similares são agrupados devido a colisão entre as suas respectivas representações Hash.

Definição 6. *Objetos similares armazenados em conjunto nas diferentes estruturas de dados consolidam Data Buckets) (a.k.a Buckets). Como por exemplo: objetos alocados no mesmo índice do sistema de arquivo invertido; objetos associados na mesma entrada de uma tabela Hash; ou objetos na mesma secção de uma kd-tree.*

Mesmo tirando proveito de formas para organizar a base de dados espacialmente e assim acelerar o tempo de execução, a Busca kNN tende a performar imprecisamente e ineficientemente ao passo que a dimensionalidade do espaço métrico aumenta. Esse fenômeno é conhecido como *"mal da dimensionalidade"* [56, 27].

2.1.2 *Mal da Dimensionalidade*

Como discutido, a organização espacial para podar a quantidade de objetos comparados na busca, depende necessariamente da detecção de áreas onde há objetos semelhantes. Em espaços métricos de alta dimensionalidade, no entanto, todos os objetos parecem ser esparsos e diferentes de muitas maneiras, o que impede que estratégias comuns de organização de dados sejam eficientes ao organizá-los espacialmente. Esse problema, conhecido como *"mal da dimensionalidade"* também enuncia que as distâncias entre um objeto consulta com alta dimensão e o objeto mais distante da base de dados, bem como o descritor mais próximo são praticamente similares, alcançando uma relação tal qual:

$$\lim_{d\to\infty} E\left(\frac{\text{dist}_{\max}(d) - \text{dist}_{\min}(d)}{\text{dist}_{\min}(d)}\right) \to 0$$

Por esses motivos, o problema k-NN se torna ainda mais desafiador no contexto de aplicações que lidam com espaços de alta dimensionalidade, onde a performance cai ao passo que o espaço d-dimensional aumenta.

Dessa forma, as soluções para ANNS endereçam não só formas de podar o espaço de buscar, mas como também maneiras para reduzir a dimensionalidade do espaço, preservando características espaciais dos objetos. Nesse sentido, compactar as dimensões dos objetos trás benefícios ao minimizar os impactos do fenômeno *"mal da dimensionalidade"*, e também ao reduzir a complexidade de armazenamento do conjunto de objetos, possibilitando a Busca kNN Aproximada aconteça de forma eficiente em grandes volumes de dados de alta dimensionalidade.

Reduzir a dimensionalidade de um espaço vetorial preservando a característica espacial é um problema conhecido na matemática e também em teoria da computação [27]. O desafio é, a partir, de um espaço S d-dimensional, encontrar um mapeamento que construa um espaço aproximado S^* d^*-dimensional, onde $d > d^*$, de forma que para qualquer par de objetos em S a proporcionalidade da distância entre eles se preserve em S^*.

No contexto de ANNS, algumas formas são bastante populares e dentre elas podemos destacar Hashing [58, 15] e Product Quantization [30]. A estratégia de Hashing propõe representar os objetos de maneira compacta por meio de códigos Hash, aproximando a similaridade entre os pares de objetos pelo cálculo da distância Hamming [32]. Por sua vez, a abordagem de Product Quantization, propõe a representação dos objetos em uma dimensão inferir por meio da combinação de pontos representativos (codewords) das subdimensões. Ambas abordagens proporcionam a compactação da representação dos objetos e consequentemente agregam custo computacional inferior ao cálculo das similaridades.

2.2 Algoritmos para ANNS

Conhecendo os mecanismos discutidos anteriormente, as estratégias mais sofisticadas para ANN combinam diferentes abordagens para a poda do conjunto de objetos, e maneiras para compactar a representação do espaço métrico sem perder as características espaciais. Dentre as mais conhecidas destacam-se Locality Sensitive Hashing (LSH) [24], FLANN (fast library for approximate nearest neighbors) [39] e PQANNS [30]. As subseções seguintes estudam os detalhes de cada uma delas.

2.2.1 Locality Sensitive Hashing

A abordagem LSH é uma das mais competitivas técnicas para Busca Aproximada dos k-vizinhos mais próximos. Em essência, o objetivo é transformar a representação dos objetos em códigos Hash, onde objetos próximos no espaço são representados por códigos similares. Diferentemente do propósito tradicional de Hashing, a motivação é maximizar as colisões, para que assim, objetos similares tenham mesma representação hash e sejam associados à uma mesma entrada na Tabela Hash. A figura 2.1 ilustra essa motivação.

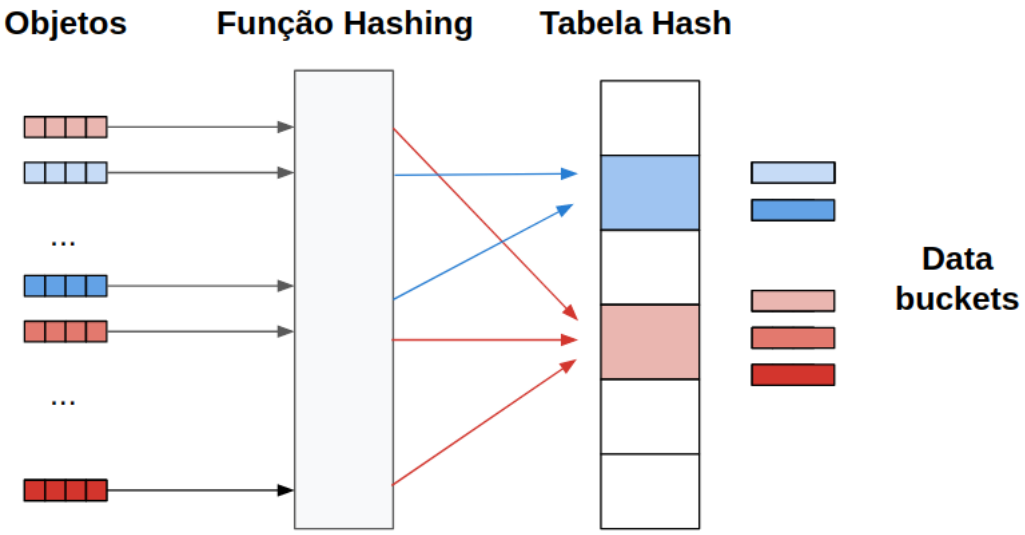


Figura 2.1: Objetos são mapeados por meio de uma função Hash. Aqueles similares vão ser representados por códigos Hash iguais e estarão associados à uma mesma entrada da Tabela Hash. Cada entrada na tabela representa um *Buckets*.

Sabemos que a quantidade de entradas na Tabela Hash é consideravelmente menor que a dimensão do espaço inicial. A função Hash empregada pela abordagem LSH, é uma

função $h : S \leftarrow HS$ que mapeia elementos do espaço S para *Buckets* $r \in HS$. Para tanto, essa função de mapeamento deve satisfazer as seguintes condições de colisão para qualquer par de objetos $x, y \in S$ escolhidos aleatoriamente:

- Se $dist(x, y) \leq R$, então $h(x) = h(y)$.
- Se $dist(x, y) \geq cR$, então $h(x) = h(y)$.

Considerando i) $R > 0$ um limite; ii) e c um fator de aproximação, escolhidos previamente. Indo além, a estratégia LSH tem dois principais parâmetros: o valor k, responsável por definir a cardinalidade do conjunto de vizinhos mais próximos; e a quantidade de Tabelas Hashs L.

No primeiro passo do algoritmo, é definido uma função Hash g, construída por meio da junção de k funções hash aleatórias h $(h_1, h_2, ..., h_k)$, tal que para qualquer objeto $p \in S$ $g(p) = [h_1(p), ..., h_k(p)]$. O algoritmo constrói L tabelas hash, sendo que cada uma delas corresponde à uma função aleatória g.

Durante a fase de pré-processamento, todos os objetos $n \in X$ da base de dados são mapeados para cada uma das L tabelas Hash, por meio das respectivas funções g. Por fim, na fase de consulta, dado um objeto q, o algoritmo itera em todas as tabelas L, e para cada uma, encontra o conjunto de objetos que foram mapeados no mesmo *Buckets* que q, por meio da colisão entre códigos Hash resultados das funções g. O processo finaliza quando um objeto com distância cR é encontrada, indicando que os objetos mais próximos a q já foram comparados, segundo o parâmetro de aproximação c.

A abordagem LSH garante a complexidade de pré-processamento $O(nLkt)$ onde t é o tempo gasto para a execução da função h, e a complexidade de busca de $O(L(kt+dnP_2^k))$.

2.2.2 FLANN

A biblioteca FLANN (Fast Library for ANN) [39] implementa diferentes estratégias de ANN e é capaz de escolher a mais eficiente para um determinado conjunto de dados. Em trabalhos anteriores, os autores avaliaram diferentes algoritmos para ANN [40, 41], e encontraram duas abordagens que alcançaram os melhores resultados: 1) *Randomized k-d Tree.* tree; 2) *Priority Search K-Means Tree.*

2.2.2.1 Algoritmo Randomized k-d Tree

O algoritmo para ANN baseado em *Randomized k-d three* [44] se beneficia da construção de múltiplas árvores k-d, possibilitando a organização espacial dos objetos da base e a execução da busca paralelamente. As árvores são construídas de forma similar às tradicionais árvores k-d [7, 21], entretanto, na abordagem aleatória os objetos são divididos escolhendo-se aleatoriamente a dimensão a ser particionada dentre as N_d dimensões com maior variância. Os autores destacam que foi escolhido $N_D = 5$ empiricamente.

Durante a fase de busca, uma fila de prioridades mantém ordenadas a distância entre os limites de cada ramo em cada uma das árvores. Dessa forma, a busca seguindo essa fila de prioridades, vai explorar as folhas com os objetos indexados que sejam mais próximos do objeto consulta. Quando um objeto indexado é comparado, ele é marcado, e assim evita recálculos nas demais árvores. O critério de aproximação do algoritmo é determinado pela quantidade de folhas que são visitadas durante a busca. Dessa, que se todas fossem visitadas, o problema se resume a busca exaustiva.

Na figura 2.2 é possível observar como a organização do conjunto de dados por meio dessa estrutura de dados, provê a construção de *Buckets* com objetos próximos, dado a dimensão N_d da respectiva da árvore. Quanto mais dimensões são decompostas, aumenta-se a probabilidade do objeto mais próximo e do objeto consulta estarem alocados em uma mesma região.

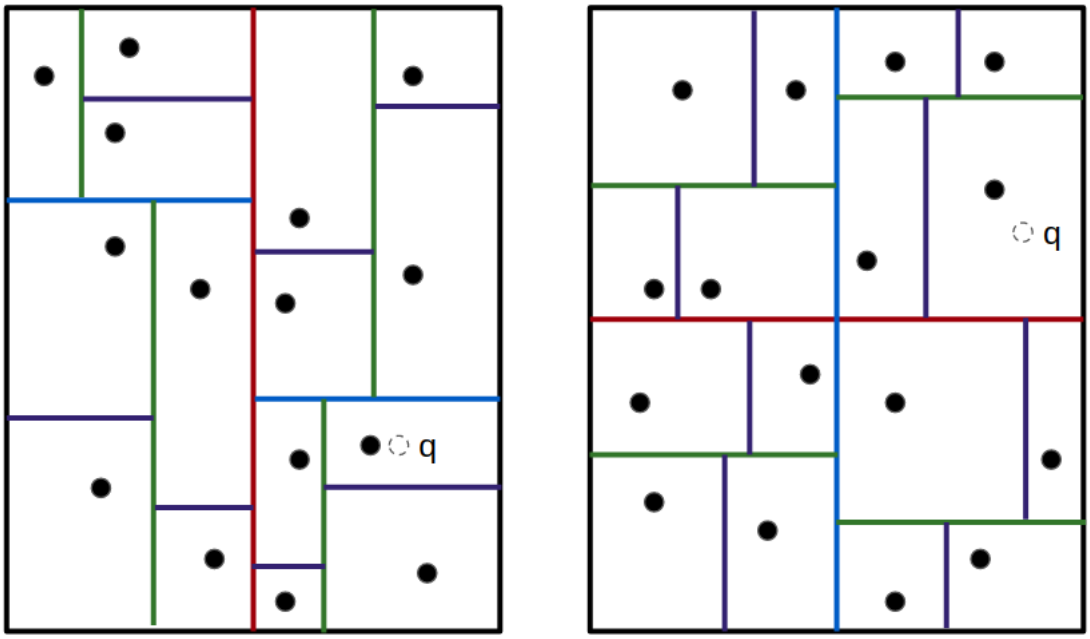


Figura 2.2: Regiões de duas *Randomized k-d threes*. A estratégia de particionamento das dimensões varia entre as árvores e assim, o objeto consulta encontrar objetos similares em regiões diferentes diferentes dado o particionamento aleatório.

2.2.2.2 Algoritmo Priority Search K-Means Tree

A intuição dessa estratégia é organizar o conjunto de dados espacialmente agrupando os objetos ao compará-los, levando em consideração todas as dimensões do espaço S, diferentemente da estratégia *Randomized k-d threes* onde apenas uma dimensão era particionada por vez (em cada árvore).

De forma geral, abordagens que se beneficiam de agrupamentos hierárquicos, como esse, já haviam sido propostos em outros trabalhos [22, 10]. Essas estratégias variam pela forma com que os objetos são agrupados e também pela maneira como a árvore construída é usada para a comparação com o objeto consulta. Os autores apontam que a abordagem implementada usa a estratégia *"best-bin-first"* para percorrer a árvore durante a fase de busca.

A solução por meio de *Priority search k-means threes* se mostrou mais eficientes em alguns conjuntos de dados onde era necessário maior precisão.

2.2.2.3 Escolha automática

A proposta do FLANN busca escolher automaticamente a melhor estratégia ANN para o conjunto de objetos de entrada, partindo da premissa de que é possível encontrar um conjunto de parâmetro ótimos para cada uma delas, e assim determinar qual performará com mais eficiência. Formalmente, os autores definem, que dado o espaço de escolha de parâmetro Θ, existe um conjunto de parâmetros $\theta \in \Theta$ onde $min_{\theta \in \Theta} c(\theta)$

Uma função de custo c é definida combinando o tempo de busca total ($s(\theta)$), o tempo de construção da árvore ($b(\theta)$), e o custo de memória do algoritmo ($m(\theta)$). Os autores destacam que cada um desses três fatores podem ter pesos diferentes (w_b, w_m) dependendo do cenário de aplicação. Formalmente a função de custo é definida como:

$$c(\theta) = \frac{s(\theta) + w_b b(\theta)}{min_{\theta \in \Theta}(s(\theta) + w_b b(\theta))} + w_m m(\theta)$$

Para encontrar o melhor algoritmo dado o conjunto de entrada. A proposta do FLANN é, em um primeiro momento, realizar testes combinatórios globais no espaço de opções dos parâmetros do algoritmo, e em sequência realizar testes combinatórios locais, partindo da melhor combinação encontrada no primeiro momento. Os autores enfatizam que essa forma não encontra o conjunto ótimo de parâmetros, entretanto se aproxima bem do melhor resultado.

O processo de escolha do melhor algoritmo ANN pode ser executado no conjunto de dados completo para uma otimização mais assertiva, ou em parte do conjunto de dados para uma otimização mais rápida. Independente da escolha da granularidade da otimização, o processo é executado apenas uma vez para cada conjunto.

2.2.3 PQANNS

A estratégia PQANNS (Product quantization in approximate nearest neighbor search [30]), emprega o método de Product Quantization para reduzir a dimensionalidade dos dados e uma estrutura de índice invertido para armazenar e organizar espacial os objetos quantizados. Dentre os métodos para ANNs mais relevantes, PQANNS se mostra mais eficiente e eficaz, apresentando melhores resultados para a vazão de consultas e menor consumo de memória para armazenar o conjunto de objetos. O desempenho e sofisticação desse método motivou a escolha, para que neste trabalho, fosse usada essa abordagem para os testes e experimentações propostas.

Nos momentos seguintes, será apresentado os conceitos de quantização para a redução de dimensionalidade dos objetos e então será descrito como a similaridade é computada no espaço quantizado. Por fim, detalharemos as fases do algoritmo PQANNS, evidenciando como os data buckets são construídos e como a busca acontece em regiões específicas.

2.2.3.1 Conceitos de Quantização

A ideia principal da quantização vetorial é mapear um vetor de alta dimensão x para um espaço de dimensão inferior. Entretanto, espera-se que a redução de dimensionalidade resulte em alguma perda de informação, já que detalhes do vetor original pode não ser completamente reconstruída a partir dos dados quantizados. No entanto, existe um benefício claro na manipulação de objetos em dimensionais inferiores, por exemplo, custos computacionais menores para calcular as distâncias usadas pela busca.

A quantização pode ser definida como uma função que leva o vetor de entrada x no espaço d-dimensional e cria uma representação de dimensão inferior $q(x)$, onde $q(x) \in C = \{c_i; i \in \tau\}$ com τ sendo um índice definido com z valores representativos para reprodução. Os valores de reprodução C são chamados de centroides e podem ser obtido em uma fase

de treinamento, por exemplo, executando um k-means em uma amostragem de objetos. O conjunto de C centroides é denominado codebook. O método de clusterização K-means podem ser usados aqui porque tende a encontrar pontos de centroide que são representativos para o conjunto de objetos de entrada. Em essência, o processo de quantização organiza os pontos em z células de forma semelhante a um diagrama de Voronoi. Assim, o vetor x é representado pelo $c_i \in C$ mais próximo onde:

$$q(x) = \arg\min d(x, c_i),\ c_i \in C. \tag{2.1}$$

O processo de quantização apresenta diferentes trade-offs de acordo com o tamanho do conjunto de centroides. Com um grande número de centroides, é possível representar um vetor (objeto) com mais precisão, e o processo de quantização seria eficaz, reduzindo a perda de informações. No entanto, à medida que o número de centroides cresce, maior é o custo computacional do processo de quantização. Product Quantization é uma estratégia para lidar com esse problema. A ideia básica dessa abordagem é alcançar a abrangência de um grande número de centroides com o uso de vários pequenos conjuntos de centroides. Isso é obtido dividindo a entrada vetor x em sub vetores a serem quantizados separadamente e combinando as sub quantizações para criar um grande número de possibilidades de quantização. Cada sub vetores u é de dimensão inferior, assumindo m subespaços a serem usados, os sub vetores $u_j, 1 \leqslant j \leqslant m$ são compostos por $D^* = D/m$ dimensões. Podemos definir o processo de Product Quantization da seguinte forma:

$$\underbrace{x_1, ..., x_{D^*}}_{u_1(x)}, ..., \underbrace{x_{D-D^*+1}, ..., x_D}_{u_m(x)} \rightarrow q_1(u_1(x)), ..., q_m(u_m(x)) \tag{2.2}$$

Como resultado, a quantização de x será um produto cartesiano da quantização de seus sub vetores:

$$q(x) = q_1(u_1) \times q_2(u_2) \times ... \times q_3(u_d) \tag{2.3}$$

Product Quantization permite que quantizadores de baixa complexidade sejam combinados por produto cartesiano, criando uma indexação de maior complexidade. Esta combinação de múltiplos pequenos centroides criam o codebook $C = C_1 \times \ldots \times C_m$. Onde, cada C_j contém z centroides, e existem z^m combinações, resultando em um codebook bem amplo.

2.2.3.2 Cálculo da distância entre objetos quantizados

O resultado do processo de quantização é um conjunto de objetos vetorizados representados por seus respectivos índices no codebook resultado do Product Quantization. Como consequência, durante fase de busca dos vizinhos mais próximos, a distância entre eles precisa ser computada no espaço quantizado. Em [30], foram propostas duas formas de calcular ou aproximar a distância entre um objeto de consulta x e os valores quantizados dos objetos armazenado no índice $(q(y))$.

A primeira abordagem é chamada Symmetric Distance Computation (SDC). Nessa abordagem calcula-se as distâncias usando os valores quantizados do objeto consulta x e dos objetos da base y. A distância $dist(x, y)$ é calculada usando os quantizadores pelo seguinte método:

$$\hat{d}(x, y) = d(q(x), q(y)) = \sqrt{\sum_{j=1}^{m} d(q_j(x), q_j(y))^2} \tag{2.4}$$

A segunda abordagem é a Asymmetric Distance Computation (ADC). O método ADC calcula as distâncias usando os quantizadores dos objetos da base indexados $(q(y))$ e o objeto original da consulta x. Esta abordagem usa x em vez de $q(x)$ na tentativa de reduzir os erros de distância devido à quantização. A distância calculada da seguinte forma.

$$\tilde{d}(x, y) = d(x, q(y)) = \sqrt{\sum_{j=1}^{m} d(u_j(x), q_j(u_j(y)))^2} \tag{2.5}$$

A Figura 2.3 ilustra a representação visual de ADC e SDC. Conforme discutido, a principal diferença é que o ADC usa o objeto de consulta original de x, enquanto SDC emprega sua quantização $q(x)$ nos cálculos de distância. Este trabalho usa ADC, pois mostrou melhorar a qualidade da pesquisa em comparação com SDC.

2.2.3.3 Algoritmo ANNS usando Product Quantization

O principal objetivo do processo de Product Quantization é reduzir a dimensionalidade dos objetos, resultando nos seguintes benefícios para o algoritmo de busca aproximada: (i) menores custos computacionais para o cálculo de distância e (ii) compactação no armazenamento dos objetos da base. No entanto, sabemos que mesmo em um espaço dimensional inferior, a busca exaustiva é impraticável. Portanto, na estratégia PQANNS

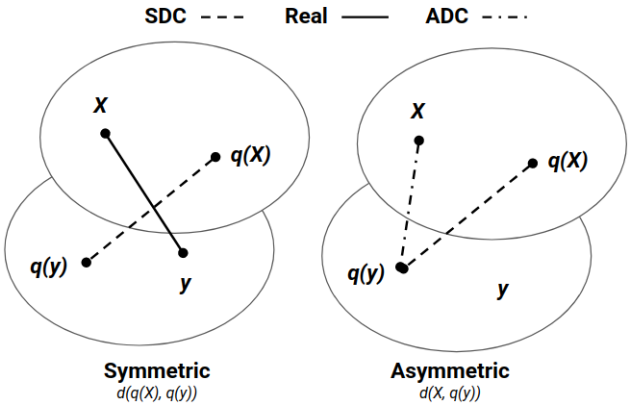


Figura 2.3: Representação de distância simétrica (esquerda) e assimétrica (direita) cálculos. A principal diferença é que a distância assimétrica usa o valor real de x ($dist(x, q(y))$), e SDC usa o valor quantizado de x ($dist(q(x), q(y))$). ADC mostrou melhorar a qualidade da pesquisa.

foi proposto o uso de um sistema de arquivo invertido com ADC (IVFADC) para podar o processo de busca e reduzir o número de cálculos de distância necessários. Esta abordagem agrupa objetos quantizado e similares da base para o mesmo centroide em entradas de um índice invertido e restringe a pesquisa à apenas algumas dessas entradas associadas à centroides que estão mais próximas do objeto de consulta.

A Figura 2.4 mostra a estrutura de indexação com o índice invertido proposto, onde cada entrada é representada por um centroide (valor quantizado mais abrangente) resultado de um processo de agrupamento usando k-means em um conjunto de dados de treinamento. Cada entrada do índice invertido contém uma lista de objetos quantizados (id e código, por objeto). O *id* pode ser uma identificação do objeto original que foi quantizado. O *code* é uma informação usada para melhorar a aproximação do cálculo da ADC, sendo a distância entre o objeto armazenado e o centroide abrangente correspondente (mais próximo). Além disso, *code* também é quantizado com Product Quantization para reduzir as demandas de armazenamento.

As fases de indexação e busca do IVFADC são apresentadas no Algoritmo 2. A indexação segue as seguintes etapas:

1. Calcula o centroide mais próximo do objeto de entrada y (linha 2).
2. Calcula o valor residual r_y de y para o k'-ézimo (mais próximo) centroide (C_c[k']) (linha 3)
3. Calcula a informação *code* por meio do Product Quantization do valor residual (linha 4)
4. Armazena *id* e *code* em uma entrada do índice invertido associado ao centroide mais próximo (linha 5)

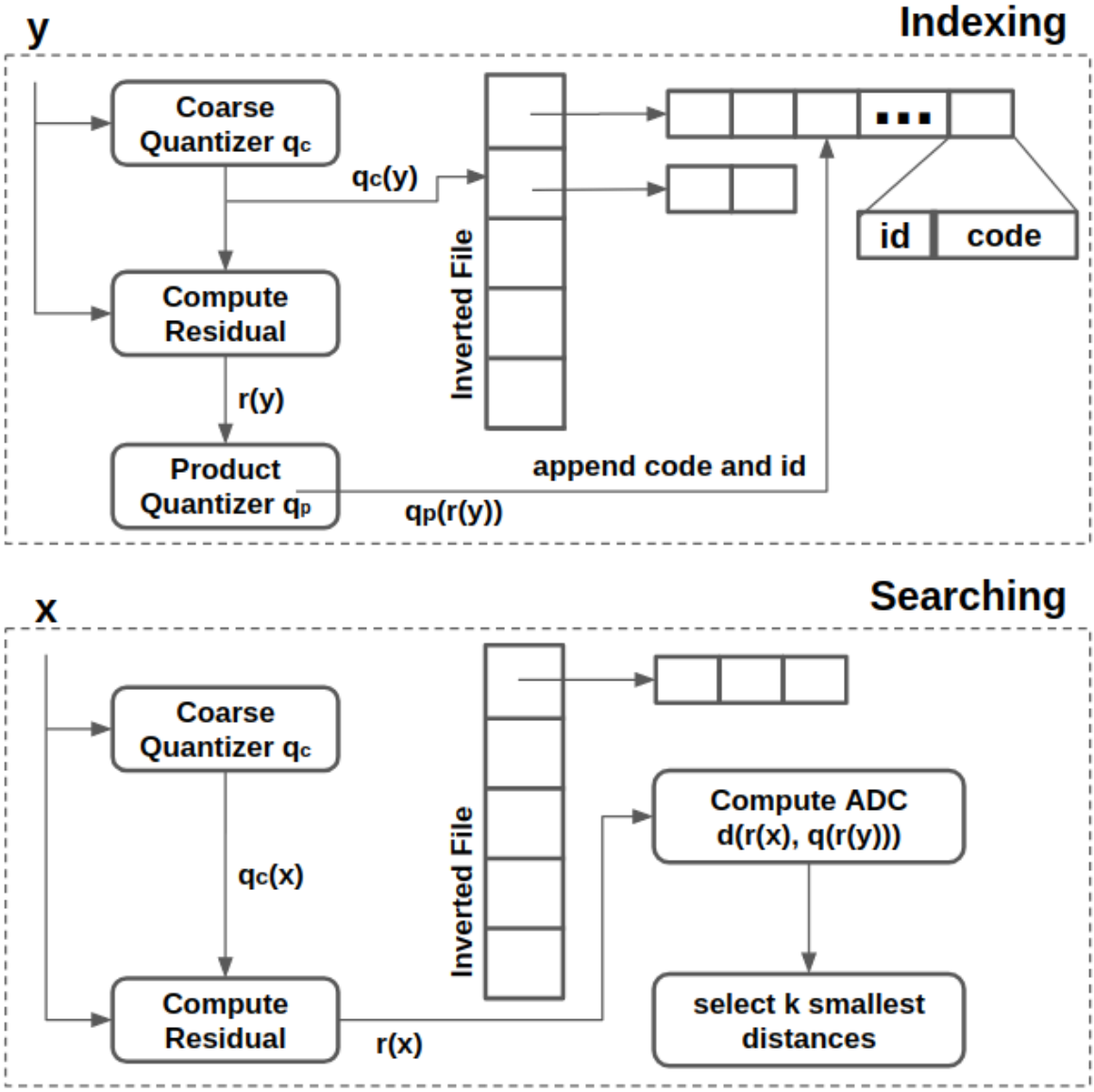


Figura 2.4: ANN com índice invertido e distância assimétrica (IVFADC)

A indexação produz o sistema de índice invertido onde os objetos da base de dados estão quantizados e armazenados nas listas invertidas correspondentes. Então, a fase de busca recebe como entrada o objeto de consulta (x), a quantidade de entradas do índice invertido para visitar (w), e o número de vizinhos mais próximos para recuperar (k), e executa as seguintes etapas:

1. Calcule os w centroides mais próximos e armazene-os em ncc

2. Pesquise nas entradas do índice invertido associadas a cada um desses centroides (linhas 8 a 11). Para cada entrada:

   a) Calcula o residual de x para o respectivo centroide (linha 9)

Algorithm 2: Indexação e Busca com IVFADC

```
1:  function Indexação(y)
2:      k' ← q_c(y)
3:      r_y ← y - C_c[k']
4:      code ← q_p(r_y)
5:      ivf[k'].append(y.id, code)
6:  function Busca(x, w, k)
7:      nnc ← kNN(C_c, x, w)
8:      for i ∈ 1 ... w do
9:          r_x ← x - C_c[ncc[i]]
10:         distList ← dist(index[ncc[i]], r_x))
11:         ann ← k-select(ann, distList,k)
12:     return ann
```

b) Calcula a distâncias ADC para todos os objetos armazenados na entrada do índice invertido (linha 10)

c) Seleciona os k objetos mais próximos e adicione-os no conjunto dos objetos encontrados nas entradas do índice invertido visitadas anteriormente. (linha 11)

3. Depois que as w entradas são visitadas, o resultado do conjunto *ann* é retornado.

2.2.4 Avaliação IVFADC

Esta subseção demonstra o desempenho do algoritmo IVFADC comparando-o com FLANN [39] e a abordagem de busca exaustiva. Como discutido antes, FLANN é uma linha de base interessante para avaliação de desempenho, usada em vários outros trabalhos, já que implementa diferentes estratégias de ANNS e escolhe a melhor para um determinado problema (conjunto de dados). O algoritmo selecionado por FLANN, neste conjunto de testes, foi *Priority Search K-Means Tree*. Por sua vez, a busca exaustiva foi implementada usando as bibliotecas BLAS/LAPACK para explorar o máximo de desempenho e servir como uma referência sólida para avaliação das compensações entre pesquisa exatas e aproximadas. Usamos um conjunto de dados com 1 milhão de descritores SIFT e 10K consultas neste experimento. A qualidade foi mensurada por meio da métrica 1-recall@1, que nos diz a porcentagem de resultados da consulta em que o vizinho mais próximo é classificado na primeira posição. Os experimentos foram executados sequencialmente usando um único núcleo de CPU, e o desempenho é avaliado conforme a precisão é variada. Neste caso, FLANN seleciona automaticamente os parâmetros para cada precisão

dada, enquanto variamos w e C_c em IVFADC (mostrado como w /# de centroides C_c na Figura 2.5), junto com R (tamanho da lista de k vizinhos mais próximos) necessário para alcançar uma determinada qualidade.

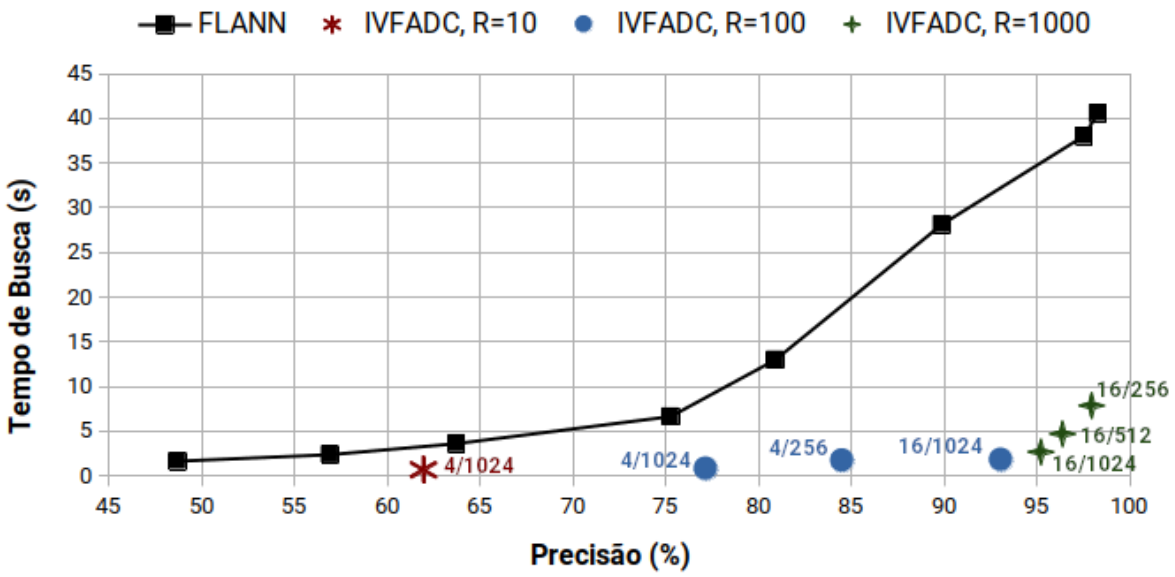


Figura 2.5: Comparação entre IVFADC e FLANN em diferentes níveis de qualidade da busca.

As compensações entre precisão e desempenho para IVFADC e FLANN são mostradas na Figura 2.5. Conforme apresentado, para o mesmo nível de precisão, IVFADC é significativamente mais rápido do que FLANN. Além disso, o IVFADC usa 26 MB de RAM, enquanto FLANN exigiu cerca de 600 MB de memória. O baixo custo de memória é outro atributo importante do IVFADC para permitir a indexação de conjuntos de dados muito grandes. Além disso, a busca exaustiva levou 57 segundos para ser executada, o que é muito mais lento do que o IVFADC, mesmo quando o último está configurado para atingir alta precisão (por exemplo, cerca de 95%).

2.3 ANNS em Memória Distribuída

As estratégias para ANNS presentes na literatura e discutidas anteriormente tem alcançados significativos resultados ao tornar possível à busca por similaridade em volumes de dados com alta dimensionalidade. Entretanto, essas soluções foram elaboradas para execuções sequenciais e em uma única máquina, com o conjunto de objetos alocados na memória principal. ANNS é um passo chave para aplicações CBMR que manipulam

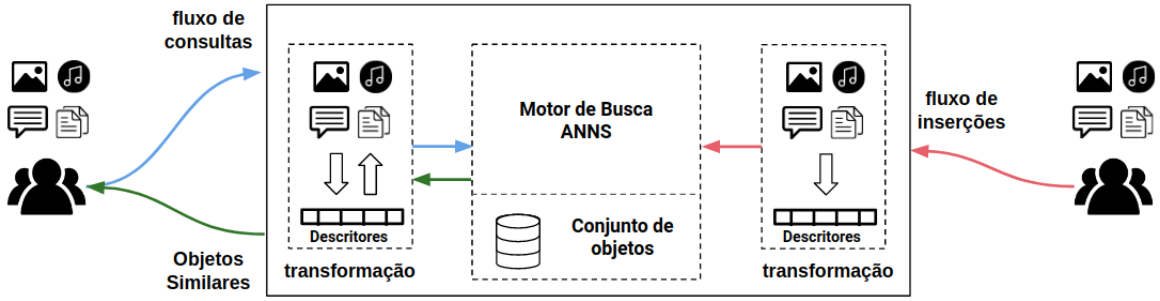


Figura 2.6: Representação do relacionamento entre componentes de uma aplicação para busca kNN em conjunto de objetos multimídia.

grandes volumes de dados, como por exemplo motores de busca de imagens na web; associação de rótulos em imagens presentes em redes sociais; identificação em tempo real de sons, dentre outras tantas envolvendo intenso fluxo de inserção de novos objetos e consultas. Essas características de aplicações excedem à capacidade de uma única máquina conseguir armazenar e executar o processo de busca em tempo hábil.

Partindo dessas observações, soluções paralelas e em memória distribuída para ANNS [48, 6, 38, 34, 39, 51, 3], foram desenvolvidas para tornar possível a busca aproximada em contextos de grandes volumes de objetos em alta dimensão, com intenso fluxo de inserções e consultas. Essas soluções tiram proveito de recursos computacionais distribuídos e heterogêneos e propõe componentes para organizar o conjunto de objetos em diversas máquinas.

Dito isto, a complexidade das aplicações CBMR, tendo ANNS como parte do fluxo de processamento de dados varia. Entretanto, alguns componentes são comuns à elas, e estão ilustrados na figura 2.6. É importante esclarecer, que, do ponto de vista da aplicação CBMR, os objetos vetoriais presentes no espaço S, conforme pontuado na definição 2, são objetos multimídia, que podem ser imagens, sons, texto, vídeos, etc, representados via vetores de características em alta dimensão, os quais nomearemos de*descritores*. Essa representação numérica em vetores d-dimensionais pode ser feita de diferentes formas, sendo Scale Invariant Feature Transform (SIFT), Vector of Locally Aggregated Descriptors (VLAD), Gist, Bag-of-Features (BoF), e Deep Features (100-1000+dimensions) [36, 53, 25, 18, 31] as mais conhecidas, e que são detalhadas durante a subseção 2.3.1. Por isso, existem componentes no núcleo dessas aplicações, que transformam objetos multimídia em descritores vetoriais equivalentes, e da mesma maneira, traduzem os descritores resultado nos objetos multimídia correspondentes. Esses componentes não fazem parte do escopo de contribuições deste trabalho que concentra os esforços na otimização do motor de busca.

O motor de busca implementa a estratégia ANNS escolhida, organiza e interagem com o conjunto de objetos. É o componente responsável por receber e gerenciar fluxos constantes de descritores. Os fluxos de dados podem ser:

- **Fluxo de Consultas**: Descritores de consulta são enviados periodicamente para

que o motor de busca encontre os descritores mais similares no conjunto de objetos.

- **Fluxo de Inserções**: Novos descritores são enviados para que o motor de busca complemente o conjunto de objetos. Compõe cenários específicos de aplicações que permitem a inserção de novos objetos em paralelo às consultas.

Ambos os fluxos possuem intensidades variáveis ao longo da execução da aplicação, sendo que, ao passo que as intensidades aumentam, as complexidades e desafios para gerenciar o volume de dados também crescem. Assim, para mensurar a performance do motor de busca, algumas métricas são comumente usadas:

- **Tempo de Resposta**: A primeira delas é o tempo de resposta $T_r(q) = T_{wait}(q) + T_s(k, q)$. Essa métrica nos diz quanto tempo foi gasto desde a entrada do objeto de consulta q no motor de busca, até o término da execução da estratégia ANNS respondendo com os seus k vizinhos mais próximos. Podemos perceber que está diretamente relacionada ao tempo gasto para iniciar a execução da busca T_{wait} e também a ao tempo total gasto pelo processo de execução da busca. O tempo de execução da busca $T_s(k, q)$, é o tempo gasto pela estratégia ANNS para encontrar os k objetos mais similares ao objeto consulta q. Conforme discutido amplamente no Capítulo 2, esse compromisso pode ser balanceado pelo nível de assertividade esperado como resultado, onde alto nível de assertividade está relacionado à um tempo de busca T_s maior.

- **Vazão de Consultas**: A vazão nos diz a quantidade de consultas que o motor de busca é capaz de responder em um intervalo de um segundo. Essa métrica está diretamente relacionada ao tempo médio que o componente gasta para responder às consultas individualmente $avg(T_s)$ e pode ser expressada como: $T_v = 1/avg(T_s)$.

As soluções paralelas e distribuídas para ANNS encontradas na literatura propõe mecanismos para otimizar o motor de busca, por meio da organização do conjunto de dados e por estratégias de paralelização para lidar com os fluxos de objetos. De forma abrangente, as soluções de alta performance para ANNS podem ser organizadas em dois conjuntos, os quais abordam complexidades diferentes dado a dinâmica do conjunto de objetos:

1. **Coleções de Objetos Estáticos**: Nesse contexto de aplicações, a coleção de descritores é estática ao longo de todo o processo de busca. Assim, as soluções nesse grupo devem lidar com os aspectos de alta performance e escalabilidade para tornar possível a aplicação de estratégias ANNS em grandes volumes de dados considerando a gestão do Fluxo de Consultas.

2. **Coleções de Objetos Dinâmicos**: Nesse conjunto de aplicações, as soluções de alta performance para ANNS devem lidar com Fluxo de Consultas e Fluxos de Inserções, concorrentemente. Nesse contexto mais desafiador, a coleção de descritores

cresce ao passo que novos descritores chegam pelo Fluxo de Inserções e compõe a base de busca. Assim, além de endereçar as questões de escalabilidade e performance para tornar possível a aplicação de estratégias ANNS em grandes volumes de dados, as soluções devem endereçar a gestão do conjunto crescente de objetos e também o controle de múltiplos fluxos de dados.

Nas subseções 2.3.2 e 2.3.3 serão discutidos as propostas mais relevantes para cada um dos grupos propostos, bem como os problemas e desafios que motivam as contribuições deste trabalho.

2.3.1 Transformação de Objetos Multimídia em Descritores

Aplicações que lidam com objetos multimídia (imagens, vídeo, som, etc), em sua grande maioria, se beneficiam de representações vetoriais com múltiplas dimensões para armazenar e processar itens. Por meio destas representações, é possível agregar conjuntos de informações relevantes que tornam possível identificar com precisão características individuais e únicas em cada objeto. Apesar de amplamente aplicada em contextos de visão computacional, como reconhecimento de objetos e cenas, renderização 3D, perseguição de movimento, etc. Para ANNS em CBMR, a transformação em uma correspondência vetorial numérica é essencial para o mapeamento espacial de objetos multimídia, o que possibilita a aplicação dos mecanismos de poda e redução de dimensionalidade proposto pelas soluções.

Dessa forma, uma abordagem popular, é a representação de objetos multimídia por meio de descritores locais, que podem ser definidos como vetores de características de objetos, ou regiões dos objetos, computados em pontos de interesse relevantes, tendo bastante sucesso em aplicações de reconhecimento de imagens. Vários trabalhos trazem técnicas para descrever regiões locais de imagem sendo que os mais relevantes e recentes concentram esforços em produzir descritores que sejam invariantes as possíveis transformações, e dentre esses a técnica SIFT [36] possui bastante destaque.

Tirando proveito da extração de descritores locais por meio de SIFT, algumas outras abordagens propõe a representação de imagens e vídeos usando diferentes técnicas de agregação de características. A mais conhecida no contexto de busca por similaridade é a Bag-of-Features (BoW), seguida por uma otimização conhecida como Vector of Locally Aggregated Descriptors (VLAD) [31].

Por fim, em paralelo as abordagens que partem da detecção e extração de descritores locais, técnicas usando Deep Learning, são capazes de representar imagens e diferentes

objetos multimídia por meio das características identificadas e representadas por meio dos pesos encontrados em camadas de redes neurais. Esse conjunto de pesos consolidam a descrição de objetos por meio de vetores numéricos multidimensionais conhecidos como Deep Features [53, 25].

2.3.1.1 Scale Invariant Feature Transform (SIFT)

SIFT é uma técnica bastante popular, proposta originalmente em [37], para a detecção e extração de descritores locais em imagens. Por meio desta abordagem é possível comparar imagens por diferentes visões, já que as características extraídas são minimamente invariáveis à algumas transformações como mudanças de perspectivas, escala, giros ou rotações, ruídos e mudanças de iluminação.

SIFT é bastante adequada para tarefas de comparação entre imagens, já que produz uma grande quantidade de descritores locais, cobrindo densamente as regiões, além de serem altamente distintos, proporcionando a capacidade de identificação com precisão de um objeto em uma grande coleção.

A obtenção de descritores SIFT seguem algumas etapas principais: i) Identificação de regiões relevantes: durante esse primeiro passo, o método SIFT encontra localizações em imagens que são invariáveis à escala e rotação; ii) Detecção de pontos chave: Por meio da diferença de filtros gaussianos pontos chaves são selecionados através de medidas de estabilidade. iii) Determinação da orientação dos pontos chaves: esse passo torna possível que os pontos chaves encontrados não sofram interferência das variações discutidas.; iv) Descritor dos pontos chaves: Construção dos descritores numéricos das regiões identificadas.

2.3.1.2 Bag-of-Features (BoF)

A técnica de Bag-of-Features representa imagens por meio de um vetor esparso de ocorrências de característica locais. Tendo como inspiração à técnica Bag-of-words presente no contexto de classificação de documentos, uma imagem pode ser tratada como um "documento"contendo uma coleção de "palavras"(características locais). De maneira ampla, esse método pode ser encarado como a construção de representações de histogramas de características locais e independentes. Para alcançar esse resultado alguns passos prin-

cipais são aplicados: i) Detecção das características locais; ii) Construção dos descritores locais e iii) geração do *Coodebook*.

A detecção e construção de descritores locais (passos i e ii) tira proveito de abordagens como SIFT para representar imagens por meio de coleções de vetores multidimensionais (cada um representando características locais). Assim, cada vetor é mapeado em *codeworkds*, e agregado em um dicionário conhecido como *Coodebook*. De forma ampla, *codeworkds* são definidos como centroides representativos extraídos por meio de técnicas de agrupamento como k-means, tendo como base todo o vetor de descritores locais. O *codebook* reúne o conjunto de *codewords*, e por meio deste mapeamento, é possível representar uma imagem por um histograma de *codewords*.

Indo além, a abordagem Fisher Vectors (FVs) [29] apresenta uma melhoria à técnica BoF, ao acrescentar, junto do histograma, a proximidade de descritores ao *codeword* mais próximo, possibilitando uma normalização que pode ser interpretada como uma medida IDF (Inverse Document Frequence).

2.3.1.3 Vector of Locally Aggregated Descriptors (VLAD)

A abordagem *Vector of Locally Aggregated Descriptors (VLAD)* é uma solução baseada em BoF e muito similar a Fisher Vectors. Nessa proposta, que pode ser vista como uma melhoria e simplificação à BoF, para cada descritor x ao associá-lo à um *codework* $c_i \in C = c_1, ..., c_k$ tal que $c_i = NN(x)$ e C é o *Codebook*, acumula-se o residual dado pela diferença $x - c_i$. Dessa maneira, a representação v de uma imagem é simplesmente dado pela agregação da soma dos residuais de cada descritor, conforme apresentado no trabalho [31]:

$$v_{i,j} = \sum_{x|NN(x)=c_i} x_j - c_{i,j}$$

onde x_j representa o j-*ésimo* componente do descritor x e $c_{i,j}$ o *codeword* mais próximo a ele. Os autores destacam que esta abordagem traz benefícios mesmo com um pequeno conjunto de *codewords*, eliminando a esparsidade decorrente da estratégia BoF e otimização o uso de memória.

2.3.1.4 Deep Features

Em redes neurais para classificação de imagens, o conjunto de treino é formado por uma coleção de imagens nas quais guiam a estrutura de pesos e organização das camadas da rede. De maneira abrangente, nas camadas mais "superficiais"algumas características óbvias são diretamente identificadas como contornos e bordas das imagens, entretanto, nas camadas mais "profundas"características mais complexas podem ser extraídas, e também criadas, pela capacidade da rede neural em correlacionar características mais simples.

Dessa maneira, as camadas mais profundas de redes neurais treinadas a partir de conjuntos de imagens, consolidam vetores multidimensionais com representações de características das imagens de entrada. Essa abordagem é uma alternativa as citadas anteriormente e vem atraindo bastante atenção para aplicações de recuperação de imagens CBIR.

2.3.2 ANNS em Conjunto de dados Estáticos

As abordagens paralelas e em memória distribuída para ANNS presentes nessa subseção consideram o conjunto de descritores para busca estático. São diversas as estratégias encontradas na literatura que se diferem em muitos aspectos. De forma abrangente, a maneira como a base de descritores é particionada entre os componentes da solução paralela definirá os compromissos que serão otimizados e as limitações associadas. Alguns trabalhos incluem abordagens baseadas na paralelização por meio do paradigma MapReduce [48, 6, 38, 26, 61], onde o conjunto de objetos é armazenado em um sistema de arquivo distribuído acessado por um ou vários processos paralelos. Por sua vez, outras estratégias propõem o particionamento da base de dados entre nós de computação distribuídos [49, 39, 51, 3, 47].

Nos tópicos seguintes serão apresentadas detalhes das técnicas paralelas propostas, por meio de diversos algoritmos ANNS. O objeto é mostrar os mecanismos usados para acelerar e tornar possível a busca em grandes volumes de objetos.

2.3.2.1 Estratégias por MapReduce

As técnicas por meio do paradigma MapReduce são pioneiras. Em [48], baseado no algoritmo LSH, a Tabela Hash é armazenado em um sistema de arquivo distribuído, que é visível para todos os nós de computação presentes na solução paralela. A busca pelos vizinhos mais próximos é então realizada de forma independente em cada nó, acessando os *buckets* necessários no sistema de arquivo para responder à consulta. O acesso à base de dados se assemelha à uma distribuição de balanceada de *buckets* que aqui chamaremos de ***Bucket Equal Split (BES)***, porém nenhum objeto é desalocado do armazenamento principal. Como pode ser observado, esta estratégia é intensiva em leitura/escrita (I/O) e pode apresentar desempenho limitado devido à falta de localidade dos dados. Esses gargalos de I/O, proporcionados pelo acesso centralizado em um sistema de arquivos distribuído, foram endereçados em [6], onde é proposto o uso de uma tabela de hash distribuída ativa (DHT) para armazenar o índice de objetos em uma memória. Essa estratégia, no entanto, limita a pesquisa a um único nó de computação, que é uma limitação significativa para a maioria dos algoritmos de ANNS que pretendem lidar com grandes quantidades de dados.

Indo além, e também usando mecanismos MapReduce, uma implementação para memória distribuída de uma estratégia ANNS baseada em Árvore de Similaridade é apresentado em [38]. Esta abordagem armazena o índice em um sistema de arquivos distribuído, tendo a mesma limitação apresentada anteriormente. No entanto, é proposto a uma otimização de reordenamento do fluxo de consultas, que traz um aspecto interessante para reduzir o volume de acessos aos dados do sistema de arquivos. Em outro trabalho [26] com relevante resultado, a solução tira proveito do *framework* Spark [61] para a implementação de uma estratégia ANNS em memória distribuída. Nesse caso, o conjunto de objetos é armazenado em estruturas chamadas Resilient Distributed Datasets (RDD) para mantê-lo na memória após o primeiro acesso. Esta abordagem foi projetada para lidar com fluxos intensos de consulta, e é capaz de amortizar as despesas gerais do sistema.

Todas essas limitações presentes nas soluções que se baseiam no paradigma MapReduce, motivaram o surgimento de paralelizações que particionam o conjunto de objetos (ou os conjuntos de *buckets*) entre os nós de computação do sistemas de memória distribuída. Essa forma de distribuir o conjunto de objetos é versátil, já que permite cada componente da solução paralela lidar com conjuntos reduzidos de objetos otimizando o uso dos recursos locais. Assim, são soluções que se destacam pelos resultados significativos em aplicações de ANNS com fluxo intenso de consultas.

Essas abordagens foram implantadas baseando na estratégias ANNS: LSH [49], FLANN [39], Multicurves [51], e IVFADC [47].

2.3.2.2 Estratégias por Particionamento dos Dados

As paralelizações em memória distribuída empregadas pelas soluções baseadas no mecanismos de particionamento e distribuição do conjunto de objetos [49, 39, 51, 47], tem como base uma abordagem Master-Worker. Nessas estratégias, a base de dados é igualmente distribuída entre os componentes Worker do motor de busca, onde indexam localmente os objetos (descritores) recebidos. Quando um objeto de consulta chega no sistema, o componente Master o envia para todos os componentes Worker, seguindo uma estratégia de *broadcasting*. Localmente, cada componente Worker realizará a busca na sua partição de dados, seguindo a estratégia ANNS base e respondendo ao Master com k objetos locais mais próximos à consulta. Por sua vez o Master fica encarregado de receber os resultados locais e agregá-los em uma resposta global. Essa abordagem intuitiva de paralelização em memória distribuída remonta uma estratégia distribuição/agregação, e nesse trabalho à chamaremos de ***Data Equal Split (DES)***. É importante destacar que esse modelo de paralelização é agnóstica à estratégia de ANNS usada localmente em cada nó de computação.

A Figura 2.7 ilustra essa estratégia de paralelização. Nela estão representados o fluxo de consultas (linhas azuis) e os nós de computação: Master e Workers.

A Distribuição de dados entre nós de computação para otimizações de ANNS em larga escala, inclui desafios interessantes. Para cada nó de processamento executar a mesma quantidade de trabalho e houver gargalos de processamento (nós extremamente mais rápidos ou mais lentos), a distribuição de dados precisa ser cuidadosamente balanceada. Assim, os trabalhos [23, 60, 50, 1] apresentam formas de balanceamento de dados e organização da execução do fluxo de consultas, para explorar plenamente o potencial da máquina de memória distribuída durante a execução da estratégia ANNS.

Trabalhos recentes relacionados à ANNS e também à outras operações com características similares, endereçam pesquisas e soluções para balanceamento de carga [23, 50, 1]. Em [23], é implementado um algoritmo chamado Lista de Clusters (LC) [11], e sua paralelização acontece ao distribuir listas de cluster do conjunto de objetos (*buckets*) entre os componentes presentes na máquina de memória distribuída. A estratégia de balanceamento de carga proposta replica listas de cluster no sistema distribuído para permitir que a carga seja equilibrado. Esta estratégia não considera a proximidade espacial dos clusters atribuídos em um mesmo nó de computação e também pode incorrer em sobrecarga significativa devido aos dados replicação. Além disso, o algoritmo LC tem uma maior demanda de memória em comparação com outras estratégias, como por exemplo o IVFADC, que também se baseiam na criação dos *buckets* por meio da clusterização dos objetos, entretanto esse último apenas armazena uma representação compacta dos descritores. Outros trabalhos [50, 1] foram desenvolvidos em técnicas de balanceamento

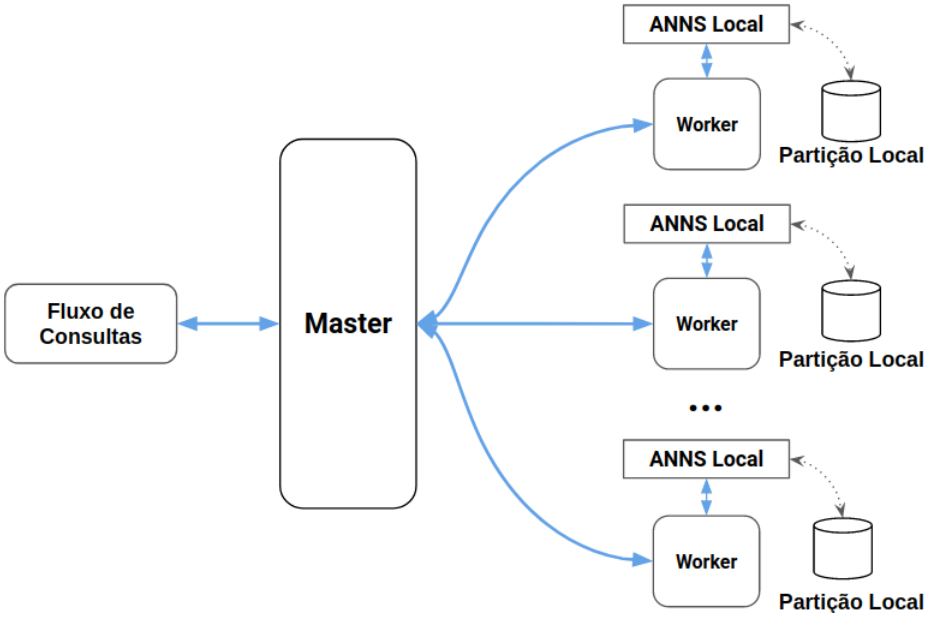


Figura 2.7: Paralelização em memória distribuída para ANNS. Nessa solução, uma abordagem Master-Worker é empregada, e a base de dados inicial é distribuída igualmente entre os componentes Workers. No fluxo de consulta (linhas azuis), o componente Master é responsável por encaminhar os descritores de consulta para todos os Workes e receber os resultados das buscas locais, consolidando os k descritores mais próximos. Nessa abordagem, é possível instanciar diferentes estratégias de ANNS para a indexação das partições dos dados localmente nos Workers.

de carga para a operação All Pairs Similarity Search (APSS). Nestes trabalhos, um grafo ponderado é usado para representar a relação entre os objetos, e esta relação é ainda usada para reduzir o desequilíbrio e minimizar a computação da similaridade entre os pares.

Apesar dessas soluções citadas para o balanceamento de carga não terem sido extensivamente avaliadas em ANNS, elas inspiram otimizações importantes para novos conjuntos de abordagens em memória distribuída nesse contexto de pesquisa.

2.3.2.3 Desafios e Oportunidades

As diferentes estratégias para ANNS em conjunto de Dados estáticos diferem-se em muitos aspectos e assim é proposto uma sumarização baseada em algumas características: (i) o *framework* usado para a implementação; (ii) a capacidade de manter o conjunto de dados indexado em memória, tornando possível recuperar rapidamente os vizinhos mais próximos de um objeto de consulta; (iii) a estratégia de particionamento de dados usada; (iv) o uso ou não de mecanismos de distribuição de dados baseados na proximidade (*locality-aware*) dos objetos, e (v) balanceamento de carga usado. A comparação das

abordagens citadas está sumarizada na Tabela 2.1.

ANNS	(i)	(ii)	(iii)	(iv)	(v)
LSH [48]	MapReduce [16]	✗	BES	✗	✗
LSH [6]	DHT [52]	✓	BES	✗	✗
Index Tree [38]	MapReduce [16]	✗	BES	✗	✗
eCP [26]	Spark [61]	✓	BES	✗	✗
LSH [49]	MPI [12]	✓	DES	✗	✗
FLANN [39]	MPI [12]	✓	DES	✗	✗
Multicurves [51]	Anthill [19]	✓	DES	✗	✗
IVFADC [47]	MPI [12]	✓	DES	✗	✗
LC [23]	MPI [12]	✓	BES	✗	✓
Neste Trabalho (IVFADC)	MPI [12]	✓	DES/BES SABES SABES++	✓	✓

Tabela 2.1: Comparação entre as estratégias paralelas e distribuídas para ANNS.

Apesar dos trabalhos citados alcançarem interessantes resultados e em alguns casos atingirem o objetivo de tornar possível a Busca aproximada em grandes volumes de dados altamente dimensionais, existem características nas quais observamos ser fraquezas e assim nos guiam em potenciais contribuições:

1. Apesar de extremamente popular e amplamente usada entre as estratégias paralelas e distribuídas para ANNS, o mecanismo DES de particionamento sofre com significativos gargalos de comunicação e despesas por gestão e controle do sistema, visto que, para cada consulta todos os componentes Workers precisam ser usados. Esse aspecto limita a escalabilidade e performance paralela do motor de busca.

2. Além disso, as estratégias de particionamento e distribuição dos dados não consideram a associação à um mesmo nó de computação *buckets* que são próximos no espaço. Associar regiões em um mesmo nó de computação tem grande potencial em otimizar o processo de busca já que *buckets* próximos no espaço tem grandes chances de serem acessados juntos ao executar a consulta por vizinhos próximos.

3. Por mais que algumas propostas endereçaram abordagens para o balanceamento de carga, esse aspecto não está integrando ao problema de ANNS propriamente. Assim, garantirá a alocação de *buckets* similares em um mesmo nó de computação mantendo o balanceamento de carga, pode potencializar os ganhos para as soluções ANNS em sistema distribuído.

Esses desafios motivam as contribuições iniciais deste trabalho, que serão detalhadas no Capítulo 3. Para fins de posicionamento na Tabela 2.1, a última linha aponta os aspectos que propomos avançar no estado da arte.

2.3.3 ANNS em Conjunto de dados Dinâmicos

As soluções em memória distribuídas para ANN devem ser capazes de lidar com desafiadoras características de tempo real dos cenários de aplicações CBMR. Diferentemente das abordagens comentadas na seção anterior, compreendemos aqui que o conjunto de dados, não é estático e estar em constante crescimento.

Ao passo que as bases de dados multimédia estão sendo consultadas intensamente, existe em paralelo, um grande fluxo de novos objetos (Fluxo de Inserções) sendo produzidos e que precisam ser indexados para compor o conjunto de busca já disponível. Essa é a realidade de inúmeras aplicações que se tornaram populares e amplamente usadas nas últimas décadas. Um exemplo central são as redes sociais, nas quais é possível compartilhar e interagir com vários objetos multimídia, permitindo o consumo e produção, em larga escala e em tempo real. Junto à característica do crescimento constante do conjunto de objetos, parte dele pode se tornar obsoleto e pouco interessante para o contexto da aplicação, forçando as soluções de ANNS serem resilientes ao descarte frequente de dados indexados. Por sua vez, Fluxos de Consultas e Fluxos de Inserções possuem intensidades variáveis ao longo da permanência do sistema. Existem movimentos, em que as aplicações online recebem maior intensidade de consultas, ao passo que em outros momentos a intensidade de inserção é maior.

Entendemos que todas as características apontadas permeiam compromissos de alta performance que as soluções ANNS nesse contexto de aplicações devem endereçar: i) Escalabilidade e performance paralela em máquina com memória distribuída; ii) Organização e gestão dinâmica e adaptativa dos do conjunto crescente do objetos que compõem a base de busca; iii) Gestão eficiente dos fluxos concorrentes, adaptando-se aos momentos de intensidade variáveis.

Poucos trabalhos na literatura consideram esse cenário complexo de aplicações para ANNS. Como estado da arte a proposta apresentada em [49] mostra uma solução que endereça os compromissos apresentados anteriormente alcançando significativo resultado para um volume expressivo de dados (1 bilhão de objetos). A solução paralela e em memória distribuída é baseada na estratégia LSH, onde o conjunto de objetos é particionado igualmente entre os nós de computação (estratégia de particionamento DES) e em cada um as Tabelas Hashs de consulta (nomeadas de *Static Tables*) são construídas para indexar os objetos presentes na partição e assim construir os *buckets* locais. Com essa abordagem, a proposta garante a escalabilidade e a gestão eficiente do Fluxo de consultas. Extensivamente, a arquitetura proposta para o motor de busca lida com o Fluxo de Inserções e com o descarte de objetos antigos. Dessa forma, é proposto que em cada nó do sistema distribuído, seja construída, além da Tabela Hash para consulta, uma Tabela Hash auxiliar otimizada para indexação ágil de novos descritores. Essa tabela auxiliar,

nomeada no trabalho de *Delta Table* é responsável por receber a indexação frequente dos objetos que chegam por meio do Fluxo de Inserções.

Periodicamente, as *Delta Tables* devem ser combinadas com as *Static Tables* para que assim, *buckets* mais antigos (indexados nas *Static Tables*) possam ser descartados e renovados pelos *buckets* mais recentes presentas na *Delta Tables*. A periodicidade do processo de combinação de tabelas é controlado por um parâmetro η que diz qual o limite (percentagem em relação à capacidade total de dados suportada no nó) de novos objetos devem ser indexados na *Delta Tables*. Diretamente, compreendemos que se esse parâmetro é amplo (valor alto) menos combinações vão acontecer, porém em *Delta Tables* com maior volume de objetos (maior custo computacional do processo de combinação). Em contrapartida, valores mais restritos do parâmetro η levam a combinações frequentes com pouco volume de dados. O balanceamento entre os compromissos decorrentes da periodicidade do processo de combinação é particular em cada cenário de aplicações e precisa ser empiricamente avaliado. Em [49] os autores apontam que $\eta = 10\%$ permite que a combinação aconteça a cada 1 milhão de inserções, o que corresponde à um intervalo de 864 segundos, tendo como base o parâmetro $M = 4$ e o cenário de teste específico apresentado. As consultas que chegam no sistema enquanto o processo de combinação está acontecendo são armazenadas em *buffers*, para execução posterior. As *Delta Tables* são zeradas ao fim do processo de combinação. O descarte de *buckets* antigos é realizado arbitrariamente quando a combinação proporciona um acúmulo de objetos na *Static Table* que excede um parâmetro de espaço definido. Os autores apontam que as despesas totais com esse processo, correspondem à um tempo total de 1.7% do total de execução do motor de busca.

Para permitir que consultas aconteçam simultaneamente á inserções, além do mecanismo da separação de Tabelas Hashes que atendem as demandas separadamente, os autores propõe que as inserções de novos descritores aconteçam em uma janela deslizante de nós de computação. Dessa maneira, o parâmetro M da solução guia quantos componentes *Workers* irão receber o Fluxo de Inserções em determinado momento. Nesse sentido, apenas os nós presentes na janela corrente instanciam *Delta Tables*. Essa estratégia, além de funcionar como uma gestão dos fluxos concorrentes garante que a distribuição dos novos objetos mantenha a estratégia de particionamento DES usada pela solução. Na Figura 2.8 estão representandos os componentes e fluxos descritos pela proposta apresentada em [49].

Os aspectos dessa solução que tornam possível a execução concorrente de Fluxo de Consulta e Fluxo de Inserções e também o descarte de *buckets* mais antigos evidenciam, além da organização espacial, uma organização temporal dos *buckets*. De maneira bem simples, *buckets* presentes na *Delta Table* estão em um recorte temporal mais recente que *buckets* presentes na *Static Table*. Podemos então definir essa concepção temporal dos dados como:

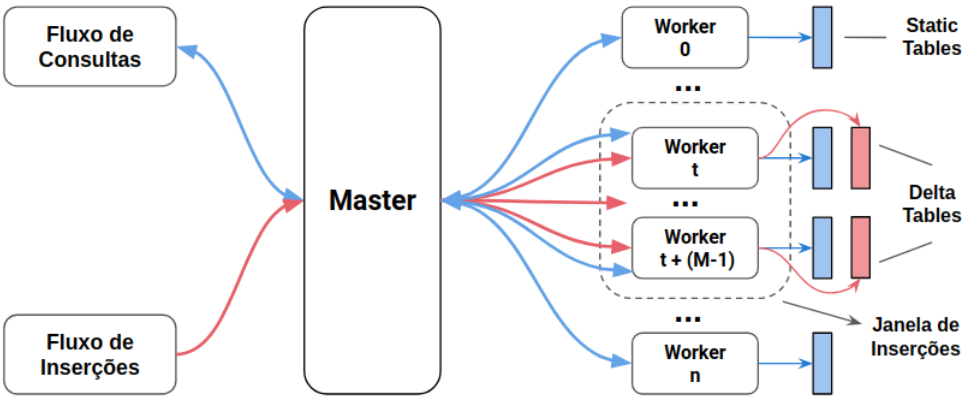


Figura 2.8: Arquitetura em máquina distribuída proposta em [49]. A solução se baseia na estratégia de busca LSH, particionando igualmente o conjunto de objetos entre os nós de computação. Em cada nó existe uma *Static Table* responsável por responder localmente as consultas. Dentro de uma janela de M nós, são instanciadas *Delta Tables*, nas quais irão ser indexados os novos objetos que chegam por meio do fluxo de inserções. Periodicamente as *Delta Tables* são combinadas com *Static Table*, para que assim, a base de busca se renove e aconteça o descarte de *buckets* antigos.

Definição 7. ***Bucket temporal*** *são conjuntos de objetos próximos no espaço, compondo um bucket (vide definição 6) e, também, que foram indexados em instantes próximos no tempo. Um instante temporal t_n está n intervalos (granularidade de tempo) distante do início do ciclo de vida da aplicação. Formalmente, um bucket Temporal TX_n é tal que $\forall x, y \in TX_n$ e $v \in (X - TX_n) | dist(x, y) \leq dist(x, v)$ e, também, $t(x) = t(y)$ e $t(x) \neq t(v)$, onde t é instante em que algum objeto foi indexado.*

Agregando esse aspecto temporal aos dados, podemos perceber que na solução proposta em [49] a consulta é sempre realizada em *buckets* com uma flexibilidade de desatualização de até 864 segundos, considerando o parâmetro $\eta = 10\%$. Já que a *Static Table* é atualizada com os objetos novos da *Delta Table* em intervalos de 864 segundos. Para esse contexto de aplicação esse atraso na atualização é aceitável, entretanto em alguns outros cenários esse aspecto pode ser mais restrito ou mais amplo. Assim, esse compromisso é definido como:

Definição 8. ***Flexibilidade de Desatualização do Bucket (Bucket Outdated Flexibility ou BOF)*** *é um parâmetro da aplicação que guia qual o intervalo de desatualização do conjunto de objetos é aceitável para que a busca aconteça. Um bucket é dito desatualizado, quando existem indexações pendentes por um período de tempo superior a BOF. Intervalo de BOF mais restritos forçam o sistema a atualizar o conjunto de busca mais frequentemente, da mesma forma que BOF mais amplos permitem o sistema atualizar o conjunto de busca em intervalos mais espaçados.*

Para o trabalho [49], percebemos diretamente que BOF mais restritos impactariam em combinações de tabelas mais frequentes, acarretando aumento no tempo de resposta

das consultas, já que frequentemente estariam esperando a atualização da *Static Table* se concretizar.

2.3.3.1 Gestão de Fluxos Concorrentes

Fluxo de Consulta e Fluxo de Inserções podem variar a intensidade ao longo do tempo de vida da aplicação em cenários online. De forma geral, em alguns momentos os fluxos de dados podem estar mais ou menos intensos, o que nos leva a observar que configurações estáticas de paralelismo para tratar esse aspecto dinâmico proporciona um motor de busca com recursos computacionais sub ou superdimensionado. A gestão dinâmica de recursos e adaptação às diferentes intensidades do volume de dados é um aspecto pouco explorado no contexto de ANNS.

Pontualmente, no trabalho Hypercurves [51] é apresentados algumas estratégias dinâmicas para adaptar o paralelismo e a alocação de *threads* em núcleos de processamento CPU-GPU para acelerar busca aproximada em aplicações online com fluxo variável de consultas. Com essa adaptação dinâmica é alcançado uma redução de até 74% no tempo de resposta das consultas, quando comparado com uma abordagem de alocação de recursos estática. Indo além, no trabalho [4], discute-se uma abordagem para alocação dinâmica de *threads* e também a granularidade de tarefas em uma aplicação de busca aproximada por meio de IVFADC. Por sua vez, no trabalho [49] foi proposto o uso de uma janela deslizante para limitar a concorrência entre os fluxos de consulta e inserções em nós de computação específicos. Entretanto não é discutido uma adaptação à variação dinâmica da intensidade desses fluxos ao longo do ciclo de vida da aplicação.

Apesar de não relacionados ao contexto de ANNS, outros trabalhos [9, 17, 54] propõem mecanismos para a adaptação dinâmica de recursos computacionais em um nó de computação tendo como foco a variação da intensidade de fluxos de dados de entrada. Esse contexto de pesquisa, conhecido como *Parallelism tuning* tem agregado importantes contribuições para diferentes nichos de aplicações com interesse em computação de alta performance para objetivos específicos. No trabalho [17], os autores propõem o uso de *threads* auxiliares para decisão de qual o número ideal *threads* CPUs devem ser usadas dado as características da tarefa que está sendo executada. Essas *threads* auxiliares, em paralelo à execução da aplicação, tentam maximizar uma função objetivo dado os recursos CPUs disponíveis e as características da tarefa. Similarmente, no trabalho [9] são propostas algumas estratégias de escalonamento para adaptação dinâmica do nível de paralelismo entre *loops* aninhados. Indo além, o trabalho [54] apresenta mecanismos para a escolha da melhor configuração de paralelismo por meio de técnicas de *machine*

learning.

Essas contribuições apontam que a adaptação dinâmica de recursos computacionais em cenários onde a intensidade do volume de dado de entrada varia, beneficia consideravelmente a performance dos sistemas. Os trabalhos no contexto de ANNS em aplicações com fluxos intensos e variáveis exploram superficialmente essas oportunidades de otimização adaptativa.

2.3.3.2 Desafios e Oportunidades

Nessa seção foram apresentadas algumas soluções paralelas e em memória distribuída que lidam com características mais desafiadoras do cenários de ANNS em aplicações online. Nesse contexto, discutimos o trabalho estado da arte, o qual propõe soluções para gerenciar os fluxos concorrentes de consulta e inserções e também maneiras para o descarte de *buckets* obsoletos. Além de outros trabalhos que se beneficiam da adaptação dinâmica de recursos computacionais para acelerar aplicações com fluxos de dados variáveis.

Existem alguns aspectos nessas abordagens que trazem oportunidades de otimizações interessantes:

1. O trabalho [49], estado da arte na proposta de solução distribuída para ANNs em bases dinâmicas, se beneficia de um particionamento DES dos dados, o que, conforme discutido na Subseção 2.3.2.3, limita a escalabilidade da aplicação.

2. Ainda relacionado ao trabalho [49], o alto custo para a combinação de novos descritores (*Delta Tables*) na base de consulta (*Static Table*) inviabiliza aplicações onde o parâmetro BOF é restrito. Atualizações constantes guiam a solução para um aumento no tempo de resposta das consultas, já que frequentemente teriam que esperar a conclusão do processo de combinação das tabelas.

3. As soluções paralelas e distribuídas para ANNS apresentam configurações estáticas de paralelismo nos nós de computação. Conforme discutido, esse aspecto sub ou superdimensiona o motor de busca ao passo que a intensidade dos fluxos de dados variam ao longo do tempo de vida da aplicação.

Esses desafios incorporam as motivações das contribuições deste trabalho de tese, que serão detalhadas no Capítulo 4.

2.4 Sumário

Neste capítulo foram apresentadas as definições formais, os principais componentes, desafios e soluções que compõe o problema da Busca kNN Aproximada em grandes volumes de dados multimídia.

Tendo em vista o cenário desafiador de aplicações online CBMR, foram abordados contextos de aplicações distintos. No primeiro grupo, as aplicações lidam com a busca aproximada em conjuntos de objetos estáticos, e no tópico 2.3.2.3, foi pontuado problemas nas soluções estado da arte relacionados à organização espacial dos objetos da base, bem como balanceamento de carga entre os nós de computação envolvidos na solução paralela. Por sua vez, o segundo grupo é formado por aplicações que contemplam um cenário mais desafiador onde a base de objetos de busca cresce por meio da indexação de novos objetos. Nesse contexto, durante o tópico 2.3.3.2, foram apontados problemas relacionados à gestão dos fluxos concorrentes de consultas e inserções, o descarte de *buckets* obsoletos, bem como a aplicação subótima dos recursos computacionais envolvidos no motor de busca.

Assim, nos Capítulos 3 e 4 serão expostos, para cada um dos aspectos levantados nas subseções 2.3.2.3 e 2.3.3.2, os mecanismos propostos por esse trabalho para avançar o estado da arte nessa área de pesquisa.

Capítulo 3

ANNS em Memória Distribuída e *Buckets*

No Capítulo 2, foram abordadas soluções paralelas e em memória distribuída para ANNS, examinando trabalhos de referência na literatura. Essas abordagens englobam uma variedade de características, que vão desde a maneira como o conjunto de objetos é particionado até métodos para gerenciar fluxos de consultas e inserções em contextos desafiadores de aplicações CBMR.

Considerando esse amplo conjunto de trabalhos relacionados e, dentro desse contexto, levando em conta os desafios destacados nas subseções 2.3.2.3 e 2.3.3.2, este capítulo focará na arquitetura em memória distribuída proposta. A Seção 3.1 fornecerá detalhes sobre essa arquitetura, que tem como objetivo: i) suportar a implementação de diversas abordagens ANNS que organizam os dados em *buckets*; ii) permitir a personalização da estratégia de particionamento dos dados; iii) oferecer uma gestão adaptativa para múltiplos fluxos de dados.

Em seguida, na Seção 3.2, serão exploradas as possibilidades de otimização do modelo paralelo proposto, por meio da descrição de diferentes estratégias de particionamento dos dados entre os *Workers*. Entre as estratégias abordadas, estão as abordagens presentes na literatura, como DES e BES, bem como as contribuições deste trabalho de tese, SABES e SABES++. Estas últimas aproveitam a proximidade espacial entre os *buckets* para otimizar o particionamento dos dados, maximizando a utilização dos recursos paralelos e a escalabilidade do sistema.

Por fim, na Seção 3.3, será conduzida uma avaliação experimental que compara as diferentes abordagens de particionamento apresentadas. Nessa avaliação, serão discutidos os aspectos que destacam as contribuições deste trabalho, ressaltando os pontos que diferenciam essas abordagens e enfatizando seus resultados.

3.1 Paralelização em Memória Distribuída

Os detalhes de design e implementação da arquitetura em memória distribuída proposta por este trabalho para contemplar os desafios de ANNS em grandes volumes de dados são apresentada nesta seção. Como discutido anteriormente, o objetivo principal é minimizar o tempo de execução de consultas individuais, para que as métricas de sistema (tempo de resposta e vazão) sejam otimizadas globalmente. Para isso, foi projetado uma paralelização em memória distribuída por meio do *framework* Message Passing Interface (MPI) [12] e que mantém, em cada nó de computação, *buckets* em memória para acesso rápido. Além disso, a arquitetura foi projetado para permitir a implantação rápida de diferentes estratégias de particionamento de dados e, para lidar com os fluxos concorrentes de inserções e consultas, a arquitetura contempla unidades de controle para adaptação em tempo de execução de recursos computacionais. Por fim, é importante destacar que para a implementação e testes usamos a estratégia ANNS IVFADC. Entretanto, a arquitetura desenvolvida foi construída para suportar a implementação de quaisquer algoritmos de ANNS que se baseiem na organização do conjunto de objetos em *buckets*.

De forma geral, podemos decompor a solução nas fases de construção dos *buckets* e execução da busca, que são implementadas usando cinco tipos de nós de processamento: (i) **Leitores**: responsáveis por ler o conjunto de objetos e distribuí-lo entre os nós de processamento; (ii) **Coordenadores (Co)**: responsáveis por receber o fluxo de consultas e encaminhá-los para um conjunto específico de Processadores de Consulta (QPs) e, também, para computar o conjunto global de vizinhos mais próximos; (iii) **Produtores de Consulta (QS)**: responsável por gerar um fluxo de consulta constante e com intensidade variável; (iv) **Produtores de Inserções (IS)**: responsável por gerar um fluxo de novos objetos a serem indexados; (v) **Processadores de consulta (QPs)**: responsáveis por manter a indexação local dos *buckets* e executar a consulta em sua partição local de objetos.

A *Fase de construção dos buckets* tem dois passos principais. O primeiro é responsável por configurar a estrutura de dados usada pela estratégia ANNS para armazenar os *buckets*. Esse passo é particular de cada algoritmo ANNS, já que cada um deles se beneficia de diferentes formas de indexação dos objetos. Para o LSH, nesse passo, são feitas as configurações iniciais das Tabelas Hash, da mesma forma que para FLANN a estrutura das árvores de busca são instanciadas. Em muitos casos, um conjunto de objetos representativos, que aqui chamaremos de conjunto de treino, é usado para encontrar a configuração mais adequada da estrutura de dados para o contexto da aplicação em foco.

Neste trabalho, conforme mencionado, foi usado a estratégia IVFADC como base de implementação e testes. Portanto, na etapa de configuração, um conjunto de *Centroides Representantes* (C_c) é usado para definir a quantidade e os índices das listas no

arquivo invertido que indexará os *buckets*. Durante esta fase, um único nó de computação executa o algoritmo *k-means* tendo como entrada o conjunto de dados de treino, criando como resultado o conjunto de centroides representativos ou o *Codebook*. Essa fase só é executada uma vez para um determinado conjunto de dados, porque C_c pode ser persistida para reutilização em execuções futuras.

Posteriormente, o segundo passo é a distribuição do conjunto inicial de objetos. Esse passo é comum entre as diferentes estratégias de ANNS, em que cada Processador de Consulta (QP) irá receber uma partição do conjunto de objetos e irá indexá-los em sua estrutura de dados local. Para essa distribuição inicial, processos Leitores carregam os objetos da base e os enviam para os processos QPs respeitando a estratégia de distribuição de dados escolhida. É importante destacar que, para a abordagem IVFADC, visando otimizar os custos de tráfego de dados pela rede, os processos Leitores quantizam os objetos antes de enviá-los aos QPs.

A função *sendTo* implementada no processo Leitor (bloco amarelo na Figura 3.1) é responsável por escolher o(s) processo(s) QP para os quais os objetos serão enviados. Este método é um dos componentes propostos para serem customizados para implementar diferentes estratégias de particionamento do conjunto de objetos. As diferentes propostas de distribuição dos dados, bem como detalhes de implementação e testes, são discutidos na Seção 3.2. Como linha de base, assumimos que a abordagem round-robin é usada pelo método *sendTo* nessa fase de distribuição de dados. Consequentemente, os descritores de entrada são divididos igualmente entre QPs (estratégia DES). A Figura 3.1 fornece uma visão geral dessa fase e os componentes envolvidos.

A *fase de busca* é formada por dois contextos de execuções que acontecem em paralelo. Um deles está relacionado ao Fluxo de Consultas, em que a busca pelos objetos mais similares a um objeto alvo é realizada. Para esse processo, o sistema usa os seguintes processos: (i) **Produtor de Consultas (QS)**, (ii) **Coordenadores (Co)** e (iii) **Processador de Consultas (QP)**. Os processos QSs enviam consultas ao sistema, que são recebidas pelos Coordenadores. O Co destinatário irá encaminhar a consulta aos QPs responsáveis por processá-la, e esperará até que a busca local seja finalizada e os QPs retornem resultados para serem agregados em uma resposta global. Essa comunicação é realizada de forma assíncrona, sendo que o Co poderá executar outras funções enquanto o resultado local não é recebido.

O conjunto de QPs que irão receber uma consulta é definido pela operação *forward* (componente em amarelo no processo Coordenador representado na Figura 3.2). Esse método é a segunda função que deve ser personalizada para implementar diferentes estratégias de particionamento dos dados.

Como exemplo - e seguindo nossa linha de base -, a distribuição de objetos aconteceu por meio do particionamento DES, em que foi implementada a abordagem round-robin no método *sendTo*. Dessa maneira, cada consulta durante a fase de busca deve ser en-

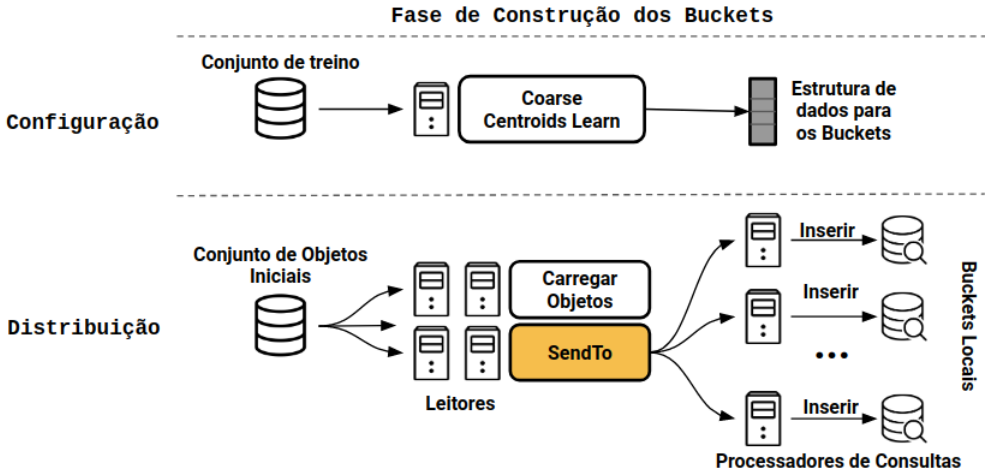


Figura 3.1: Nesta primeira fase proposta pela solução em memória distribuída, os objetos da base são distribuídos para os Processadores de Consulta (QPs). Essa fase acontece em dois passos. O primeiro deles configura a estrutura de dados particular da estratégia ANNS escolhida. Nesse passo, os *buckets* são implementados e, em alguns casos, um conjunto de objetos de treino precisa ser usado para personalizar as configurações. A estratégia IVFADC treina ou carrega o conjunto de centroides representativos (Codebook). Em sequência, o segundo passo é o processo de distribuição de descritores, que segue a estratégia de particionamento implementada no método *sendTo*.

viada para todos os QPs pelo método *forward* implementando uma estratégia *broadcast*. Intuitivamente, já que *buckets* foram igualmente particionados entre QPs, cada entrada nos índices locais devem ser visitadas durante a busca local.

Para concluir, o outro contexto de execução está relacionado ao Fluxo de Inserções, onde novos objetos devem ser indexados localmente para que os QPs componham a base de busca. Nesse processo, o componente Produtor de Inserções (QI), por meio do método *sendTo*, encaminha novos descritores para os componentes QPs. Em cada QP, ao receber um novo descritor vindo do Fluxo de Inserções, deverá ser indexados localmente para compor o conjunto de objetos de busca. Nesse contexto de execuções, alguns mecanismos são necessários para que os processos QPs sejam capazes de lidar de maneira eficiente com a gestão concorrente de consultas e inserções, bem como a gestão do conjunto crescente de objetos. Esses mecanismos serão detalhados no Capítulo 4. Os processos QSs vão distribuir os novos objetos entre QPs seguindo a implementação do método *sendTo* que, na abordagem linha de base (DES), implementa um mecanismo *round-robin*. A Figura 3.1 mostra os contextos de execuções durante essa fase do sistema proposto, bem como a interação entre os componentes usados.

Os coordenadores e Processadores de Consulta são implementados como processos multithread por meio da biblioteca Pthreads (conforme ilustrado na Figura 3.3). O Coordenador usa três threads: (i) **Thread de Recepção**: responsável por receber consultas e armazená-las em uma fila de tarefas (fila de consulta); (ii) **Thread de Distribuição**:

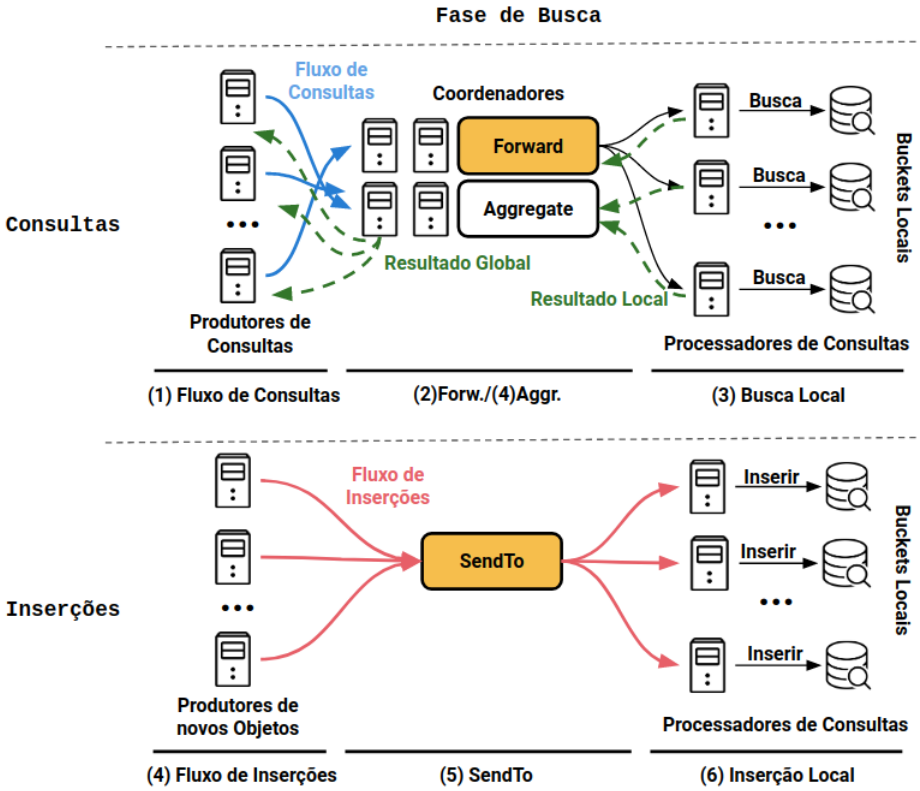


Figura 3.2: A fase de busca apresenta dois contextos de execução que acontecem em paralelo. (i) Um deles se relaciona com o Fluxo de Consultas e começa com o processo Produtor de Consultas (QS) enviando consultas para os processos Coordenadores, que serão responsáveis por encaminhá-las aos QPs designados para executar a busca local. QPs performam a estratégia ANNS localmente em seus índices e enviam os k objetos locais mais similares ao Coordenador para agregar em um conjunto de k objetos globais mais similares à consulta. (ii) O segundo está relacionado ao Fluxo de Inserções e acontece quando Produtores de Inserções (IS) encaminham novos descritores para os QPs indexarem e comporem a base de objetos locais para busca. A maneira como os objetos são particionados entre QPs afetarão a dinâmica de envio e encaminhamento de consultas e novos objetos durante o processamento de consultas e inserções, respectivamente. Essa maneira pode ser personalizada (*sendTo* e *forward*) implementando diferentes estratégias de distribuição de dados. Esse aspecto será estudado na Seção 3.2.

responsável por recuperar uma consulta da fila de tarefas e executar a operação *forward* para encontrar o conjunto de QPs que devem cooperar para responder a essa consulta e enviá-la a eles, e (iii) **Thread de Agregação**: responsável por receber os k-vizinhos mais próximos locais (resultado do processo de busca local em cada QPs) e agregá-los em uma resposta global.

Por sua vez, os QPs são implementados com os seguintes componentes: (i) **Thread de Recepção**: responsável por receber objetos para consultas encaminhadas pelos Coordenadores e também novos objetos provenientes do Fluxo de Inserções. A Thread de Recepção transformará o objeto recebido em uma Tarefa de Consulta ($Task_c$) ou Tarefa de Inserção ($Task_i$) respeitando a natureza do objeto. Ambos tipos de tarefas são

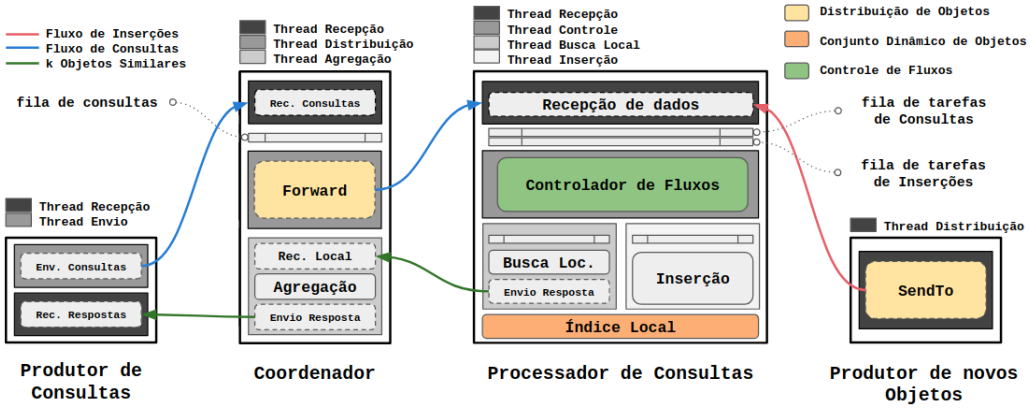


Figura 3.3: Coordenadores e Processadores de Consulta são implementados como processos multithread usando a biblioteca Pthreads. O esquema de threads é implementado por meio de uma abordagem consumidor/produtor. Esta estratégia é muito importante para não criar gargalos devido ao fluxo constante de consultas recebidas pelos Coordenadores e encaminhado para Processadores de Consultas, bem como o fluxo constante de novos objetos.

armazenadas em filas isoladas (fila de Tarefas de Consultas e fila de tarefas de Inserções). (ii) **Thread para Controle de Fluxo**: responsável por executar o componente Controlador de fluxos em que é definida dinamicamente a alocação de recursos computacionais (Threads para Busca Local e Threads para Inserções) que garantirão a execução eficiente de Tarefas de Consultas e Tarefas de Inserções. (iii) **Threads para Busca Local**: responsável por recuperar uma Tarefa de Consulta da respectiva fila de tarefas e realizar a pesquisa ANNS no índice local. Esta busca é executada paralelamente usando nível de paralelismo interno definido pelo componente Controlador de fluxos. Podem haver mais de uma thread desse tipo executando paralelamente, também definido pelo Controlador de fluxo. Após a busca local ser finalizada, os resultados são enviados para o Co responsável por aquela consulta em particular. (iv) **Threads para Inserções**: responsável por recuperar uma Tarefa de Inserção da respectiva fila de tarefas e indexar o novo objeto recebido. Podem haver mais de uma thread desse tipo executando paralelamente, também definido pelo Controlador de fluxo.

Os detalhes da implementação do Controlador de Fluxos, bem como as estratégias proposta para a gestão eficiente da execução de Tarefas de Consultas e Tarefas de Inserções, são apresentado em detalhes no Capítulo 4.

3.2 Estratégias para Particionamento de Dados

As contribuições que serão discutidas nesta seção são motivadas pelas observações levantadas durante a Subseção 2.3.2.3, onde foi destacado que as soluções presentes na literatura se baseiam em um particionamento equilibrado do conjunto de objetos (DES) e essa abordagem incorre em limitações da escalabilidade da solução paralela. Além disto, os trabalhos relacionados não consideram a proximidade espacial entre *buckets* ao distribuir o conjunto de dados, sendo que *buckets* próximos tendem a ser acessados juntos frequentemente. Pensando nisso, neste trabalho de tese é proposto estratégias para o particionamento dos dados entre Processadores de Consultas (QPs) que exploram a organização inteligente dos *buckets* por meio da arquitetura de memória distribuída apresentada anteriormente e pela personalização de métodos para envio e encaminhamento de objetos, *sendTo* e *forward*: (i) *Spatial-Aware Bucket Equal Split* (SABES) na Subseção 3.2.3; (ii) *Spatial-Aware Bucket Equal Split* com Balanceamento de Dados (SABES++) na Subseção 3.2.4.

Além disto, é descrita na Subseção 3.2.1 a implementação da estratégia Data Equal Split (DES) e na Subseção 3.2.2 a abordagem Bucket Equal Split (BES) ambas encontradas na literatura e que serão usadas como linha de base para as avaliações experimentais. Por fim, na Subseção 3.2.5 a complexidade computacional das estratégias são estudadas.

3.2.1 Data Equal Split (DES)

A estratégia DES divide o conjunto de objetos de entrada igualmente entre os processos QP por meio de uma distribuição *round-robin* durante a fase de construção dos *buckets* (*sendTo*). Consequentemente, em cada QP haverá, no índice local, entradas para todos os *buckets* enquanto os objetos em cada um deles estão divididos entre os demais QPs. Dessa maneira, para garantir que os *buckets* de interesse da estratégia ANNS sejam integralmente visitados durante o processo de pesquisa, é preciso enviar a consulta para todos os QPs presentes na máquina de memória distribuída. Assim, o método *forward* implementa uma abordagem *broadcast*. Esta estratégia foi encontrado em vários trabalho anteriores [49, 39, 3].

Usando a abordagem IVFADC, que é base neste trabalho, cada QP tem uma estrutura IVF completa com entradas associadas a todos os centroides representativos (Cc). Ao enviar a consulta para todos os QPs, garante-se que as w listas associadas aos centroides mais próximos serão integralmente visitadas, já que os descritores nessas listas estão divididos entre vários QPs diferentes. A Figura 3.4 ilustra o processo descrito.

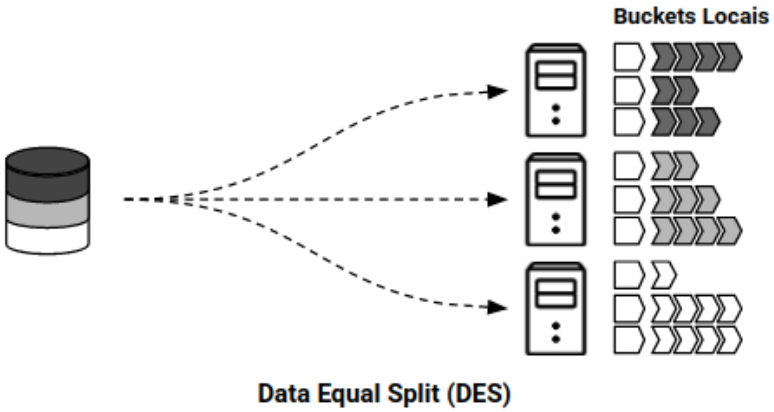


Figura 3.4: Distribuição DES do conjunto de objetos durante a paralelização da estratégia IVFADC. Nela, todos os QPs possuem entradas para todos os centroides representados, sendo que as listas associadas possuem descritores divididos entre vários QPs diferentes.

3.2.2 Bucket Equal Split (BES)

Conforme amplamente discutido no Capítulo 2, a maioria dos algoritmos de indexação para ANNS segue um padrão comum de organização de dados que consistem na construção de *buckets*. Pela definição 6, espera-se que os objetos armazenados em um mesmo *bucket* sejam próximos no espaço, e para acelerar o processo de busca, apenas alguns conjuntos de *buckets* são visitados durante o processo de pesquisa aproximada. Por causa dessa característica, uma outra abordagem intuitiva para particionar o conjunto de objetos entre QPs é dividir os *buckets* integralmente entre eles. Esta estratégia é chamado *Bucket Equal Split* e em cada QP, no índice local, haverá a mesma quantidade de entradas porém para diferentes *buckets*. Durante a busca, espera-se que com esta abordagem, apenas um subconjunto dos QPs terá que ser consultado para responder a uma consulta, sendo esses aqueles que possuem indexado os *buckets* de interesse da estratégia ANNS. Comparativamente com a abordagem DES, esta estratégia reduz o tráfego de comunicação e gastos relacionados à alocação de recursos para o início do processo de computação da busca local.

Assim, usando IVFADC, as entradas do arquivo invertido são distribuídas igualmente entre os QPs. Esta distribuição garante que todos os objetos em uma determinada entrada do IVF (um *bucket*) serão armazenados em conjunto em um único QP.

Uma vez que cada entrada de arquivo invertido tem um centroide representativo associado, esta estratégia pode ser simplificada ao particionamento de centroides e seus respectivos objetos mais próximos entre os QPs. Dado que $|C_c|$ centroides são instanciados, o $i' - \acute{e}simo$ QP (QP_i, $i \in 1 \ldots n$) armazena $|C_c|/n$ centroides. As implementações personalizadas de *sendTo* e *forward* são apresentadas no Algoritmo 3. A Figura 3.5 ilustra a organização proposta por BES.

Durante a fase de distribuição dos buckets e para o envio de novos objetos pelo

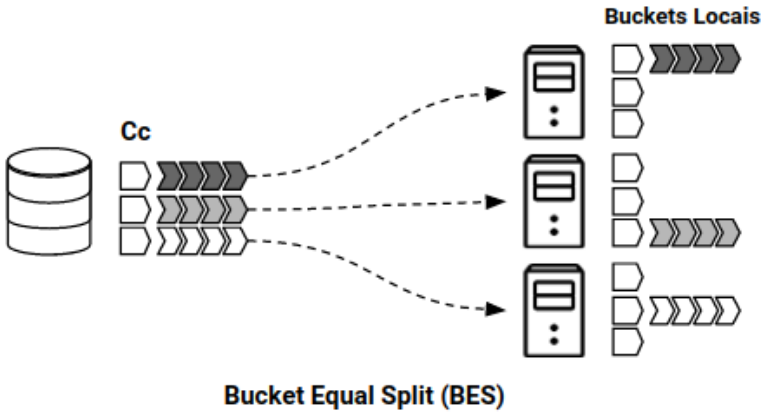


Figura 3.5: A abordagem BES propõe a distribuição igualitária dos *buckets* entre os QPs. Usando a estratégia IVFADC, BES particiona o conjunto de centroides igualmente entre os processadores de consultas. É esperado que dessa maneira, um conjunto menor de QP seja necessário durante o processo de busca em comparação com a solução DES.

Fluxo de Inserções, o *sendTo* recebe um objeto y para ser indexado e calcula seu centroide mais próximo k', que por sua vez, está associado a entrada da lista invertida na qual y deve ser armazenado (linha 2). O mapeamento do centroide para a instância QP é então calculada (linha 3) e retornada (linha 4).

Algorithm 3: Métodos SendTo e Forward para BES

```
1: function sendTo(y)
2:     k' ← q_c(y)
3:     target ← k'/n
4:     return target
5: function forward(x, w)
6:     nnc ← kNN(C_c, x, w)
7:     for i ∈ 1 ... w do
8:         target ← target ∪ ncc[i]/n
9:     return target
```

Na fase de busca, conforme discutido antes, em vez de enviar o vetor de consulta x para todos os QPs como é feito por DES, é preciso encontrar o conjunto de QPs que são responsáveis por armazenar as w listas invertidas (*buckets*) representados pelos centroides mais próximos da consulta x. Assim, conforme mostrado no Algoritmo 3, o método *forward* calcula os w centroides representativos mais próximos da consulta x (linha 6) e então encontra quais são as respectivas instâncias QP que armazenam as listas invertidas associadas à eles (linha 8). A abordagem BES tende a usar um número menor de QPs do que o DES, já que envia o objeto de consulta para no máximo $min(w, n)$, onde n é o número de instâncias QP, e w tende a ser muito menor do que n em grandes sistemas distribuídos.

3.2.3 Spatial-Aware Bucket Equal Split (SABES)

A abordagem SABES é uma contribuição deste trabalho de tese, que avança o estado da arte ao fornecer uma distribuição de tira proveito não só da organização espacial dos objetos, como feito por BES, mas como também da proximidade espacial entre os *buckets*.

Dessa forma, SABES estende BES propondo uma distribuição onde *buckets* próximos no espaço sejam atribuídos à um mesmo QP. A premissa para a construção da abordagem SABES é que *buckets* espacialmente próximos têm alta probabilidade de serem visitados em uma mesma consulta. Consequentemente, armazenar conjuntos de *buckets* próximos em um mesmo nó de processamento reduziria ainda mais o número de QPs usados para responder consultas.

Algorithm 4: Construção das Regiões SABES, métodos SendTo e Forward

```
1:  function regionConstruction(C_c, n)
2:      R ← k-means(C_c, n)
3:      return R
4:  function sendTo(y)
5:      k' ← q_c(y)
6:      target ← getRegion(R, k')
7:      return target
8:  function forward(x, w)
9:      nnc ← kNN(C_c, x, w)
10:     for i ∈ 1 ... w do
11:         target ← target ∪ getRegion(R, nnc[i])
12:     return target
```

O SABES tem uma etapa *off-line* adicional ou de pré-processamento em que os *buckets* precisam ser agrupados em n macro regiões R, sendo que cada macro região R_i será atribuída a um QP_i. Usando a abordagem IVFADC, essas regiões são calculadas usando o algoritmo $k-means$ tendo o conjunto de centroides representativos C_c como entrada, conforme implementado pelo método $regionConstruction(C_c, n)$ no Algoritmo 4. O retorno R é um um dicionário que associa cada centroide à macro região que ele foi agrupado. Esse dicionário será posteriormente usado nos métodos *sendTo* e *forward*. A macro região é em essência um aglomerado de centroides representativos. As operações *sendTo* e *forward* também são apresentados no Algoritmo 4 e possuem implementações semelhantes aos mesmo métodos presentes na abordagem BES. A principal mudança é a função *getRegion* consulta o dicionário onde é mapeado centroides à macro regiões R_i. Esta função simplesmente acessa o dicionário indexado pelo id do centroide e retorna sua respectiva região. A Figura 3.6 complementa o algoritmo discutido ao mostrar uma visualização do processo de distribuição proposto por SABES.

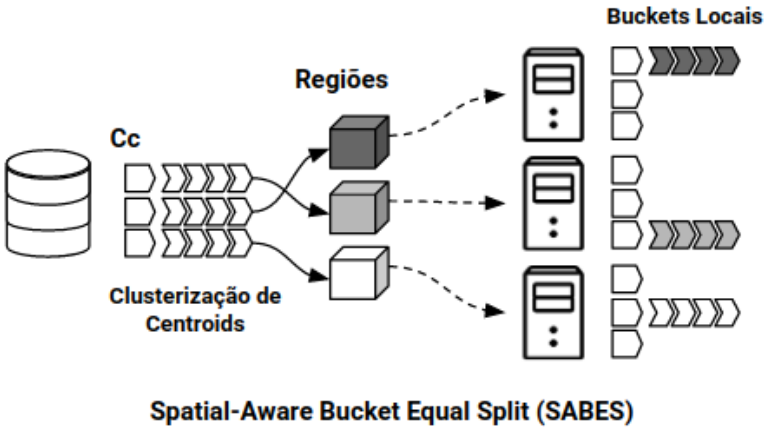


Figura 3.6: SABES é uma das abordagens para organização do conjunto de objetos contribuição deste trabalho de tese. A estratégia proposta tira proveito da proximidade espacial entre os *buckets* para alocá-los em um mesmo QP. A intuição é que *buckets* espacialmente próximos tendem a ter alta probabilidade de ser visitados em uma mesma consulta. Com essa organização espera-se uma redução ainda maior do conjunto de QPs necessários para uma mesma busca.

3.2.4 Spatial-Aware Bucket Equal Split Balanceado (SABES++)

A abordagem proposta SABES atribui em cada QP, um agrupamento de *buckets* que são próximos no espaço. Esses agrupamentos, nomeado de macro regiões, apresentam variada quantidade de *buckets*, que por sua vez, apresentam quantidades desiguais de objetos associados. Assim, intuitivamente podemos dizer que a quantidade de objetos presentes em cada macro região pode divergir significativamente, e consequentemente, ao associar essas regiões aos QPs, a estratégia SABES introduz desbalanceamento de carga entre os processadores de consulta da máquina em memória distribuída. Esse desbalanceamento de carga proporciona desequilíbrio entre os esforços de armazenamento e processamento da busca local entre as instâncias QP, podendo impactar diretamente no desempenho da solução ao criar gargalos de execução.

Assim, a fim de mitigar o desequilíbrio decorrente da distribuição feita por SABES, a abordagem SABES++ propõe um nova estratégia para organizar *buckets* em macro regiões. Nesta abordagem, os *buckets* são agrupados considerando sua localização espacial, assim como era feito por SABES, e também considerando o número de objetos associados a eles. Esse agrupamento, que balanceia a somatória de objetos em cada *buckets* presente nas macro regiões, é implementado por meio do algoritmo $k - means$ ponderado [33] que visa equilibrar os pesos (número de objetos em cada *buckets*), enquanto, ao mesmo tempo, preserva a proximidade espacial entre aqueles que estão no mesmo grupo.

O Algoritmo 5 apresenta o método *regionConstruction* implementado pela contri-

Algorithm 5: Construção das Regiões SABES++

```
1: function regionConstruction(C_c, n)
2:     R ← weighted-k-means(C_c, W, n)
3:     return R
```

buição SABES++ por meio da estratégia IVFADC. Esse método é executado off-line em uma fase de pré-processamento, e consistem da execução do $k-means$ ponderado (line 2) que calcula as os conjunto de centroides (macro regiões R). Esta estratégia mantém a proximidade espacial entre os centroides (*buckets*) alocados em cada QP, no entanto, balanceia, pelo melhor esforço, a quantidade de objetos armazenados em cada uma das instâncias. Os métodos *sendTo* e *forward* para a solução SABES++ são exatamente os mesmos usados por SABES, detalhados na seção anterior seção. A Figura 3.7 mostra os componentes dessa abordagem discutida.

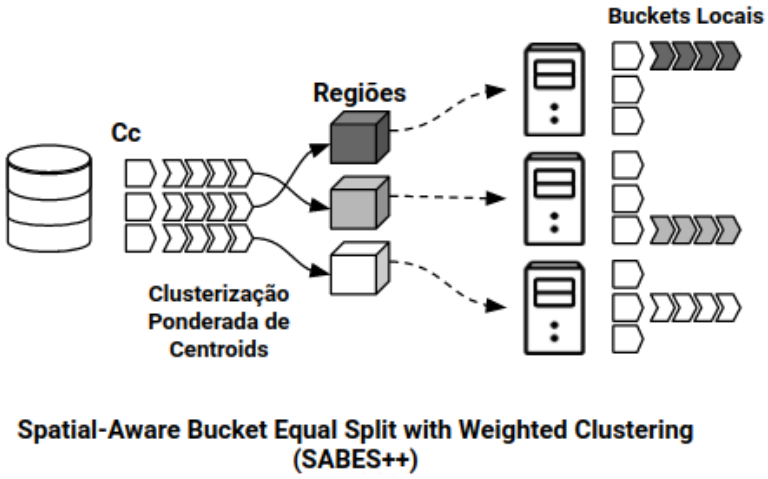


Figura 3.7: SABES++ estende a estratégia SABES ao introduzir um agrupamento dos *buckets* ponderando pela quantidade de objetos que estão associados a cada um dos grupos. O objeto é minimizar, pelo melhor esforço, o potencial desequilíbrio na carga de trabalho que a distribuição SABES possa incorrer.

3.2.5 Complexidade Computacional

Esta seção resume a complexidade das estratégias de particionamento de dados discutidas anteriormente. Dado um descritor de entrada (objeto de consulta ou objeto para ser indexado), serão detalhadas as complexidade dos métodos *sendTo* e *forward* associados à abordagem de distribuição.

Essencialmente, a complexidade das operações *sendTo* é a mesma para todas as estratégias. Ela é dada pelo custo associado ao processo de preparação do descritor para ser indexado e também pelo custo de se encontrar o *buckets* mais próximo responsável por armazenar esse descritor. Esses custos estão diretamente relacionados à estratégia ANNS instanciada. Com IVFADC, *sendTo* é dominado pelo custo de quantizar o descritor ($q_c(y)$), e após a quantização ser calculada, encontrar o centroides representativos mais próximos de y. O resto da decisão sobre para qual QP enviar é constante em todos casos. É importante destacar que até durante a abordagem DES, o descritor é quantizado antes de ser enviado para reduzir a largura de banda da rede.

Estratégia	**forward**	**# de Mensagens**
DES	O(1)	O(—QPs—)
BES	O(—Cc—*d+—Cc—*log w + O(w))	O(min(w,—QPs—))
SABES	O(—Cc—*d+—Cc—*log w + O(w))	O(min(w,—QPs—))
SABES++	O(—Cc—*d+—Cc—*log w + O(w))	O(min(w,—QPs—))

Tabela 3.1: Complexidade computacional das estratégias de particionamento dos dados.

Por sua vez, o método *forward* tem complexidades diferentes dado a organização espacial elaborada durante a distribuição dos objetos, entretanto ainda sim está associado à estrutura de indexação usada pela estratégia ANNS. Continuaremos a usar IVFADC como base para as comparações entre as complexidades.

A estratégia DES distribui o descritor de consulta a todos os QPs durante a etapa *forward*, portanto, o cálculo de destino nesse passo é $O(1)$ ao ponto que é preciso enviar $|QPs|$ mensagens. Já BES, SABES e SABES++ devem encontrar os w centroides representativos mais próximos do descritor de consulta, durante o o método *forward* a fim de enviar mensagens apenas para QPs que armazenam os *buckets* de interesse. Para encontrar o conjunto de w centroides mais próximos, o algoritmo $k-NN$ é aplicado no conjunto completo de centroides representativos, tendo custo de $O(|Cc|*D+|Cc|*logw)$. O restante do método *forward* consiste em mapear os w centroides mais próximos para os *ids* dos QPs que armazenam os objetos associados à eles. Isto é realizado em $O(1)$ para cada centroide, uma vez que um dicionário com esse mapeamento está disponível a partir de um passo offline de pré processamento, resultando em $O(w)$ para os w centroides mais próximos encontrados. Assim, a complexidade *forward* será $O(|Cc|*d+|Cc|*logw+O(w))$ para BES, SABES e SABES++. Adicionalmente, um limite de $min(w, |QPs|)$ mensagem serão enviadas para o conjunto de QPs.

Todas as complexidades estão sumarizadas na Tabela 3.1.

3.3 Avaliação Experimental

Nesta seção, avaliamos as estratégias e otimizações propostas para a distribuição do conjunto de objetos entre Processadores de Consulta. Tendo DES e BES como abordagens linha de base, o objetivo é avaliar sistematicamente se a organização inteligente dos objetos proposta por SABES e SABES++ alcançam significativos benefícios de desempenho ao reduzirem a quantidade de QPs necessários para o processo de uma consulta. Apesar da arquitetura proposta e as estratégias de distribuição descritas anteriormente serem agnósticas à abordagem ANNS, em nossos experimentos implementamos a estratégia IVFADC.

Dito isto, os testes estão configurados em um sistema com 160 nós de computação conectados por meio de um *FDR Infiniband switch*. Cada nó é equipado com 2 CPUs Intel Haswell E5-2695 v3, 128 GB de RAM e executa um sistema operacional Ubuntu servidor 20.04 OS. Além disso, usamos C++14, Intel MPI 3.1 e OpenMP 4.0 para as implementações.

O IVFADC foi configurado com 4.096 centroides representativos, $m = 8$ (número de subespaços de quantização), 256 centroides por subdimensão compondo o Codebook de quantização. O conjunto de dados usado tem até 16 bilhões de descritores SIFT de 128 dimensões e 10000 descritores para consultas. A qualidade dos resultados da pesquisa é avaliada usando a métrica *recall@R*, que corresponde a porcentagem ou fração dos resultados das consultas onde o descritor mais próximo está entre as primeiras R posições [39].

Nossa avaliação de escalabilidade da máquina em memória distribuída, apresentada na Subseção 3.3.4, emprega a abordagem de configurações ***weak scaling***, onde o conjunto de dados indexados aumenta proporcionalmente ao número de nós de computação. Essa abordagem é mais apropriada no contexto de aplicações deste trabalho de tese, no qual deve lidar com a característica de conjuntos de dados crescentes. A configuração do sistema distribuído, tem como linha de base: 1 Coordenador (Co) e 4 Processadores de Consulta (QPs), além de um conjunto de dados com 500 milhões de descritores SIFT. Aumentamos proporcionalmente o conjunto de objetos e os recursos computacionais até a configuração: 32 Coordenadores e 128 QPs, usando um conjunto de dados de 16 bilhões de descritores SIFT.

Cada instância do Coordenador e do Processador de Consulta é executado exclusivamente em um nó de computação. Todos os experimentos foram repetidos 3 vezes e observamos pequenas variações no tempo de execução com desvio padrão menor que 1,6%. Os resultados são apresentado como a média dessas 3 execuções.

3.3.1 Eficiência das Estratégias de Particionamento de Dados

Esta subseção avalia diferentes execuções do processo de busca por meio do IVFADC implementado na arquitetura proposta, que diferem pelas estratégias de particionamento do conjunto de dados instanciadas.

A avaliação realizada neste momento usa 40 nós de computação: 8 coordenadores e 32 Processadores de Consulta, a fim de focar no impacto de desempenho da estratégia de particionamento empregada. Uma avaliação com um número maior de nós é apresentado e discutido na Seção 3.3.4. Além disso, o conjunto de dados usado apresenta 4 bilhões de descritores SIFT e 10000 consultas foram executadas, enquanto IVFADC foi configurado com $C_c = 4096$ (centroides representativos) e $w = 16$ (número de listas a serem pesquisadas), mantendo recall@R=100 de $63,7\%$.

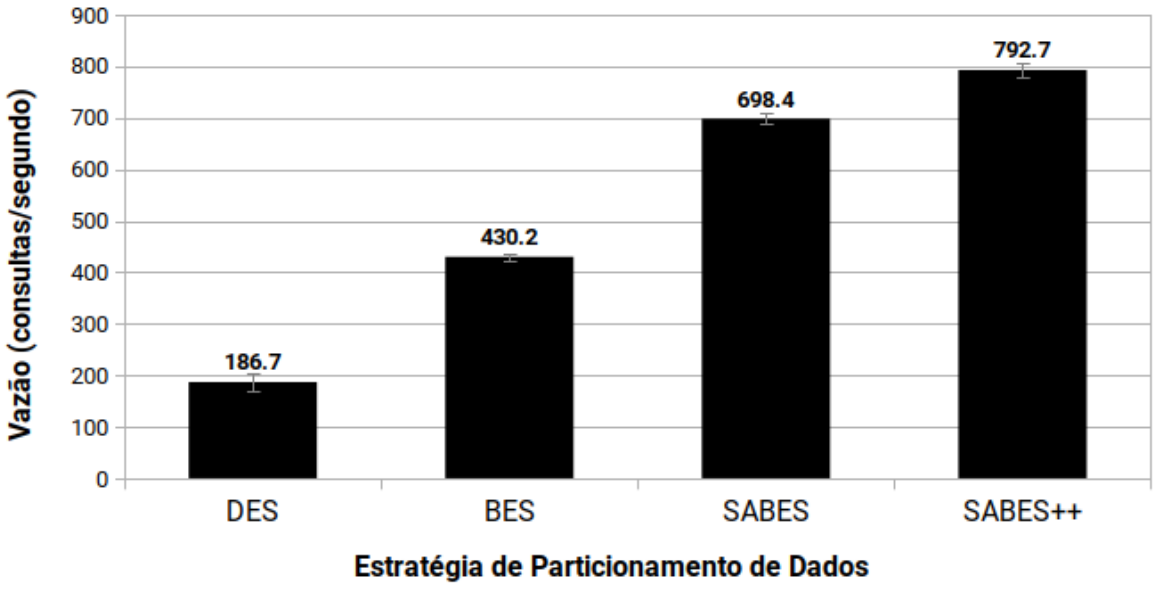


Figura 3.8: O desempenho da estratégia IVFADC (consultas/segundo) variando a abordagem de particionamento de dados em um ambiente distribuído com 40 nós de computação.

Os resultados experimentais são mostrados na Figura 3.8 onde foram comparadas as diferentes estratégias de distribuição. Em primeiro lugar, destaca-se que a estratégia DES é a menos eficiente dentre elas. BES, SABES e SABES++ alcançaram acelerações de, respectivamente, $2,3\times$, $3,7\times$ e $4,2\times$ em comparação à DES.

As melhorias de desempenho alcançadas por BES, SABES e SABES++ estão diretamente relacionadas a uma redução do conjunto de QPs necessários para responder a uma consulta. A abordagem DES precisa encaminhar cada consulta pada todos os QPs que compõem a máquina de busca, enquanto as demais estratégias, por particionar objetos distribuindo integralmente listas do arquivo invertido (*buckets*) entre QPs, precisam

encaminhar a consulta apenas para aqueles que armazenam pelo menos uma das lista de arquivos invertida a ser pesquisada. Para validar esse argumento, a Tabela 3.2 mostra o número de QPs usados por consulta (QPs/Consulta) para cada uma das estratégias. DES usa todos os 32 QPs, como esperado, ao passo que BES e SABES precisam em média de apenas, 12,7 e 4,3 QPs, respectivamente, para responder a uma consulta. Essa característica consolida a afirmação de que ao distribuir *buckets* próximos no espaço em um mesmo Processador de Consulta, conforme proposto por SABES e SABES++, estamos aumentando a localidade de busca, que acontecerá em apenas um subconjunto de QPs.

Indo além, pode-se observar que a abordagem SABES++ não reduz o número de QPs usados em comparação a SABES. Sua melhoria de desempenho é uma consequência de uma distribuição de *buckets* mais elaborada, na qual reduz o desequilíbrio de carga entre os QPs, eliminando potenciais gargalos de execução. Esse aspecto é discutido em detalhes na Subseção 3.3.3.

Estratégia	**QP/Consulta**	**Busca Local**	**Tempo de Busca Total**	**Vazão**
DES	32	10000	1344.8	186.7
BES	12.7	3995.9	586.6	430.2
SABES	4.3	1349.8	225.6	698.4
SABES++	4.4	1382.7	230.4	792.7

Tabela 3.2: Comparação do número de QP usados por consulta entre as estratégias de busca evidenciando o tempo de busca total e a vazão de tarefas executadas.

O número de QPs usados para responder a uma consulta impacta as demandas de comunicação e custos computacionais associados ao processo de receber e processar mensagens de entrada em um nó de computação. Além disso, um menor número de QPs necessários no processamento de uma busca está diretamente relacionado a uma pesquisa que visita localmente maior quantidade de descritores. Essa característica torna o processamento local da estratégia ANNS mais eficaz pelo fato da densidade computacional ser maior em cada máquina. Na Tabela 3.2, esse aspecto pode ser visto por meio da coluna *Tempo Total de Pesquisa*, na qual apresenta a soma do tempo exigido por todos os QPs para realizar o processo de busca local.

Com essas observações é interessante evidenciar que, na abordagem DES, cada *bucket* tem seus objetos divididos entre as máquinas, assim a pesquisa precisa acontecer em vários pequenos *buckets* locais. A pesquisar em poucos *buckets* maiores em comparação com a pesquisa em vários *buckets* menores é mais eficiente porque: (i) mais paralelismo está disponível e os núcleos da CPU estão efetivamente usado (densidade computacional maior); (ii) a fase $k - select$, onde os objetos mais próximos são recuperados em cada *bucket* local é executado menos vezes (um por *bucket* na abordagem DES vs. um em cada QP que armazena um *bucket* integralmente). Esses aspectos melhoram a eficiência geral da solução paralela proposta.

Tabela 3.3: Distribuição de dados usando SABES e SABES++. É apresentado a quantidade mínimo e máximo de objetos (descritores) atribuídos a um QP e o desvio padrão entre todos os 32 QPs.

		Distribuição de Objetos		
Estratégia	**# of QPs**	**min**	**max**	**std**
SABES	4	167078445	336612032	89578841
	32	14249858	62618917	12678393
SABES++	4	160689301	390357008	75624558
	32	21451654	46859624	4549251

3.3.2 Impacto do Balanceamento de Cargar

A Figura 3.8 mostrou que SABES++ é a abordagem mais rápida, alcançando uma aceleração de $1,13\times$ em relação a SABES. Além disso, a Tabela 3.2 mostra que SABES e SABES++ visitam uma quantidade semelhante de QPs para responder a uma consulta. Entretanto, os ganhos com SABES++ veem de sua habilidade em equilibrar a carga de trabalho (número de descritores) entre QPs, mantendo a característica da alocação por proximidade espacial conforme proposto por SABES. O balanceamento de dados em SABES++ é apresentado na Tabela 3.3 que apresenta o máximo e o mínimo da quantidade de descritores indexados por QPs e m ambas abordagens. Nota-se uma redução significativa no desequilíbrio de dados com SABES++, que reduz o *makespan*(espaço de tempo entre o inicio e o fim da tarefa) geral dessa estratégia.

3.3.3 Impacto da Quantidade de Centroides

Indo além nas avaliações, observamos o desempenho relativo das abordagens de particionamento de dados para diferentes quantidades de centroides representativos (C_c) ou número de IVF listas (*buckets*). Neste caso, variamos o número a quantidade de centroides representativos e medimos o desempenho de todas as estratégias, seguindo as orientações dos testes anteriores. Nesta subseção, estamos interessados em entender se o número de centroides impactaria o desempenho relativo das estratégias de particionamento de dados. Os resultados são apresentados na Figura 3.9. Como pode ser observado, o desempenho das estratégias propostas (relativo a DES) não muda significativamente quando o número de centroides usados varia.

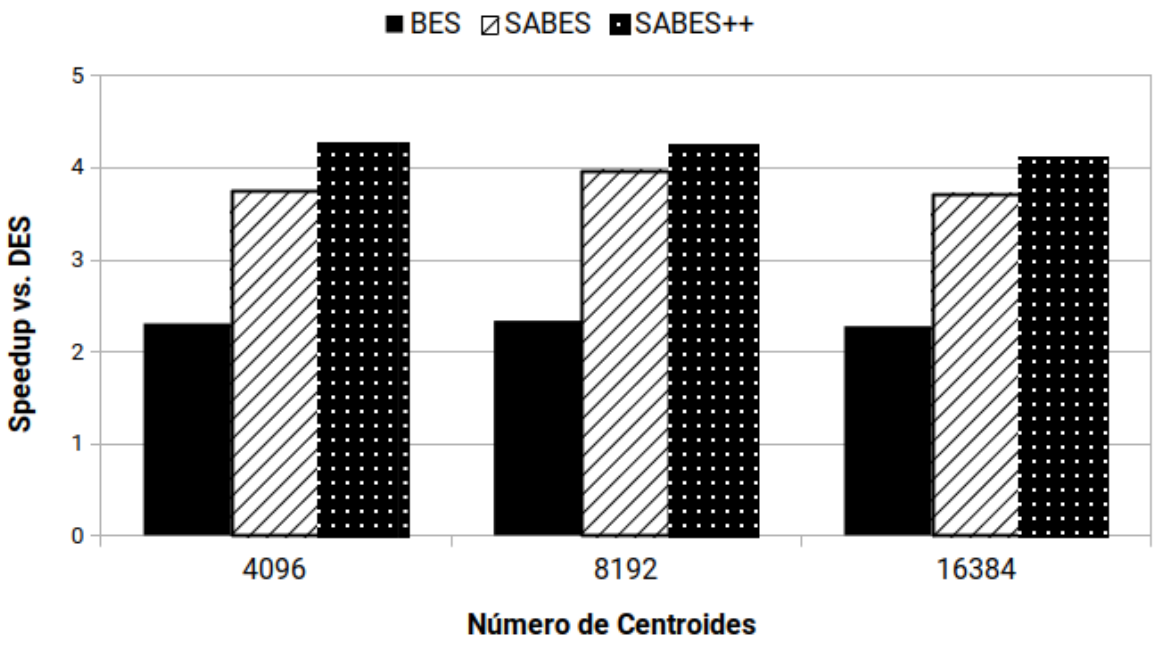


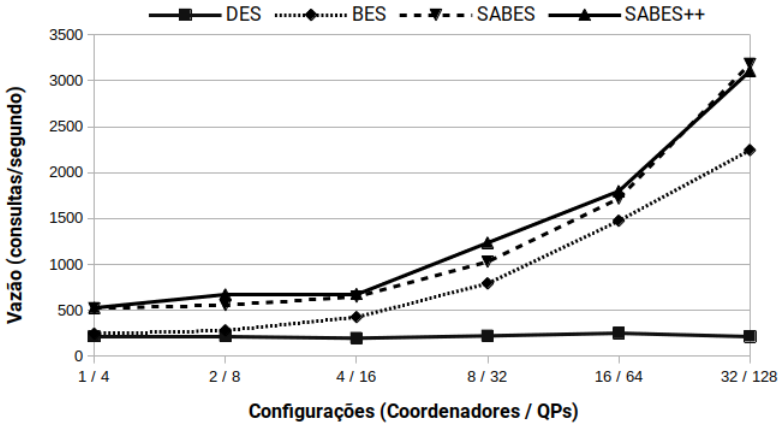
Figura 3.9: Acelerações obtidas pelas diferentes estratégias de particionamento de dados em relação a DES, conforme o número de centroides usados varia.

3.3.4 Escalabilidade da Solução Paralela e Distribuída

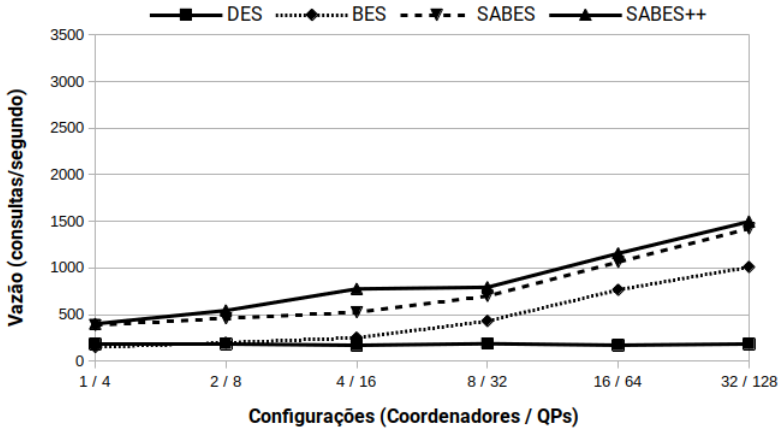
Esta subseção avalia a escalabilidade da solução paralela para o IVFADC por meio das diferentes estratégias de particionamento de dados variando a quantidade de nós de computação. O experimento consiste em aplicar a abordagem *weak scaling*, em que o tamanho do conjunto de dados e o número de nós de computação usados aumentam proporcionalmente. Conforme discutido anteriormente, as configurações variam entre (i) 1 Coordenador e 4 QPs em um conjunto de dados de 500 milhões de descritores SIFT até (ii) 32 Coordenadores e 128 QPs (total de 160 nós) em um conjunto de 16 bilhões de descritores SIFT. Este experimento de escalabilidade foi repetido variando a quantidade de listas a serem visitadas durante a busca (w).

A Figura 3.10 mostra os valores de vazão (consultas/segundo) para cada uma das configurações avaliadas durante o teste de escalabilidade. Primeiro é importante notar que a abordagem DES alcançou um desempenho de escalabilidade próximo ao linear, enquanto as propostas BES, SABES e SABES++ alcançaram escalabilidade superlinear. Consequentemente, os ganhos de desempenho dessas estratégias em relação à estratégia DES, são ainda mais expressivos com o aumento da quantidade de nós de computação.

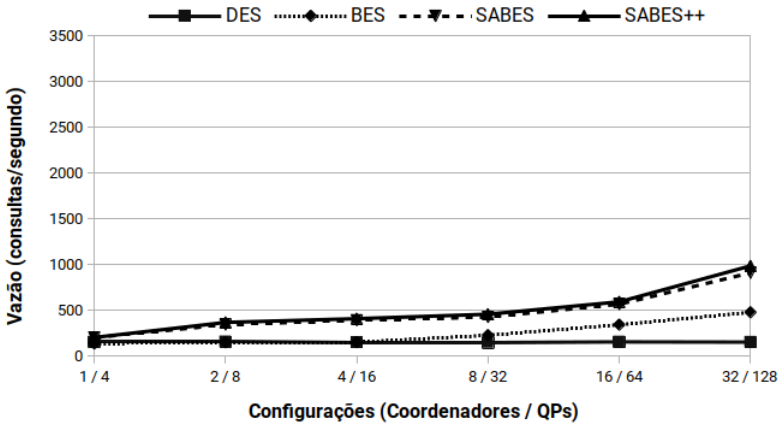
A escalabilidade superlinear do BES, SABES e SABES++ é um reflexo da redução considerável da quantidade de QPs necessários para responder uma consulta, envolvendo



(a) Escalabilidade usando w=8



(b) Escalabilidade usando w=16



(c) Escalabilidade usando w=32

Figura 3.10: Desempenho da abordagem IVFADC (consultas / s) em uma avaliação *weak scaling*, onde o número de nós (coordenadores e QPs) e o tamanho do conjunto de dados aumentam proporcionalmente. A configuração da linha de base é composta por 5 nós de computação (1 Coordenador e 4 QPs) usando um conjunto de 500 milhões de descritores SIFT. A avaliação de escalabilidade é repetido para diferentes valores de w.

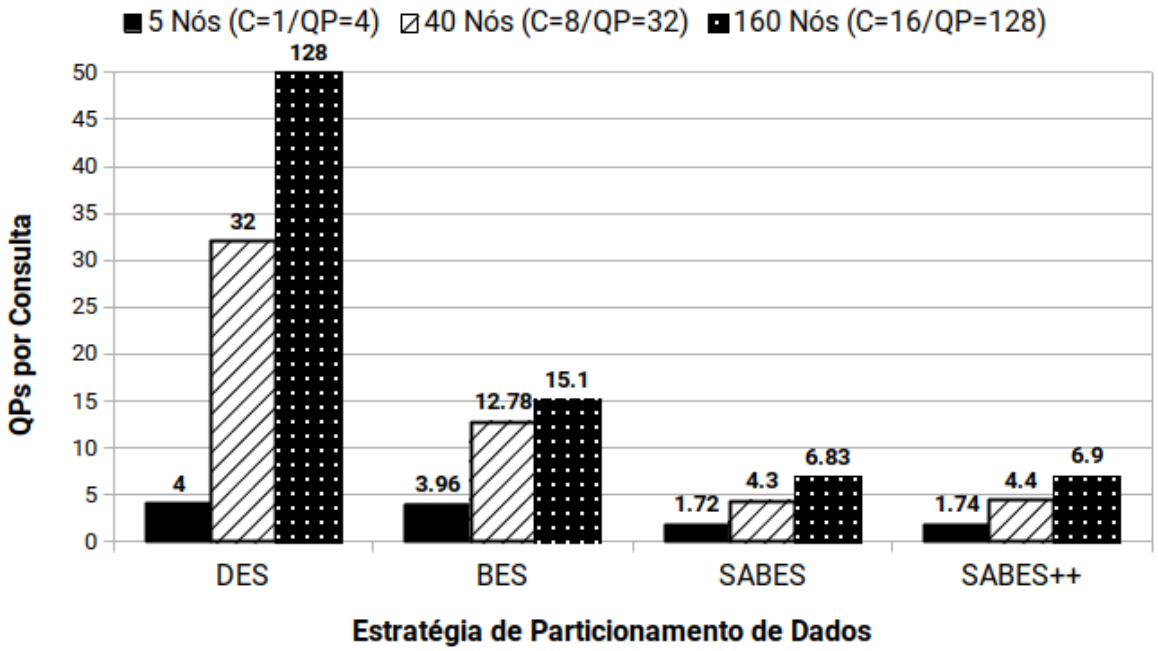


Figura 3.11: Número médio de QPs usados para responder a uma consulta para diferentes estratégias de particionamento de dados e número de nós de computação aplicados na solução paralela.

não mais que w QPs no pior caso. Consequentemente o tempo para processar a pesquisa localmente (aspecto discutido na subseção anterior seção) é menor. Além disso, a Figura 3.11 mostra que o número de QPs usados para responder a uma consulta crescem sublinearmente em relação ao número de nós para BES, SABES e SABES++. Por exemplo, SABES usa em média $4,30$ e $6,83$ QPs para responder a uma consulta em configurações onde 40 e 160 nós forem empregados, respectivamente. Nesse caso, os recursos de computação disponíveis são $4\times$ maiores com 160, enquanto o número de QPs envolvidos na pesquisa aumentou apenas $1,58\times$. Uma consequência deste aspecto, é que haverão mais consultas sendo processadas simultaneamente e cada processamento local é mais eficiente por conta da densidade computacional maior. Essas características resultam na escalabilidade superlinear mostrada na Figura 3.10.

Por fim, as variações em w não afetaram significativamente o desempenho relativo entre as estratégias de particionamento de dados, entretanto afetaram a vazão de consultas (como esperado) e a inclinação dos ganhos superlineares. Isso é explicado, já que valores de menores para w levaram a um desempenho superlinear mais forte, ao passo que maiores valores de w tornam mais desafiador para a estratégia de particionamento de dados minimizar a quantidade de QPs usados.

3.4 Sumário

Neste capítulo foi apresentado uma arquitetura paralela em memória distribuída escalável e eficiente para ANNS baseada em *buckets* que recebe fluxos concorrentes de inserções e consultas. A arquitetura discutida foi apresentado no contexto do algoritmo IVFADC, entretanto, o modelo de paralelismo proposto é suficientemente genérico para implementar múltiplas estratégias de indexação para ANNS disponíveis na literatura. Neste caso, o algoritmo ANNS a ser implantado deve ser instanciado localmente em cada QPs.

Além disso, essa arquitetura é versátil e customizável para a implementação de diferentes estratégias de distribuição de dados, modificando a dinâmica de roteamento de mensagens, sem alterar a estrutura de componentes. Com isto as propostas implementadas e avaliadas neste trabalho de tese podem ser aproveitadas por outros algoritmos de busca aproximada que se baseiam na organização do conjunto de objetos em *buckets*.

Dessa maneira, na Seção 3.2 foi apresentado diferentes estratégias para a distribuição do conjunto de objetos entre QPs. Para tanto os métodos de roteamento para a comunicação (*forward* e *sendTO*) entre Coordenados e Processadores de Consultas foram personalizados para implementar as abordagens de particionamento DES, BES, SABES e SABES++.

As estratégias DES e BES são amplamente encontradas em trabalhos relacionados da literatura que até então não haviam sido sistematicamente avaliadas e comparadas. Além disso, durante o Capítulo 2 apontamos aspectos que comprometem o desempenho dessas distribuições de dados e dessa maneira, motivou a proposta das estratégias SABES e SABES++. Em linhas gerais, SABES, estende BES ao propor que *buckets* próximos no espaço sejam alocados em um mesmo QP. Adicionalmente, SABES++, propõe melhorar o balanceamento de carga proporcionado por SABES.

Durante a Seção 3.3 foi apresentado uma detalhada avaliação experimental do desempenho da solução paralela para IVFADC por meio da arquitetura distribuída proposta e também variando as estratégias de distribuição do conjunto de objetos. Os resultados mostraram que BES, SABES e SABES++ alcançaram escalabilidade superlinear em configurações de *weak scaling*, enquanto o DES apresenta desempenho próximo do linear. Além disso, a organização espacial de *buckets* proposta pelas estratégias SABES e SABES++ levaram a um maior rendimento e melhor escalabilidade. A habilidade da abordagem SABES++ de balancear os dados indexados pelos QPs proporcionou uma melhoria de desempenho de cerca de $1,12\times$ em comparação com SABES.

É importante deixar claro que embora amplamente utilizados, DES e BES não foram sistematicamente comparados na literatura, e essa também é uma contribuição deste trabalho.

Nos experimentos conduzidos neste capítulo isolamos a característica dinâmica do

conjunto de dados, no qual permite que fluxo de consultas aconteçam em paralelo á fluxo de inserção de novos descritores. Essa abordagem permitiu compreender os benefícios de cada uma das estratégias de distribuição dos dados ao tratar isoladamente fluxo de consultas. Dessa maneira, no Capítulo 4 descreveremos os mecanismos propostos para lidar eficientemente com os aspectos dinâmicos das aplicações online, bem como incorporaremos as estratégias de distribuições de dados que foram destaque nas análises e conclusões seguintes.

Capítulo 4

Fluxos Simultâneos e Conjunto Dinâmico de Objetos

Neste capítulo, abordaremos as implicações decorrentes da atualização da base de dados durante a execução da aplicação de busca aproximada. Além disso, detalharemos os componentes que foram implementados e as soluções propostas para garantir um controle eficiente das operações concorrentes, consultas e inserções de novos descritores. Vale destacar que a operação de inserção de novos descritores não havia sido abordada nas contribuições apresentadas no Capítulo 3.

Portanto, conforme analisado na Subseção 2.3.3, a atualização da base de dados em paralelo ao fluxo de inserções surge como um dos principais desafios em aplicações de Recuperação de Informação Baseada em Conteúdo (CBMR). Nesse contexto, questões provocativas emergem, tais como:

- **O volume de dados indexados localmente cresce ao longo do tempo**. Característica impacta diretamente no tempo local de busca, já que os Processadores de Consultas (QPs) devem armazenar e visitar *bucktes* cada vez maiores. Além disso, para manter a eficiente da busca local e tornar o acesso rápido, o índice local é mantido na memória principal do nó de processamento, tornando ainda mais limitado o recurso de armazenamento.

- **Consultas e inserções acontecem concorrentemente na estrutura de dados que indexam os *buckets***. A execução de tarefas com naturezas diferentes incorre em problemas de concorrência de leitura e escrita, que podem comprometer o desempenho do sistema paralelo. Essa característica é potencializada com o volume intenso dos fluxos de entrada.

- **Os Fluxos de Consulta e Inserções possuem intensidades variáveis ao longo da aplicação**. A configuração estática do nível de paralelismo da aplicação ao definir, em cada um dos nós de processamentos, a quantidade de *threads* necessárias para processar tarefas de inserções e consultas é subótima.

Considerando os aspectos que foram abordados e os desafios intrínsecos ao comportamento dinâmico do conjunto de dados, como discutido na Subseção 2.3.3, compreendemos que a solução para a busca aproximada em memória distribuída deve englobar não apenas as responsabilidades mencionadas no Capítulo 3, mas também deve ser capaz de adaptar de forma eficaz os recursos computacionais disponíveis, a fim de equilibrar as cargas de processamento provenientes das operações de inserção e consulta. Além disso, é crucial abordar o descarte de descritores indexados que, ao longo do tempo, perdem relevância.

Nesse contexto, este capítulo apresenta estratégias e mecanismos que têm como objetivo otimizar a aplicação de busca aproximada em um ambiente dinâmico de Recuperação de Informação Baseada em Conteúdo (CBMR). Para atingir esse objetivo, é proposta uma abordagem inovadora para a indexação de descritores, a qual incorpora tanto aspéctos temporais quanto espaciais. Em seguida, é detalhado um mecanismo que viabiliza o controle adaptativo dos fluxos de dados variáveis, conferindo à busca aproximada em memória distribuída uma resiliência diante das flutuações nas intensidades dos fluxos de entrada de dados.

As metodologias propostas e estudadas foram implementadas como componentes integrantes da arquitetura da máquina de busca em memória distribuída, previamente introduzida no Capítulo 3. Neste contexto, a Seção 4.1 detalha minuciosamente a arquitetura interna proposta dos nós QPs, visando enfrentar os desafios destacados. As seções subsequentes focam nos principais componentes desenvolvidos para integrar a solução final.

O primeiro deles, denominado *Índice Temporal*, é apresentado em detalhes na Seção 4.2. Sua função é gerenciar o volume de dados indexados localmente, permitindo o descarte de descritores indexados que perderam relevância para a aplicação. Este componente propõe uma abordagem de organização dos *buckets* não somente com base nas características espaciais dos objetos, mas também levando em consideração os atributos temporais associados, como o momento em que o objeto foi indexado (conforme formalizado na Definição 7). Além disso, é garantido que consultas e inserções possam ser realizadas de forma concorrente, sem bloqueios decorrentes da competição entre operações de leitura e escrita de dados. Isso assegura que novos objetos indexados estejam prontamente disponíveis para compor a base de busca.

Por outro lado, o componente *Stream Controller* é discutido na Seção 4.3. Na referida seção, é proposta uma abordagem dinâmica para o controle dos fluxos de entrada, equilibrando a alocação de recursos internos do QP entre a execução de operações de inserção e consultas. A seção detalha a implementação de uma heurística denominada MS-ADAPT, que visa adaptar de maneira dinâmica esses recursos, com o intuito de otimizar tanto o tempo de resposta quanto a vazão das consultas. No final da Seção 4.3, é conduzida uma avaliação experimental abrangente, na qual o desempenho da heurística MS-ADAPT

é comparado com a abordagem de linha de base proposta. Durante essa avaliação, são analisados os ganhos de desempenho em cada QP. Por sua vez, na Seção 4.4, é avaliado a escalabilidade da solução paralela em memória distribuída por meio das otimizações propostas.

4.1 Processador de Consultas

Na Seção 3.1 foram brevemente caracterizados os componentes que fazem parte do Processador de Consultas (QP). Como bem descrito, o QP é um processo multithread que organiza a execução dos fluxos de entrada por meio de tarefas alocadas em filas FIFO (*first-in first-out*), na Figura 4.1 estão representados com detalhes todos os componentes e relacionamentos que fazem parte deste processo.

Os objetos que chegam no nesse nó de computação imediatamente são classificados entre Tarefa de Consulta ($Task_c$) ou Tarefa de Inserção ($Task_i$), dado o componente que o enviou. Essa classificação é uma responsabilidade da *Thread de Recepção* que alocará tarefas nas respectivas filas: **Fila de Tarefas de Consulta (QQ)** ou **Fila de Tarefas de Inserções (IQ)**. Uma tarefa agrupa o descritor de origem e metainformações necessárias (como por exemplo o identificador do nó que enviou a mensagem, o tamanho dos descritor em *bytes*, etc) para o controle de comunicação na arquitetura paralela, bem como o(s) *bucket(s)* de interesse. Além disso, a intensidade de cada fluxo de dados, é mensurada pela taxa de tarefas (tarefas por segundo) que são alocadas nas respectivas filas, podendo ser então: **Taxa de Chegada de Consultas (QSR)** ou **Taxa de Chegada de Inserções (ISR)**. Valores altos associados a QSR e ISR representam fluxos de dados intensos e caracterizam um cenário desafiador para o sistema.

Existem *threads* dedicadas ao processamento de cada tipo de tarefa. As *Threads para Busca Local* são componentes trabalhadores que, quando ociosos, acessam a Fila de Tarefas de Consultas e recuperam uma nova $Task_c$. Esse tipo de trabalhador irá executar um processo de busca local (método *search* do *Índice Temporal* que será exposto na Seção 4.2), visitando os *buckets* de interesse e encontrando os k-vizinhos mais próximos localmente. É importante destacar que, o método *search* pode se beneficiar de uma execução paralela, e assim, o processo de busca local é otimizado em dois níveis de paralelismo: (i) **Paralelismo Externo de Consultas (QOT)**: se dá pela execução em paralelo de diferentes *Threads para Busca Local*, as quais executam $Task_c$ independentes. Essas *threads* externas permanecem ativas ao longo do ciclo de vida do Processador de Consultas, sendo finalizadas apenas pelo Controlador de Fluxo (que será descrito na Seção 4.3). (ii) **Paralelismo Interno de Consultas (QIT)**: se dá pela execução do

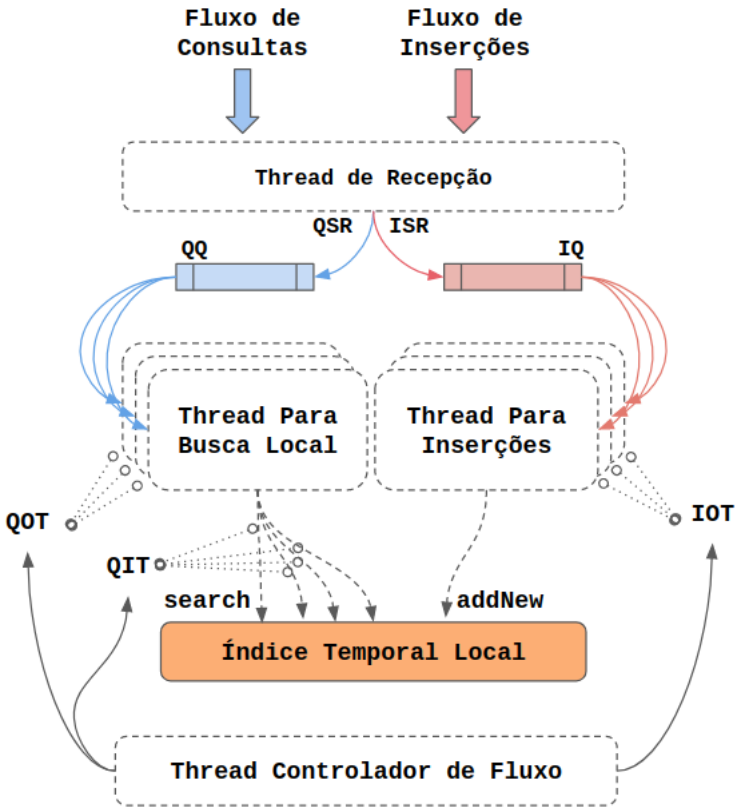


Figura 4.1: O Processador de Consultas (QP) é um componente *multithread* responsável por gerenciar a execução concorrente de consultas e inserções nos *buckets* locais. Cada descritor de entrada é alocado em uma fila de tarefas (QQ para consultas e IQ para inserções) dado o seu fluxo de origem. *Threads para Busca Local* resgatam tarefas de consultas e executam paralelamente o método *search* no Índice Temporal Local. Existem dois níveis de paralelismo durante o processo de pesquisa: (i) Externo (QOT): composto pelas execuções simultâneas de tarefas de consulta e; (ii) Interno (QIT): por meio da execução paralela de cada tarefa. Concorrentemente *Threads para Inserções* coletam tarefas de inserção e executam o método *addNewDescriptor*, que indexa um novo objeto no *bucket* mais próximo do Índice Temporal Local. Esse processo é otimizado apenas por um paralelismo externo (IOT). A *Threads Controlador de Fluxo* é persistente ao longo do ciclo de vida do processo QP e executa o Controlador de Fluxo (Seção 4.3), no qual é responsável por adaptar a alocação de recursos entre os níveis de paralelismo QOT, IOT e QIT.

método *search* paralelamente. Nesse caso diferentes *threads* internas participam da busca referente à uma $Task_c$, realizam a visita em partições diferentes dos *bucktes* de interesse e, ao fim do processo de pesquisa, são finalizadas.

Por sua vez, as **Threads para Inserções**, são componentes trabalhadores que acessam a Fila de Tarefas de Inserções e recuperam uma nova $Task_i$. O processo de execução de uma tarefa de inserção consistem em indexar o objeto associado ao *bucket* mais próximo. Para tanto, as **Threads para Inserções** executam o *indexNew* do *Índice Temporal*, que também será exposto na Seção 4.2. É importante destacar que, pela natureza do processo de indexação de um novo descritor, o *indexNew* não se beneficia de

uma otimização por meio de paralelismo interno e assim, um único nível de paralelismo é possível: (i) **Paralelismo Externo de Inserções (IOT)**: formado pela execução em paralelo de diferentes **Threads para Inserções**, nas quais $Task_i$ independentes são tratadas.

Vale enfatizar que para a coleta de tarefas em ambas filas globais, é aplicado uma abordagem de *bag-of-tasks*, na qual garante-se o controle de acesso de múltiplas *threads* concorrentemente. E além disso, conforme discutido, a proposta do *Índice Temporal* torna possível que tarefas de consulta e tarefas de inserções aconteçam simultaneamente com eficiência no índice local.

Ademais, para coordenar a alocação das *threads* descritas anteriormente e garantir níveis de paralelismo adequados às taxas de chegada de tarefas, QSR e ISR, a **Thread para Controle de Fluxo** executa o Controlador de Fluxo que será descrito na Seção 4.3. Essa *thread* é persistente ao longo do ciclo de vida do Processador de Consultas.

4.2 Suportando Índices Temporal

Durante a Subseção 2.3.3 foi pontuado que aspectos inerentes ao comportamento dinâmico da base de dados em aplicações CBMR, acrescentam complexidades em diferentes contextos da máquina em memória distribuída para busca aproximada. Alguns desses desafios, permeiam a capacidade de tornar possível que a busca por descritores similares aconteça de forma paralela e concorrente à inserção de novos descritores na estrutura que organiza a base de dados. Além disso, e complementarmente, agregar informações espaciais dos descritores e também temporais, como por exemplo o instante em que foi indexado, torna a indexação um mecanismo capaz de permitir que a busca aproximada escolha em qual região espacial e temporal do conjunto de descritores é interessante a procura pelos objetos similares. Portanto, incluir a informação temporal na organização espacial dos descritores, permite que consultas e inserções possam acontecer em partições distintas e, indo além, torna a maquina em memória distribuída capaz de distinguir eficientemente descritores que estão indexados à mais tempo indexados, habilitando a oportunidade de descartar aqueles que eventualmente são mais antigos e desinteressantes. Dessa maneira, para endereçar os benefícios apontados, é proposto o componente Índice Temporal.

O Índice Temporal proposto tem por objetivo fornecer uma estrutura de dados na qual objetos são agrupados pela proximidade espacial e também por coincidência do instante em que foram indexados. Esse organização armazena objetos em *buckets temporais* conforme formalizado na Definição 7, onde enuncia que $\forall x, y \in TX_n$, os objetos x e y foram indexados no instante t_n e $v \in (X - TX_n) | dist(x, y) \leq dist(x, v) \wedge t(x) \neq t(v)$.

De maneira ampla, é proposto o particionamento dos *buckets* em grupos temporais, os quais vão sendo criados à medida que a linha do tempo avança e um intervalo é finalizado. Um intervalo é formado por s segundos, e é um parâmetro da aplicação (como exemplo, no trabalho [49] foi adotado intervalos de $s = 864$ segundos). Assim, o instante t_n é iniciado $(n * s)$ segundos após o inicio da aplicação e todos os objetos indexados entre $[(n * s), ((n + 1) * s]$ pertencem ao *bucket temporal* TX_n. A Figura 4.2 mostra a organização de um índice local por meio de *buckets temporais*.

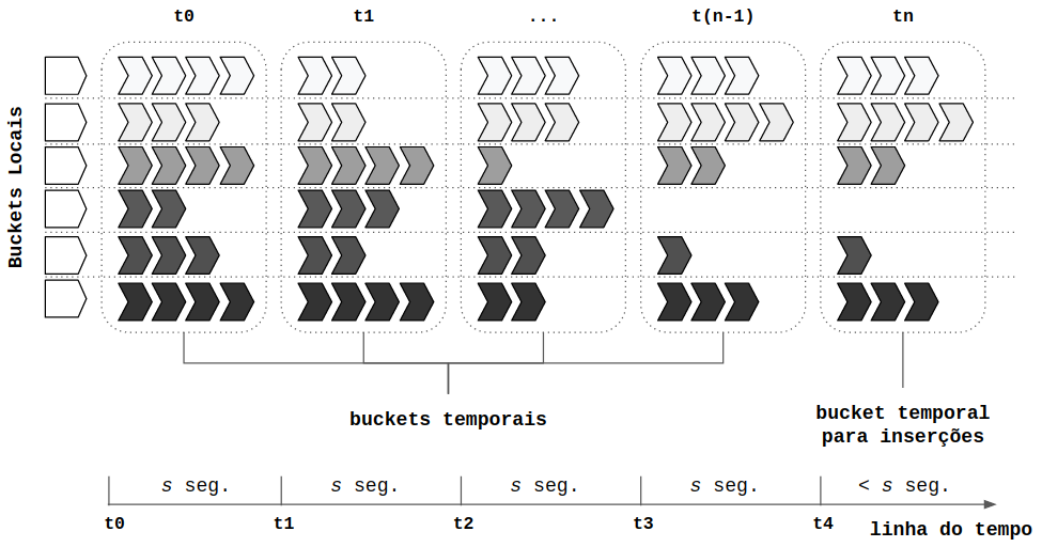


Figura 4.2: O Índice Temporal organiza objetos pela proximidade espacial e também pela coincidência do instante em que foram indexados. A cada intervalo de s segundos, um novo *bucket temporal* é criado. Os *bucket temporais* em t_n recebem a indexação de novos objetos.

4.2.1 Busca e Indexação

Em vista da organização apresentada, os objetos no Índice Temporal estão indexados por meio de dois níveis: (i)**Espacial**: o objeto está armazenado no *bucket* mais próximo e; (ii) **Temporal**: o objeto está na partição do *bucket* referente ao instante em que foi indexado. O processo de indexação de um novo objeto é similar à indexação discutida na Seção 3.1, assim, primeiro encontra-se o *bucket* mais próximo e então o objeto é armazenado na partição temporal que representa o instante corrente (t_n) deste *bucket*. Dessa maneira, apenas o subconjunto corrente de *buckets temporais* irá receber a indexação de novos descritores. A operação de indexação no Índice Temporal é executada pelo método *addNewDescriptor* e está ilustrada na Figura 4.3.

Por sua vez, a busca pelos k-vizinhos mais próximos à um objeto, no Índice Temporal é executada pelo método *search*. Nessa operação, os *bucktes* de interesse são visitados em todas as partições temporais presentes. Pensando na otimização desse processo de busca local é proposto executá-lo paralelamente por meio de *threads* destinadas ao Paralelismo Interno de Consultas (QIT), conforme descrito na Seção 4.1. Nessa abordagem as partições temporais T dos w *buckets* de interesse são distribuídos dentre as QIT *threads* que executam a busca pelos k-vizinhos mais próximos em cada um dos $(T * w)/QIT$ *bucket temporal* assinalados à ela. É importante destacar, que antes de iniciar a visita à um *bucket*, cada *thread* verifica se esse está atualizado (conforme Definição 8). Caso contrário, a *thread* é bloqueada até a atualização ser efetivada. Ao fim das pesquisas, é executado um passo de agregação (passo *merge* do método *search*) dos k-vizinhos encontrados por cada QIT *threads* em um conjunto de k-vizinhos global à essa busca local. A distribuição do conjunto de *buckets temporais* entre QIT *threads*, e a agregação das respostas individuais é responsabilidade da *Thread para Busca Local* que compõe o Paralelismo Externo de Consultas (QOT). O método *search* também está ilustrado na Figura 4.3.

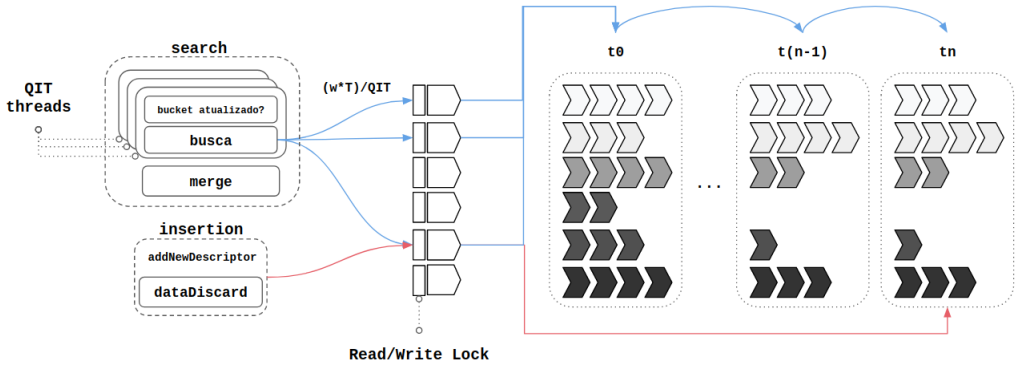


Figura 4.3: A busca no Índice Temporal acontece de forma paralela por meio de QIT *threads*. Cada *thread* visita $(w * T)/QIT$ *buckets temporais* e ao final do processo os resultados individuais são agregados, no passo *merge*, em uma resposta global. Por sua vez, a indexação de novos objetos acontece na partição temporal mais recente do *bucket* mais próximo. O controle de acesso para leitura e escrita em cada *buckets* é feito pelo padrão *read/write lock*, garantindo a consistência e baixo custo de sincronização para execuções concorrentes de consultas e inserções.

4.2.2 Descarte de Objetos

Para garantir que a quantidade de objetos indexados não extrapole a capacidade de armazenamento em memória principal do Processador de Consulta, é definido, por parâmetro, um limite máximo de objetos por Índice Temporal Local.

Esse parâmetro (T) define o tamanho da janela de instantes que estarão disponíveis para a busca. Ao iniciar um novo instante, caso já existam T instantes disponíveis, os *buckets temporais* associados ao instante mais antigo são descartados. Essa estratégia garante um limite de T de partições temporais em cada *buckets*, e controla, não só a quantidade de objetos indexados em um Índice Temporal Local, mas como também a complexidade computacional do processo de busca.

4.2.3 Concorrência entre operações

É de grande importância evidenciar que as operações de indexação de novos objetos e a consulta acontecem simultaneamente no Índice Temporal. Entretanto, existe concorrência entre consultas (leitura) e indexação (escrita) apenas no subconjunto t_n de *buckets temporais*. Para gerenciar essas operações concorrentes, é proposto uma sincronização baseada na estratégia *read/write lock* [35], no qual o acesso à um *bucket* está limitado à uma operação de indexação por vez. Dessa maneira, antes de iniciar a execução de escrita em um *bucket*, a *thread* responsável pela operação obtém o bloqueio do *bucket* e, caso tenha sucesso, a operação é executada, caso contrário, irá esperar até o *bucket* ser liberado. Essa estratégia permite que todo o Índice Temporal seja consultado e que novos descritores sejam indexados com o custo apenas de sincronização no subconjunto de *buckets* referentes ao instante corrente. Essa sincronização proporcionado pelo *read/write lock* possui pequeno impacto no tempo de execução de aplicação, já que a operação de indexação, responsável por bloquear o acesso ao *bucket*, é em média 52.6× mais rápida que a operação de leitura em um *bucket* e os bloqueios acontecem apenas em um subconjunto restrito de *buckets*.

4.2.4 Avaliações

A solução apresentada no trabalho [49] propôs a construção de duas estruturas de *buckets* (Static Tables e Delta Tables) para suportar a indexação de novos descritores concorrentemente à consultas. A abordagem discutida é específica para o algoritmo de busca LSH, e sugere que novos objetos sejam indexados apenas em Delta Tables ao passo que as consultas aconteçam apenas em Static Tables, para eliminar bloqueios de leitura à uma Tabela que está constantemente recebendo indexação de novos descritores. Essa

Tempo Total	Read/Write Lock	Descarte	Custo
300s	1,31s	0,56s	0.62%

Tabela 4.1: Porcentagem de tempo gasto por mecanismos de sincronização necessários pelo Índice Temporal.

solução, apesar de evitar os custos decorrentes de sincronização de acesso à estrutura de indexação usada para a pesquisa, necessita que constantemente os novos descritores indexados em Delta Tables sejam combinados nas Static Tables para compor a base de busca. Esse processo insere uma complexidade no sistema que impacta diretamente o tempo de resposta das consultas, já que quando necessária a combinação, toda consulta que chega no sistema é armazenada para execução posterior.

O Índice temporal proposto e apresentado na seção anterior, remove essa complexidade ao permitir que consultas e inserções aconteçam na mesma estrutura, com irrisório custo de sincronização. Esse aspecto é possível apenas pelo fato do Índice Temporal organizar objetos pela proximidade espacial e pelo instante em que foram indexados, por meio de *buckets temporais* independentes. Para avaliar o impacto da solução apresentada, foi usado um Processador de Consulta que executa a estratégia IVFADC configurada com 4096 centroides representativos (*buckets*) particionados em $T = 4$ instantes temporais. Inicialmente o Índice Temporal foi preenchido com $1M$ de descritores SIFT, e os fluxos de consulta e inserções, tiveram intensidade (QSR e ISR) constante ao longo da execução da aplicação com duração total de 300 segundos. Para esse teste, foi usado o intervalo $s = 120$ segundos de forma que o Índice Temporal pudesse realizar pelo menos dois processos de descarte de dados. Foi mensurado o tempo total gasto (acumulado) com os bloqueios decorrentes da sincronização desempenhada pelo *read/write lock* e também pelo processo de descarte de objetos antigos. A Tabela 4.1 sumariza os números alcançados. Nela é possível observar que as despesas totais relacionadas ao custos das sincronizações (coluna "Custo"), representam apenas 0.62% do tempo total de execução da aplicação. Durante essa avaliação foram indexados 2025486 novos descritores e 57870 consultas concluídas, com vazão equivalente a $192,9$ consultas por segundo.

O trabalho [49] aponta que o custo para a gestão dos fluxos concorrentes por meio das estruturas de Static Tables e Delta Tables é $1,7\%$ do tempo total de execução da aplicação. Assim, o Índice Temporal proposto, alcançou um desempenho $2,74\times$ superior à proposta presente na literatura.

Por fim, é importante destacar que existe uma relação entre os parâmetros T (quantidade de partições temporais) e s (instantes temporais), e o tempo de busca local. De forma superficial, a configuração destes parâmetros está diretamente relacionada às características da aplicação online, e irá direcionar se as buscas locais vão visitar muitos *buckets temporais* pequenos ou poucos *buckets temporais* grandes. Explorar esses compromissos

é uma oportunidade para extensão deste trabalho de tese.

4.3 Gestão adaptativa de recursos

No cenário de aplicações online, a taxa de chegadas de consultas e de inserção de novos descritores não é constante. Esse aspecto acrescenta interessantes desafios em termos do desempenho da solução em memória distribuída, e está diretamente relacionado com a configuração de paralelismo empregada durante a busca local em um Processador de Consultas. Alguns trabalhos anteriores, discutidos na Subseção 2.3.3.1, mostraram que, em diferentes contextos de aplicações e também para busca aproximada, a variabilidade na intensidade de fluxos de dados torna uma configuração de paralelismo estática subótima em termos do tempo de resposta do sistema.

Conforme foi apresentado nas Seções 4.1 e 4.2, o Processador de consulta é implementado por meio de diversos conjuntos de *threads*, e dentre elas, algumas compõem os níveis de paralelismo para execução de tarefas de consultas e tarefas de inserções: (i) Externo para Consultas (QOT); (ii) Interno para Consultas (QIT); (iii) Externo para Inserções (IOT). O paralelismo Externo (QOT e IOT), desempenhado respectivamente por *Threads para Busca Local* e *Threads para Inserções*, definem o número de tarefas que serão executadas independentemente e concorrentemente. Por sua vez, o paralelismo Interno (QIT), possível apenas em tarefas de consulta, refere-se ao tamanho do grupo de *threads* (quantidade de *threads*) usadas para executar cada uma das tarefas.

Durante esta seção, o nível de paralelismo e o particionamento de recursos empregado em QP para a execução de tarefas, será representado pela tupla $[QOT][QIT][IOT]$, a motivação para a adaptação dinâmica desta configuração é dada tal que, conforme inicialmente discutido no trabalho [4], para diferentes intensidades dos fluxos de dados, diferentes combinações do nível de paralelismo alcançam o melhor tempo de resposta de consultas e, além disso, fluxos concorrentes de inserções e consultas demandam distribuições de recursos paralelos dinâmicas ao longo do ciclo de vida da aplicação para alcançar .

Sendo assim, em cada Processador de Consulta, é implementado um componente responsável por configurar dinamicamente o nível de paralelismo $[QOT][QIT][IOT]$ que será usado para execução das tarefas. O componente Controlador de Fluxo (ver Figura 3.3), pode implementar diferentes estratégias para a adaptação dinâmica das*threads* disponíveis (*thread pool*) em uma configuração $[QOT][QIT][IOT]$. E dessa forma, neste trabalho é proposto a abordagem MS-ADAPT que, dado diferentes observações relacionadas ao fator de carga dos fluxos de entrada, consegue, em tempo de execução, definir

o nível de paralelismo mais adequado aquele cenário. Essa estratégia é apresentada na Subseção 4.3.1.

Ademais, foi avaliado MS-ADAPT observando o desempenho local ao mensurar o tempo de resposta de consultas em um nó de processamento QP. Como linha de base, foram usadas diferentes configurações estáticas de paralelismo, e uma abordagem de inserções sob demanda, que aqui chamaremos *Lazy Insertions*, para o controle dos fluxos concorrentes. Ao comparar MS-ADAPT com a linha de base é possível compreender o quão bem a estratégia proposta consegue adaptar-se dinamicamente à variação da intensidade dos fluxos. Essa avaliação experimental é apresentada na Seção 4.4.

4.3.1 MS-ADAPT

A estratégia MS-ADAPT foi proposta para adaptar dinamicamente o nível de paralelismo $[QOT][QIT][IOT]$. A intuição por detrás desta abordagem é a otimização do tempo de resposta de consultas ao maximizar a quantidade de recursos alocados para QOT e QIT mantendo *buckets* atualizados, em respeito ao parâmetro de BOF.

Conforme enunciado pela Definição 8, o parâmetro da aplicação BOF (Flexibilidade de desatualização de *buckets*) diz qual o intervalo máximo aceitável (em segundos) que *buckets* podem estar desatualizados para que a busca aconteça. Um *bucket* q é dito desatualizado, caso exista pelo menos uma tarefa de inserção, que tenha q como alvo, pendente na lista de tarefas IQ. A desatualização de *buckets* penaliza consideravelmente o tempo de resposta de consultas, já que consultas que visitam *buckets* momentaneamente desatualizados, precisam ser bloqueadas até que a indexação pendente seja efetivada. Dessa maneira, MS-ADAPT propõe manter a taxa de desatualização de cada *bucket* inferior ao parâmetro BOF ao empregando a quantidade mínima necessária de *threads* para a execução de tarefas de inserção (IOT).

Com isso, ao destinar o máximo de recursos possíveis para a execução de tarefas de consulta o balanceamento de *threads* entre QOT e QIT irá garantir a otimização das métricas de vazão e tempo de resposta dado a variabilidade da intensidade do fluxo. No trabalho [4], mostrou-se que em momentos com baixa intensidade, ao aumentar o paralelismo interno (QIT) o tempo de execução da busca local é acelerado, impactando diretamente na redução do tempo de resposta de consultas. Por sua vez, quando a intensidade do fluxo de tarefas é alta, várias consultas ficam disponíveis para computação simultânea e, assim, aumentar o paralelismo externo (e diminuindo o interno) tende a ser a melhor opção para a otimização da vazão de tarefas de consulta do sistema.

Para alcançar a intuição descrita, MS-ADAPT envolve três passos principais:

(i) Monitoramento; (ii) Otimização; (iii) Reconfiguração. Cada uma das etapas estão descritas na sequência.

4.3.1.1 Monitoramento

Nesta etapa são coletadas diferentes métricas relacionadas a intensidade dos fluxos de entrada. Essas métricas fornecem a visão necessária para que, no passo de Otimização, seja possível compreender qual a configuração mais apropriada para lidar com o estado corrente do sistema. Pontualmente, são monitorados:

- *Taxa de chegada de consultas (QAR)*: quantidade média de tarefas alocadas por segunda na fila de tarefas de consultas (QQ).
- *Tamanho da fila de consultas (QQS)*: quantidade média total de tarefas pendentes alocadas na fila de tarefas de consulta.
- *Desatualização máxima de bucket (BOM)*: valor de desatualização do *bucket* mais desatualizado.

É importante destacar, que as métricas descritas correspondem à média de valores coletados em um intervalo de 0.05 segundos.

4.3.1.2 Otimização

Durante este passo, por meio das métricas coletadas anteriormente, são identificados sinais que indicam quais dos nível de paralelismo (QOT, QIT ou IOT) precisam ser potencializados ou reduzidos. Assim, são observados:

- **Variação da métrica QAR**: (i) **Aumento de QAR**. Esse sinal indica uma crescente intensidade do fluxo de consultas. Consequentemente, é preciso potencializar os recursos de paralelismo externo (QOT), para que, conforme proposto em [4], a vazão do sistema seja suficiente para reduzir o acumulo de tarefas em QQ. (ii) **Redução de QAR**. Por sua vez, este sinal indica que a taxa de chegada de tarefas de consulta está mais amena e, portanto, é possível liberar *threads* em QOT para: 1) acelerar o tempo de execução de consultas, acrescentando *threads* em QIT ou 2) aumentar a vazão de tarefas de inserção, potencializando IOT.

- **Aumento de QQS**: Este sinal indica que existe acúmulo crescente na fila de tarefas de consulta, portanto, é preciso, assim como pelo aumento de QAR, acrescentar *threads* em QOT para otimizar a vazão do sistema.

- **BOM próximo do limite BOF**: (i) **BOM próximo de BOF**. Sinal que aponta a existência de *bucket* com intervalo de desatualização próximo do limite aceitável para que não haja bloqueio na consulta. Nesse sentido, é preciso aumentar *threads* externas (IOT) para a execução de tarefas de inserção, afim de acelerar a atualização de *buckets*. (ii) **BOM próximo de zero**. Indica que praticamente todos os *buckets* estão completamente atualizados e, portanto, é possível liberar recursos de IOT para otimizar a execução do fluxo de tarefas de consultas.

A Figura 4.4 ilustra os diferentes sinais descritos e quais as possíveis reações.

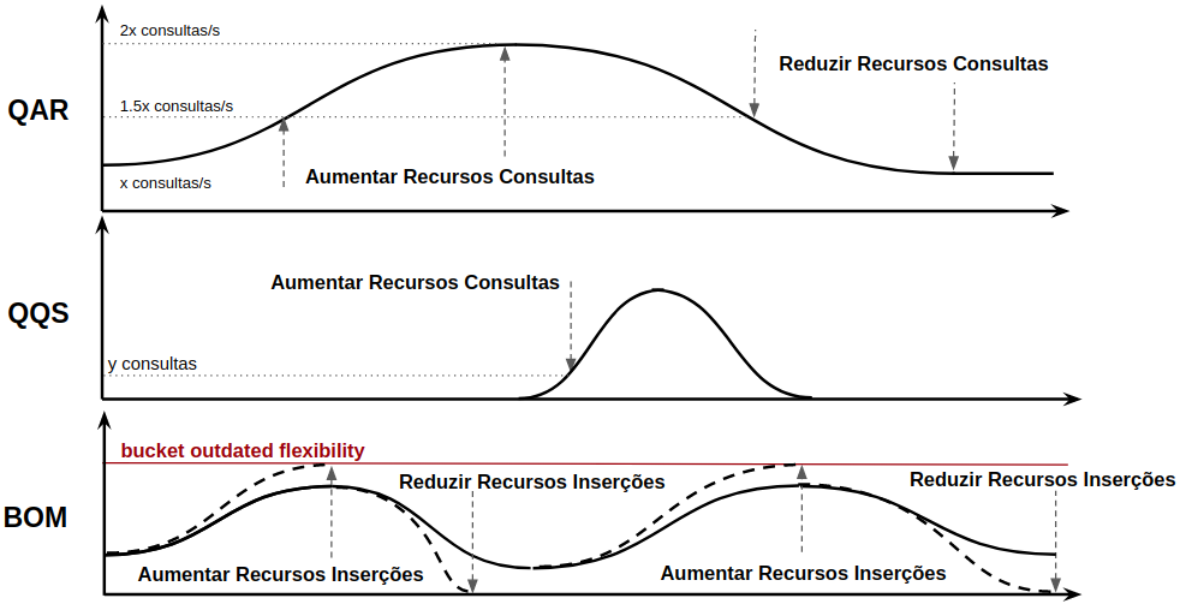


Figura 4.4: Para cada uma das métricas monitoradas, existem comportamentos de interesse que guiam a abordagem MS-ADAPT no balanceamento dos níveis de paralelismo QOT, QIT e IOT.

4.3.1.3 Reconfiguração

Após identificada a configuração $[QOT][QIT][IOT]$ mais adequada ao estado momentâneo do sistema, inicia-se o passo de reconfiguração. Para isso, o componente Controlador de Fluxo aguarda até que todas as *threads* em execução terminem o trabalho corrente, e posteriormente, implementa a nova distribuição de paralelismo.

Para que não haja comprometimento na performance do sistema, é mantido um *thread pool*, no qual ao iniciar uma nova configuração, *threads* são associados à uma função de trabalho: QOT, QIT ou IOT. Com esse formato, as reconfigurações acontecem sem proporcionar inconsistência ou perda de desempenho ao sistema.

4.3.1.4 Algoritmo

O algorítimo MS-ADAPT inicia-se com U *threads* (*thread pool*) disponíveis para serem alocadas entre os níveis de paralelismo QOT, QIT e IOT tal que, $(|\text{QOT}| * |\text{QIT}|) + |\text{IOT}| = |U|$. A configuração inicial é dada por $[|U| - 1][1][1]$, na qual o máximo de *threads* estão alocadas para otimizar a vazão da execução de consultas. O Algoritmo 6 mostra a sequencia de execuções desempenhada pela estratégia MS-ADAPT e é correspondente a: 1) O Monitoramento captura as métricas de interesse; 2) O passo de Otimização conduz a escolha da configuração mais adequada seguindo os sinais e reações apontados na descrição apresentada no Tópico 4.3.1.2. 3) Por fim, dado a configuração resultante do passo 2, o Controlador de Fluxo implementa os níveis de paralelismo definido, por meio do passo Reconfiguração.

Algorithm 6: Algoritmo MS-ADAPT

```
1  def ms_adapt(QOT, IOT, BOF):
2      Passo 1: Monitoramento
3      BOM_i, QQS_i, QAR_i ← monitor()
4
5      Passo 2: Otimização
6      if BOM ≥ BOF then
7          [nQOT][nQIT][nIOT] ← aumentarIOT(QOT, IOT)
8      else
9          if BOM ≤ 0 then
10             [nQOT][nQIT][nIOT] ← reduzirIOT(QOT, IOT)
11     if (QQS_i > QQS_{i-1}) ∨ (QAR_i > QAR_{i-1}) then
12         [nQOT][nQIT][nIOT] ← aumentarQOT(QOT, IOT)
13     if QAR_i < QAR_{i-1} then
14         [nQOT][nQIT][nIOT] ← reduzirQOT(QOT, IOT)
15     else
16         [nQOT][nQIT][nIOT] ← [QOT][QIT][IOT]
17
18     Passo 3: Reconfiguração
19     return reconfigurar([nQOT][nQIT][nIOT])
20
```

Durante o passo 2, o algoritmo checa se a métrica BOM está no limiar máximo in-

dicado pelo parâmetro BOF, e caso positivo, tenta acrescentar mais uma *thread* para lidar com o fluxo de inserções (*aumentarIOT*). Caso contrário, é verificado se BOF alcançou o limiar mais baixo, indicando que se é ou não possível liberar recursos do fluxo de inserções (*reduzirIOT*). Quando a métrica BOF está entre o limiar máximo e mínimo, o algoritmo MS-ADAPT verifica se houve acúmulo de descritores na fila de consultas (QQS) ou se a taxa de chegada de consultas (QAR) aumentou. Caso alguma das condições apontadas tenham sido atendidas, então é necessário aumentar recursos para lidar com o fluxo de consultas (*aumentarQQT*). Por fim, e caso os cenários descritos anteriormente não tenham sido aceitos, MS-ADAPT verifica se houve queda na taxa de chegada de consultas (QAR), sinalizando que é possível liberar recursos de consultas (*reduzirQOT*).

A implementação dos métodos *aumentarQOT*, *reduzirQOT*, *aumentarIOT* e *reduzirIOT* estão detalhados nos Algoritmos 7, 8, 9 e 10, respectivamente. É importante observar que, no inicio dos métodos *aumentarQOT* e *aumentarIOT* (linha 2 e linha 2), é verificado se é possível acrescentar mais recursos à esse nível de paralelismo. Além disso, prioritariamente são acrescido ou decrescidos os níveis de paralelismo externo, e então as *threads* restantes são associadas ao paralelismo interno. Os acréscimos ou decréscimos podem acontecer em quantidades definidas por parâmetro, e configuradas pela variável *steps* nos algoritmos apresentados.

Algorithm 7: Aumentar QOT

```
1 def increaseQOT(QOT, IOT):
2     if QOT + step > |U| then
3         return lazy
4     nQOT ← QOT + step
5     nIOT ← IOT − step
6     nQIT ← (|U|−(newQOT+newIOT)) / newQOT
7
8     return [nQOT][nQIT][nIOT]
9
```

Algorithm 8: Reduzir QOT

```
1 def decreaseQOT(QOT, IOT):
2     if QOT − step ≤ 0 then
3         return false
4     nQOT ← QOT − step
5     nIOT ← IOT + step
6     nQIT ← (|U|−(newQOT+newIOT)) / newQOT
7
8     return [nQOT][nQIT][nIOT]
9
```

Algorithm 9: Aumentar IOT

```
1 def increaseIOT(QOT, IOT):
2     if IOT + step > |U| then
3         return lazy
4     nQOT ← QOT − step
5     nIOT ← IOT + step
6     nQIT ← (|U|−(newQOT+newIOT)) / newQOT
7
8     return [nQOT][nQIT][nIOT]
9
```

Algorithm 10: Reduzir IOT

```
1 def decreaseIOT(QOT, IOT):
2     if IOT − step ≤ 0 then
3         return false
4     nQOT ← QOT + step
5     nIOT ← IOT − step
6     nQIT ← (|U|−(newQOT+newIOT)) / newQOT
7
8     return [nQOT][nQIT][nIOT]
9
```

Em algumas situações a intensidade da carga dos fluxos de entrada extrapolam a capacidade de processamento do sistema, nesse sentido, MS-ADAPT emprega um compor-

tamento baseado na estratégia *Lazy Insertions* até que a intensidade da carga de trabalho se amenize.

A gestão de fluxos concorrentes por meio da abordagem *Lazy Insertions*, propõe que as indexações de novos objetos aconteçam sob demanda. Nesse sentido, cada QIT *thread* é responsável por, antes de iniciar a visita em um *bucket*, verificar se este está atualizado e, caso não esteja, realizar as execução das tarefas de inserções que estão pendentes. A atualização sob demanda acontece por meio da execução do método *bucketUpdate* no qual coleta um conjunto de tarefas de inserção pendentes na fila IQ, e para cada uma delas executa a rotina de inserção *addNewDescriptor*. Na Figura 4.5 está ilustrado o processo da estratégia *Lazy Insertions*.

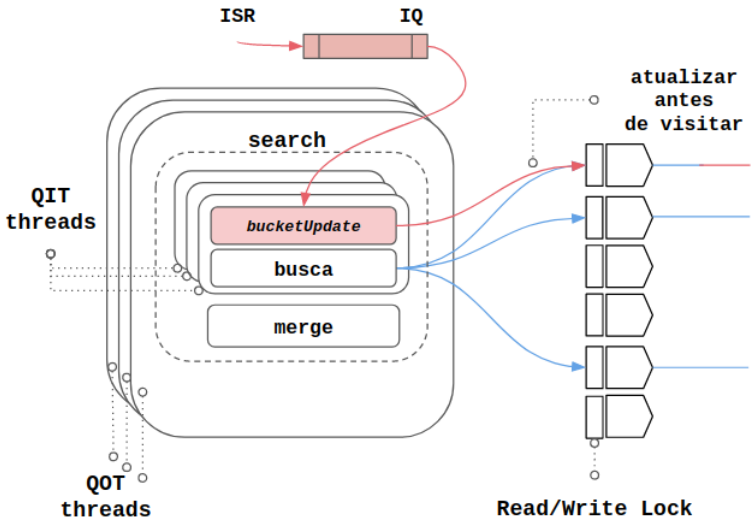


Figura 4.5: A abordagem Lazy realiza inserções sob demanda. Essa gestão de fluxo propõe que, *buckets* sejam atualizados antes das QIT iniciarem a busca local.

É importante destacar que esta solução é indiferente ao parâmetro BOF, já que, antes de qualquer busca local os *buckets* necessariamente estarão atualizados. Além disso, as inserções sob demanda acrescentam relevantes despesas computacionais ao processo de busca local, podendo ser impactante ao tempo de resposta das consultas principalmente em contexto com intenso fluxo de tarefas de inserção.

4.4 Avaliação Experimental

Nesta seção, será realizada uma avaliação do desempenho da busca aproximada utilizando a abordagem IVFADC, que foi implementada sobre a arquitetura detalhada no Capítulo 3, juntamente com os ajustes dinâmicos de paralelismo propostos neste capítulo.

A adaptação dinâmica dos níveis de paralelismo ocorre dentro do componente Controlador de Fluxo, que está presente no nó de computação conhecido como Processador de Consultas (QP). A avaliação experimental será conduzida em duas fases distintas: 1) Inicialmente, será avaliado o desempenho da busca aproximada executada localmente em um único QP. Essa abordagem se concentra nas métricas de tempo de resposta e vazão de um único nó Processador de Consultas. Isso será feito ao instanciar apenas um QP em uma máquina, permitindo a discussão dos compromissos relacionados à otimização implementada. 2) Em seguida, procederemos à avaliação da busca aproximada em uma configuração de memória distribuída, onde múltiplas máquinas estão envolvidas. Cada nó de computação QP executará a otimização proposta, e nesta fase será examinado o benefício global das otimizações locais de busca.

Nesse contexto, serão analisados tanto os resultados obtidos ao considerar um único nó Processador de Consultas quanto os ganhos alcançados ao expandir a execução para uma configuração de memória distribuída, com o intuito de avaliar o impacto das otimizações propostas em ambas as situações.

4.4.1 Avaliação Otimização MS-ADAPT

Para a avaliação da otimização proporcionada pelo ajuste dinâmico de paralelismo, foi considerado apenas uma máquina instanciando um nó de computação QP, configurada com um Índice Temporal $T = 4$ e $s = 360$ de modo que a quantidade de dados indexados durante os testes não extrapolem a memória primária da máquina. Além disso, este QP está executando uma estratégia de busca aproximada IVFADC configurada com $C_c = 4096$ e $w = 16$ mantendo a qualidade da busca equivalente a recall@R=100 de $63,7\%$. Inicialmente foi carregado no Índice Temporal 1 bilhões de descritores SIFT. A máquina está equipada com processador AMD EPYC 7742 rodando 64 cores a 2.25-3.40 GHz e 256GB de memória RAM. Um nó de processamento QP, conforme descrito na Seção 4.1, depende da execução de 4 *threads* dedicadas a controle e comunicação, portanto, estão disponíveis 60 *threads* para serem configuradas nos níveis de paralelismo [QOT][QIT][IOT].

Os experimentos conduzidos nessa seção observam o impacto das configurações de paralelismo no tempo de resposta das consultas por meio de várias intensidades dos fluxos de entrada. Para implementar diferentes taxas de chegada de tarefas foi usado como referência a vazão máxima do sistema para cada um dos fluxos. A vazão máxima (tarefas/segundo) foi computada em duas execuções isoladas onde foram efetuadas só inserções ou só consultas, além disso foi alocado a quantidade máxima de *threads* (60) para compor o paralelismo externo em cada caso (QOT e IOT), conforme discutido no

Fluxo	Configuração	Vazão
Consultas	[60][1][0]	126,18 $task_c/s$
Inserções	[0][0][60]	9350,15 $task_i/s$

Tabela 4.2: Vazão máxima de tarefas para cada um dos fluxos de entrada.

trabalho [4] onde é apontado que o máximo de recursos no paralelismo externo conduz a vazão máxima do sistema. Na Tabela 4.2 estão presentes as configurações usadas e o valor vazão máxima alcançada.

Assim, nesta seção, as diferentes intensidades dos fluxos vão ser dadas por:

- **Fator de Carga de Consultas (QLF)**: corresponde a porcentagem da vazão máxima de consulta, de forma que QLF=1 representa 126,18 $task_c/s$.
- **Fator de Carga de Inserções (ILF)**: corresponde a porcentagem da vazão máxima de inserções, de forma que ILF=1 representa 935,15 $task_i/s$.

A taxa de chegadas de tarefas, que será usada em cada execução, pode ser recuperada ao calcular $arrival_{rate}$ = (QLF(ou ILF) × respectiva vazão máxima). E é extremamente importante destacar que em cada execução é usado combinações de níveis de QLF e ILF simultaneamente sendo que a chega de tarefas segue uma distribuição *Poisson* com $\lambda = arrival_{rate}$. Dessa maneira, em cada execução da avaliação experimental foi simulado um sistema no cenário de serviços online, onde QLF e ILF foram variados em 5 níveis diferentes $(0,2;\ 0,4;\ 0,6;\ 0,8;\ 1,0)$, sendo que as possíveis combinações QLF × ILF proporcionam uma matriz com 25 cenários de cargas distintos. Foram excluídas combinações onde $QLF + ILF \geq 120$, ou seja, cenários onde a carga de trabalho é superior a 120% do máximo suportado pelo sistema (triangulo superior). Entende-se que nesses casos, a perda de desempenho é inevitável em decorrência da falta de recursos para atender a demanda de requisições.

Como linha de base, para a comparação com a proposta MS-ADAPT, foram usadas 11 diferentes configurações estáticas, as quais variam QOT e IOT em passos de 5 *threads* (variável *steps* nos Algoritmos 7, 8, 9 e 10), como por exemplo: [5][1][55], [10][1][50], etc.

É de extrema importância destacar que neste estudo, o nível de Paralelismo Interno de Consultas (QIT) foi mantido constante e definido no patamar mínimo, onde apenas uma *thread* é alocada para essa finalidade (QIT=1). Essa decisão é fundamentada na análise apresentada no trabalho anterior trabalho [4], onde se concluiu que as variações no valor de QIT têm um impacto limitado no tempo de resposta das consultas. Como resultado, optou-se por manter esse nível de paralelismo inalterado, a fim de otimizar a alocação de recursos para os níveis externos de paralelismo, ou seja, entre as consultas (QOT) e as operações de inserção (IOT).

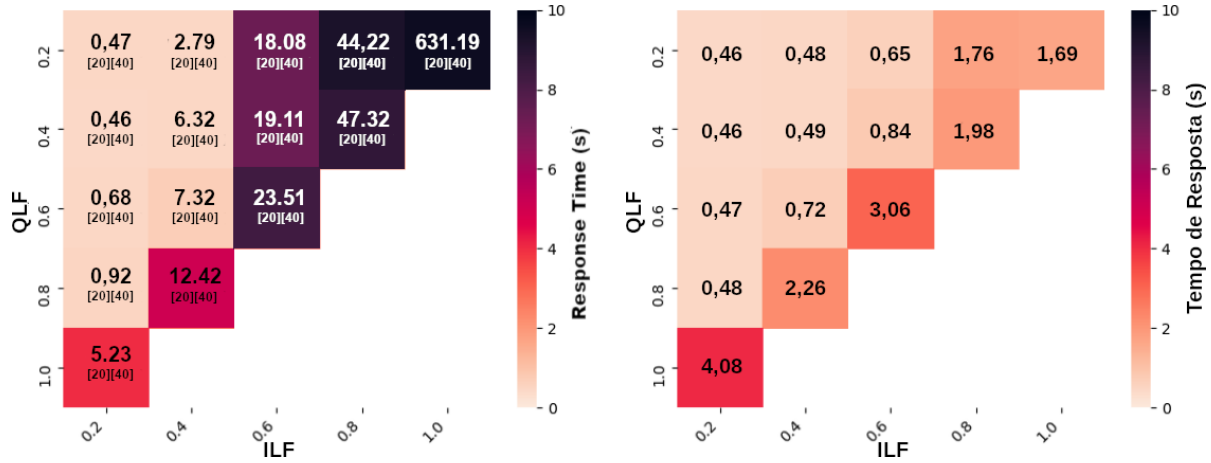
Além disso, foi estabelecido um valor de *step* = 5 para os algoritmos 7, 8, 9 e 10 presentes na estratégia MS-ADAPT. Essa escolha permite percorrer somente as configurações estáticas pré-definidas ao ajustar os níveis de paralelismo de forma dinâmica. Ademais, é relevante mencionar que, quando necessário, a estratégia MS-ADAPT inicia o modo de execução *Lazy Insertions*, configurando a distribuição [60][1][0], como definido para esse contexto específico.

As Figuras 4.6, 4.7 e 4.8, apresentam os resultados comparativos entre as abordagens linha de base e MS-ADAPT, variando o parâmetro BOF em 0, 5 e 10 respectivamente. Em cada um dos gráficos está presente o triangulo superior da matriz que combina os níveis de QLF e ILF relevantes. Nos Gráficos 4.6(a), 4.7(a) e 4.8(a) são exibidos os resultados linha de base e, cada célula contém a configuração (dentre as 11 configurações estática definidas) que alcançou o melhor tempo de resposta médio naquele cenário, bem como o valor desta métrica. Por sua vez, nos demais Gráficos, em cada célula está presente apenas o tempo de resposta médio alcançado por MS-ADAPT.

Diretamente é possível observar que nos gráficos linha de base, em cada uma das variações QLFxIFL (células), houve alternância da configuração estática (ou *Lazy*) que alcançou melhor desempenho. Esse aspecto deixa claro que a escolha de uma configuração de paralelismo estática proporciona desempenho subótimo em contextos de aplicações onde a intensidade dos fluxos de entrada variam. Pode-se observar, e conforme era esperado, ao passo que a intensidade de um fluxo aumenta, as configurações que alocam mais recursos para a execução das tarefas desse contexto, se mostram mais adequadas. É o caso da configuração [50][1][10], na qual, para todos os casos, se destacou quando QLF é igual ou superior a 0.6 e o ILF é 0.2. Por sua vez, a configuração [20][1][40] alcançou a melhor performance em cenários opostos: onde ILF é alto (0.6 e 0.8) e QLF é baixo (0.2). Indo além, foi possível perceber que a abordagem *Lazy Insertions* foi referência quando ambas intensidade dos fluxos são altas. Nestes contextos, é mais promissor a alocação de todos os recursos possíveis para a otimização da vazão de tarefas, realizando inserção sob demanda, do que balancear recursos entre os dois fluxos sob pena das penalizações pela desatualização dos *bucktes* (que se tornam mais frequentes com ILF altos).

A proposta MS-ADAPT manteve, em todos os casos, o tempo de resposta médio próximo ou superior a melhor configurações estática. A habilidade adaptativa de MS-ADAPT, provê a identificação dinâmica da variação do fator de carga dos fluxos, por meio do monitoramento sugerido, para que, dentre o conjunto de possibilidades de configurações de paralelismo, a mais adequada ao contexto seja implantada.

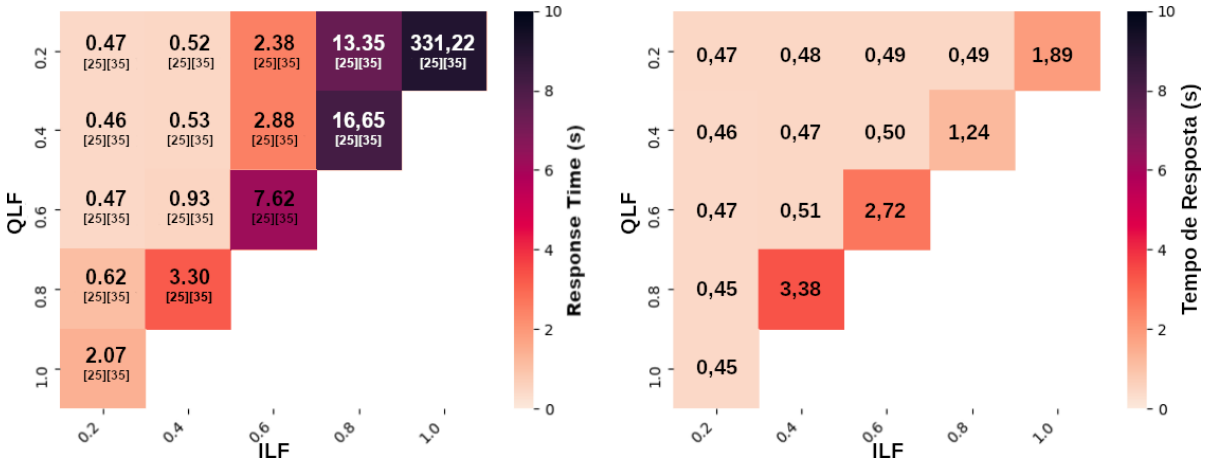
É importante destacar que, mesmo em uma combinação QLFxILF existem sutis variações da intensidade dos fluxos dado a chegada das tarefas seguirem uma distribuição Poisson. Dito isto, mesmo a melhor configurações estática ainda sim pode ser uma solução subótima. MS-ADAPT é capaz de se adaptar a essas sutis variações, e nesses casos podemos observar que o tempo de resposta médio foi superior às melhores configurações



(a) Melhor configuração estática: 20 e 40 threads para o processamento de consultas e inserções, respectivamente.

(b) MS-ADAPT

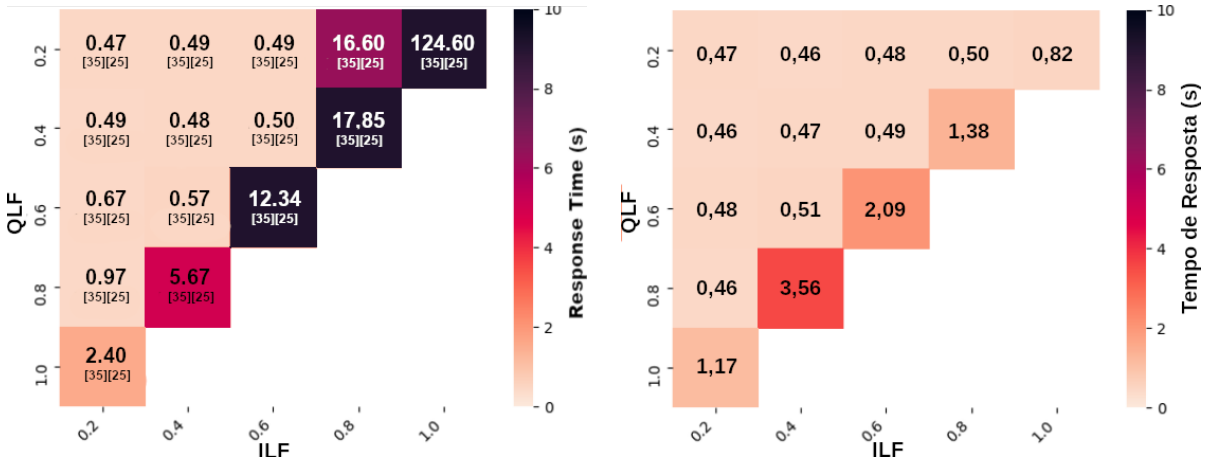
Figura 4.6: Comparação entre configurações estáticas linha de base e a abordagem MS-ADAPT com BOF=0.



(a) Melhor configuração estática: 25 e 35 threads para o processamento de consultas e inserções, respectivamente.

(b) MS-ADAPT

Figura 4.7: Comparação entre configurações estáticas linha de base e a abordagem MS-ADAPT com BOF=5.



(a) Melhor configuração estática: 35 e 25 threads para o processamento de consultas e inserções, respectivamente.

(b) MS-ADAPT

Figura 4.8: Comparação entre configurações estáticas linha de base e a abordagem MS-ADAPT com BOF=10.

estáticas. Uma forma de ilustrar o comportamento da proposta MS-ADAPT está na Figura 4.9. Neste exemplo pontual, por meio do cenário QLF=0.4 , ILF=0.4 e BOF=0.5, é possível notar que nos instantes iniciais da execução o comportamento de MS-ADAPT leva-o a reconfigurações até alcançar a faixa da melhor configuração (faixa cinza representando a configuração [40][1][20]). Apesar de se manter pela maior parte do tempo na faixa da configuração de referência, houveram momentos onde MS-ADAPT reconfigurou para níveis de paralelismo marginais, em reflexo a sutis variações da intensidade de QLF=0.4 e ILF=0.4. Nesse exemplo, MS-ADAPT superou a melhor configuração estática.

O impacto do parâmetro de flexibilidade de desatualização (BOF) pode ser observado claramente comparando os gráficos presentes nas Figuras 4.6, 4.7 e 4.8. De maneira geral, à medida que o intervalo BOF se torna mais restrito, o sistema é obrigado a lidar de forma mais intensiva com as tarefas de inserção, a fim de evitar que a desatualização dos *buckets* prejudique a eficiência da execução das consultas.

Consequentemente, as configurações estáticas que alocam uma proporção maior de recursos para o fluxo de inserção tendem a apresentar um desempenho superior em cenários com intervalos BOF mais restritos. Isso ocorre porque uma alocação mais robusta de recursos para a inserção permite manter os *buckets* atualizados e prontos para atender às consultas, minimizando o impacto da desatualização dos dados sobre o tempo de resposta e a vazão das operações de busca.

Em resumo, a variação do parâmetro BOF influencia a distribuição dos recursos entre as operações de inserção e consulta, e essa distribuição tem um impacto direto no desempenho global do sistema, especialmente em contextos onde a desatualização dos *buckets* pode se tornar um fator limitante para o desempenho das consultas.

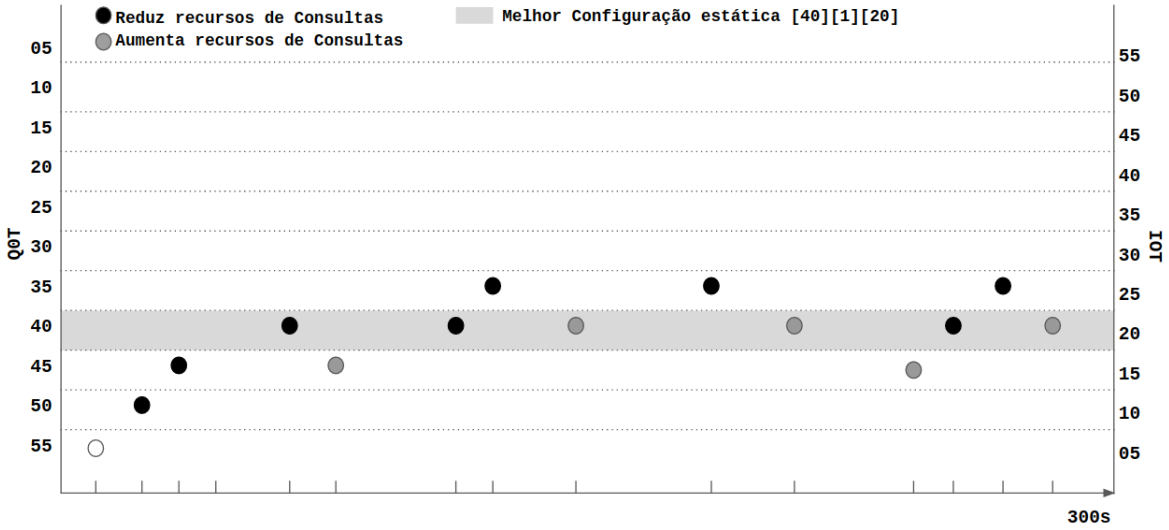
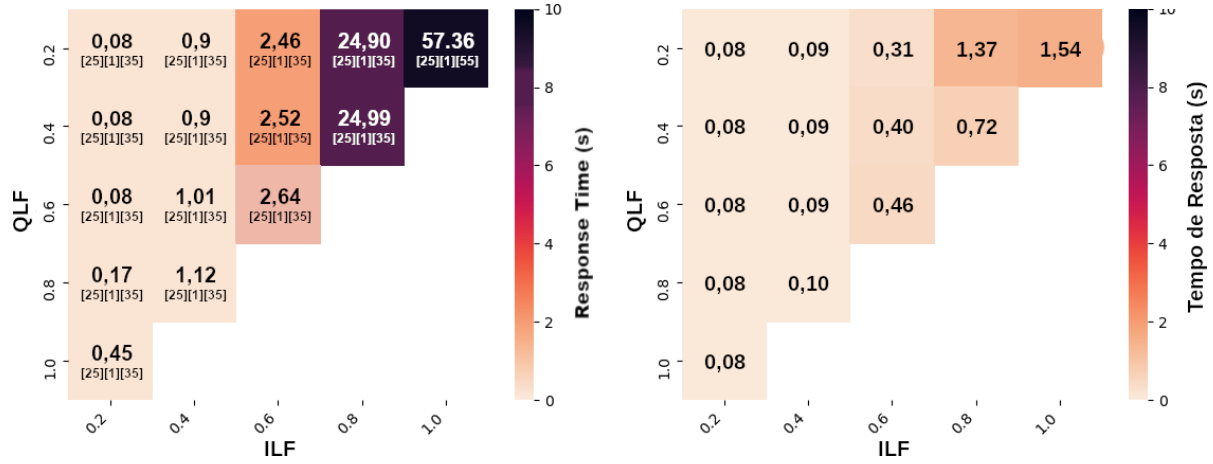


Figura 4.9: Exemplo de caminhamento performado pela abordagem MS-ADAPT. Neste cenário a configuração estática referência é [40][1][20], pode-se perceber que a estratégia de adaptação dinâmica executa diferentes reconfigurações alcançando a faixa da melhor configuração estática.

4.4.2 Avaliação Execução Distribuída

Os resultados discutidos anteriormente mostraram o impacto da adaptação dinâmica de paralelismo no tempo de resposta médio de tarefas em um único Processador de Consulta. Nesta subseção será apresentado a execução de IVFADC em máquina de memória distribuída com vários nós Processadores de Consulta executando a otimização proposta. O intuito é discutir a contribuição global das otimizações locais no processamento de tarefas de consultas. Para isso, inicialmente foi considerada uma máquina com 40 nós de processamento, sendo 8 Coordenadores e 32 Processadores de Consulta, cada nó está equipado com CPU AMD EPYC 7742 rodando 64 cores a 2.25-3.40 GHz e 256GB de memória RAM. Foi distribuído 1 bilhão de descritores SIFT entre Processador de Consulta, configurados executando IVFADC com $C_c = 8192$ e $w = 8$, além de Índices Temporais instanciados com $T = 4$ e $s = 360$ e estratégia de distribuição SABES, mantendo a qualidade da busca equivalente a recall 1@100 de $58,3\%$. Foram repetidos os cenários de testes anterior, considerando a variação QLF x ILF proposta e BOF=5, e mensurado o tempo de resposta de tarefas de consulta no Coordenador. A avaliação experimental considera novamente as configurações estática e abordagem *Lazy Insertions* como linha de base. O resultado está presente na Figura 4.10. É possível observar os compromissos discutidos anteriormente, onde a solução MS-ADAPT alcançou tempo de resposta médio próximo ou superior á melhor configuração estática. Com esse resultado conclui-se que as otimizações do processo de busca local performado em cada Processador de Consulta,

por meio a adaptação dinâmica de paralelismo, reflete em melhoria no desempenho global da máquina de busca.



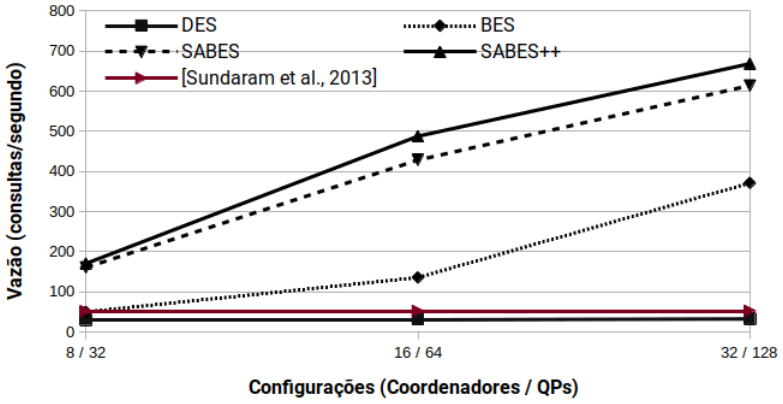
(a) Configurações Estática (b) MS-ADAPT

Figura 4.10: Comparação entre as configurações de paralelismo linha de base e a abordagem MS-ADAPT em uma máquina de memória distribuída com 40 nós.

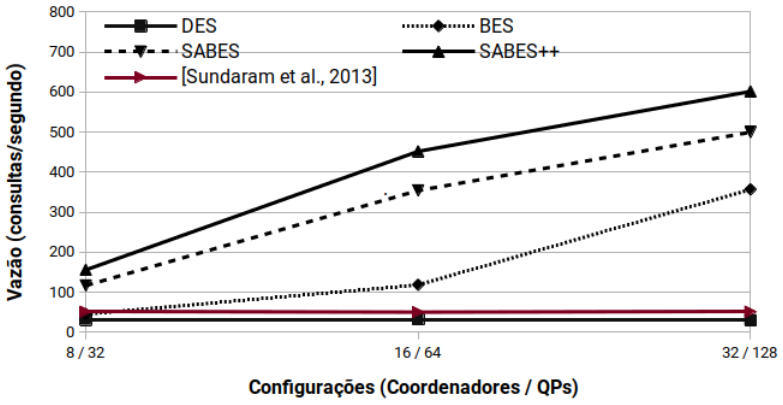
4.4.3 Avaliação de Escalabilidade

Foi avaliado a escalabilidade da máquina em memória distribuída executando IV-FADC e MS-ADAPT em cada nó Processador de Consulta. Neste experimento, foi executado teste *weak scaling*, no qual o conjunto de dados indexados aumenta proporcionalmente ao número de nós de computação. Dessa maneira, a máquina em memória distribuída foi instanciada com os seguintes recursos: (i) 40 nós (8 Co; 32 QPs) e 86 bilhões de descritores SIFT; (ii) 80 nós (16 Co; 64 QPs) e 172 bilhões de descritores SIFT; (iii) 160 nós (32 Co; 128 QPs) e 344 bilhões de descritores SIFT. As configurações locais de cada Processador de Consultas segue o que foi descrito na Subseção 4.4.2 ($C_c = 8192$ e $w = 8$), entretanto, foram variadas as estratégias de distribuição de dados (DES, BES, SABES e SABES++). Além disso, foi considerado três cenários diferentes de intensidade de fluxos de dados, onde o fator de carga de inserções cresce (ILF=0.2, 0,4 e 0,6). Para fins comparativos foi executado, nas mesmas condições, a estratégia LSH descrita em [49], com $\eta = 10\%$ e $M = 4$ (vide Subseção 2.3.3). A Figura 4.11, mostra a escalabilidade da aplicação nos conjuntos de testes propostos.

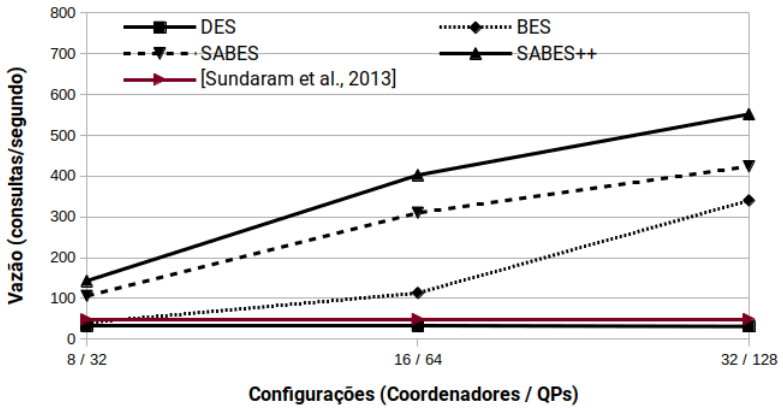
Conforme é possível observar, a escalabilidade da máquina em memória distribuída se mantém como esperado e discutido na Subseção 3.3.4. Interessantemente, ao passo que a intensidade do fluxo de inserções cresce, o impacto na redução da vazão de tarefas é mais



(a) QLF=1.0 e ILF=0.2



(b) QLF=1.0 e ILF=0.4



(c) QLF=1.0 e ILF=0.6

Figura 4.11: Vazão de tarefas da abordagem IVFADC (consultas / s) em uma avaliação *weak scaling*, onde foram variadas as estratégias de particionamento de dados e também a intensidade do fluxo de inserções. Neste experimento cada Processador de Consultas está otimizado com a adaptação dinâmica de paralelismo por meio da solução MS-ADAPT.

intenso na estratégia de distribuição SABES. Esse aspecto está diretamente relacionado ao desbalanceamento proporcionado por essa abordagem, que se potencializa quando mais descritores são indexados. Além disso, a solução LSH [49] possui comportamento e desempenho similares à execução por meio do particionamento DES, já que, em essência, também se beneficia do particionamento igualitário de dados. Entretanto, a diferença de desempenho entre essas abordagens se dá pelo fato das inserções, na solução LSH, acontecerem em uma estrutura de dados (Delta Table) separada da estrutura para consultas e, esse aspecto, elimina sincronizações necessárias a cada consulta desempenhada na abordagem IVFADC com DES, onde todos os nós precisam ser envolvidos e em cada um deles existirá eventuais bloqueios de sincronização por consultas e inserções compartilharem a mesma estrutura de dados. A solução LSH propõe agregações (entre Delta e Static Tables) com maior volume de dados em intervalos de tempo mais espaçados, o que nos mostra que, quando a distribuição de dados é igualitária, este mecanismo tende a ser mais eficiência que inserções e consultas acontecerem em um mesmo índice.

4.5 Sumário

Neste Capítulo foi apresentado as soluções proposta para endereçar os desafios do cenário de busca aproximada dos k-vizinhos mais próximos em conjunto de dados dinâmicos.

Apresentou-se, em um primeiro momento, a descrição do Índice Temporal. Esta estrutura para indexação local organiza os descritores pela proximidade espacial e também pelo instante temporal em que foram indexados, por meio de *buckets temporais*. Foi descrito os mecanismos que tornam o Índice Temporal capaz de suportar indexações de novos descritores e consultas simultaneamente, bem como o descarte de objetos antigos. O custo das sincronizações necessárias representam apenas $0,67\%$ do tempo total de execução da busca, ao passo que abordagens na literatura apresentam soluções com custo próximo de $1,17\%$. Indo além, as execuções de tarefas de consulta e tarefas de inserções podem ser potencializadas por meio de níveis de paralelismo interno e externo no Índice Temporal. E, dessa maneira, foi proposto um componente Controlador de Fluxos, capaz de ajustar dinamicamente estes níveis a fim de encontrar a combinação adequada dado a intensidade dos fluxos de entrada.

Com isso, foi descrito uma estratégia de adaptação dinâmica de paralelismo implementada no Controlador de Fluxo. A abordagem MS-ADAPT monitora diferentes métricas do sistema que apontam a intensidade dos fluxos de entrada. Por meio dessas métricas, a solução encontra a configuração de paralelismo que potencialmente minimize

o tempo de resposta das tarefas de consultas. Para a avaliação experimental foram usadas configurações estáticas de paralelismo e também uma abordagem *Lazy Insertions* como linha de base, avaliou-se o tempo de resposta médio das em cenários com diferentes intensidades de chegada de tarefas. De forma ampla, os resultados discutidos apontam que soluções estáticas para a configuração dos níveis de paralelismo do Processador de Consultas são subótimos dado a variabilidade do fator de carga dos fluxos de consultas e inserções, ao passo que a solução MS-ADAPT consegue se adaptar dinamicamente, alcançando tempo de resposta médio próximo ou superior à melhor configuração de paralelismo estática em cada cenário proposto. Por fim, foi mostrado que as otimizações proporcionadas pelo MS-ADAPT em cada nó de processamento reflete na performance de todo o sistema de busca, bem como mantém a escalabilidade do sistema.

As contribuições apresentadas neste capítulo endereçam desafios identificados no contexto de busca kNN aproximada em memória distribuída com conjunto de dados dinâmicos, e além disto, abrem a possibilidades para novas investigações e aprofundamento por meio dos componentes apresentados e dos compromissos discutidos em cada ponto deste trabalho de tese. No Capítulo 5, serão descritas estas oportunidades e a importância delas.

Capítulo 5

Conclusões

Content-Based Multimedia Retrieval (CBMR) é uma operação chave para vários serviços e aplicativos que se baseiam na coleta e tratamento de informações vindas de diferentes fontes da Web, em que o objeto principal é encontrar um conjunto de objetos similares a um objeto consulta. Devido à grande quantidade de dados multimídia disponíveis, por exemplo, em redes sociais e plataformas de streaming de vídeo, as aplicações CBMR precisam ser capazes de indexar relevante volume de dados enquanto são submetidos a intensos fluxos de consultas simultâneas.

Ao longo do tempo, a busca por objetos similares em aplicações CBMR recebeu relevantes contribuições que propuseram heurísticas aproximadas (ANNS) e tornaram possível a execução eficiente desta operação em massivos conjuntos de dados multimídia e em alta dimensões. Durante o Capítulo 2, essas contribuições foram amplamente discutidas e destacam-se aquelas que combinaram diferentes abordagens para a poda do espaço de busca e também maneiras para a redução da representação vetorial dos dados multimídia, sem perder a representatividade espacial dos mesmos. Dentre essas estão Locality Sensitive Hashing (LSH) [24], FLANN (fast library for approximate nearest neighbors) [39] e PQANNS [30]. Porém, foi visto que, mesmo usando estas sofisticadas abordagens para ANNS, o desempenho computacional estava limitado ao poder de processamento de uma única máquina e, no entanto, as demandas intensas das aplicações web recentes superam as capacidades de um único nó de processamento. Por meio desta observação, motivaram-se os recentes desenvolvimentos de algoritmos e sistemas para a busca aproximada em máquinas com memória distribuída [48, 6, 38, 34, 39, 51, 3].

Ainda durante o Capítulo 2, as contribuições paralelas e em memória distribuída para ANNS foram organizadas em dois grupos por meio das características do conjunto de dados em que a busca acontece e na forma como os fluxos de inserções e consultas são tratados. O primeiro deles compreende o conjunto de dados como sendo estático e, durante a discussão apresentada na Subseção 2.3.2.3, foram expostos problemas nas abordagens presentes na literatura relacionados à organização espacial dos objetos da base, bem como no balanceamento de carga entre os nós de processamento. Por sua vez, no segundo grupo, o conjunto de objetos de busca pode crescer ao longo do ciclo de vida da aplicação CBMR e, sendo assim, na Subseção 2.3.3.2 apontaram-se desafios relacionados à gestão dos fluxos

concorrentes de consultas e inserções, assim como na organização dinâmica dos recursos computacionais envolvidos na máquina de memória distribuída. Todos esses aspectos motivaram as contribuições deste trabalho de tese que foram desenvolvidas e apresentadas ao longo dos Capítulos 3 e 4.

Durante o Capítulo 3, foi investigado e avaliado sistematicamente abordagens de paralelização em memória distribuída disponíveis na literatura e foram propostas estratégias novas e mais eficientes. Para atingir esse objetivo, a primeira contribuição foi a proposta e implementação de uma arquitetura paralela e em memória distribuída que apresente componentes capazes de suportar o desenvolvimento e implantação de algoritmos ANNS e, também, a implementação de diferentes abordagens de particionamento de dados entre os nós de processamento. Por meio dessa arquitetura - tomando como base o algoritmo IVFADC para ANNS -, foram comparados métodos de particionamento de dados presentes na literatura (DES e BES) e foram propostas novas estratégias para esse propósito (SABES e SABES++). Os resultados demonstraram que a abordagem SABES++ alcançou uma melhoria de desempenho de 4,2×, 1,8×, respectivamente, ao comparada com DES e BES. Esses ganhos são reflexos da capacidade das estratégias propostas em melhorar a localidade dos dados e, consequentemente, o processamento da busca local nos nós de processamento. Além disso, a variação do número de nós de computação usados mostrou que BES, SABES e SABES++ têm eficiência superlinear ao observar a escalabilidade fraca (*weak scaling*), o que, por sua vez, faz com que os ganhos de desempenho sejam cada vez maiores, em comparação à DES, quando a quantidade de nós de processamento aumenta.

Extensivamente, durante o Capítulo 4 foram endereçados os desafiadores aspectos de tempo real de aplicações CBMR, em que a base de dados cresce ao longo do ciclo de vida da aplicação. Nesse contexto, fluxos de inserções de novos descritores executam simultaneamente com fluxos intensos de consultas. A arquitetura proposta durante o Capítulo 3 foi implementada para suportar a execução desses fluxos concorrentes, bem como possibilitar a implementação de maneiras para a gestão adaptativa dos recursos computacionais necessários para o processamento destas cargas de entrada. As contribuições presentes no Capítulo 4 incluem a elaboração da organização dos dados por meio das características espaciais, como já estava sendo feito, associado às características temporais dos mesmos, possibilitando que consultas e inserções aconteçam em diferentes partições do conjunto de objetos simultaneamente. Além disso, foi proposto um algoritmo para a adaptação dinâmica da quantidade de recursos paralelos necessários para lidar com os diferentes fluxos. Essa técnica, MS-ADAPT, identifica variações na intensidade dos fluxos de entrada e escolhe a melhor configuração dos níveis de paralelismo a ser empregada. Os resultados discutidos apontam que soluções estáticas para a configuração dos níveis de paralelismo em nós de processamento são subótimos dado a variabilidade do fator de carga dos fluxos de consultas e inserções, ao passo que a solução MS-ADAPT consegue

se adaptar dinamicamente, alcançando tempo de resposta médio próximo ou superior à melhor configuração de paralelismo estática em cada cenário proposto.

Por fim, conforme observado ao longo deste trabalho de tese, a busca aproximada por similaridade em grandes volumes de objetos multimídia agrega desafiadores aspectos computacionais. Por ser uma operação chave em aplicações CBMR o desempenho e qualidade do resultado da busca são características extremamente importantes e de relevância como objeto de pesquisa e desenvolvimento. Durante este trabalho, procurou-se estudar, entender e propor soluções que, por meio da exploração de características do conjunto de objetos em alta dimensões e do perfil dos fluxos de dados de entrada, aumentassem o poder de aplicação da busca aproximada em grandes volumes de dados crescentes. As contribuições apresentadas superam, pelo melhor esforço do autor desta tese, o estado da arte encontrado na literatura e acrescentam análises e observações que impulsionam a discussão de outros aspectos relevantes no contexto. Essas novas discussões abrem a possibilidades de extensa análise e desenvolvimento científico, conforme será brevemente discutido na Seção em sequência.

5.1 Oportunidades Futuras

Durante o desenvolvimento deste trabalho de tese, algumas oportunidades de extensões foram identificadas como trabalhos futuros no domínio do problema.

1. **Processamento GPU *intra-node***. Os nós de processamento da máquina em memória distribuída (QPs) se beneficiam do processamento de CPUs com múltiplos núcleos para a execução da busca local. Como uma alternativa e, potencialmente, com uma oportunidade de intensificar os ganhos de desempenho dessa operação, é possível tirar proveito do paralelismo massivo presentes em GPUs. Acredita-se que as estratégias de distribuição do conjunto de objetos, propostas durante o Capítulo 3, conduzem à uma organização local de dados que seria mais apropriada de ser consultada por meio do paralelismo intenso de GPUs, já que, nessas abordagens, busca-se em um conjunto menor de nós de processamento, porém em maiores conjuntos locais de dados. Dessa maneira, a densidade computacional da busca local é mais expressiva e mais adequada para ser explorada pelo processamento paralelo massivo das GPUs, em que é possível, inclusive, executar pacotes de consultas (*batches*) variáveis dado a intensidade do fluxo de consultas no instante.

2. **Computação Heterogênea para a Busca Local**. Como extensão à oportunidade descrita anteriormente, acredita-se que a combinação entre o processamento

em CPU e o processamento em GPU para a busca local, pode se tornar uma atrativa possibilidade em extrair o máximo dos recursos computacionais presentes na arquitetura paralela e em memória distribuída. O uso coordenado dessas duas unidades de processamento, alia-se à proposta de uma decisão inteligente e dinâmica para direcionar o fluxo de processamento de consultas: ora para CPU, ora para GPU.

3. **Balanceamento de Carga**. As discussões conduzidas durante o Capítulo 3 identificaram que as abordagens propostas SABES e SABES++, proporcionam organizações do conjunto de objetos minimizando a quantidade de nós de processamento necessários para realizar uma busca, entretanto podem ocasionar em um expressivo desbalanceamento na quantidade de objetos indexados localmente em cada nó de processamento. Como uma possibilidade de extensão com relevante importância, aponta-se a elaboração, em conjunto com SABES e SABES++, de mecanismos de balanceamento de carga, combinando este aspecto com a localidade espacial dos dados durante o particionamento entre os nós de computação.

4. Demais possibilidades de trabalhos futuros podem ser endereçadas nos componentes: i) novas estratégias de distribuição de dados e; ii) algoritmos para a adaptação dinâmica de paralelismo nos nós de processamento. Para essas operações, outros aspectos relevantes do sistema podem ser considerados em contexto de dados específicos, trazendo contribuições de desempenho para aplicações de busca aproximada implementadas por meio da arquitetura proposta. Além disso, outros algoritmos ANNS podem ser amplamente avaliados e comparados, tendo a infraestrutura proposta como base.

Referências

[1] Maha Ahmed Alabduljalil, Xun Tang, and Tao Yang. Optimizing Parallel Algorithms for All Pairs Similarity Search. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 203–212, New York, NY, USA, 2013. Association for Computing Machinery.

[2] Flora Amato, Luca Greco, Fabio Persia, Silvestro Roberto Poccia, and Aniello De Santo. *Content-Based Multimedia Retrieval*, pages 291–310. Springer International Publishing, Cham, 2015.

[3] Guilherme Andrade, André Fernandes, Jeremias M. Gomes, Renato Ferreira, and George Teodoro. Large-scale parallel similarity search with Product Quantization for online multimedia services. *Journal of Parallel and Distributed Computing*, 125:81 – 92, 2019.

[4] Guilherme Andrade, George Teodoro, and Renato Ferreira. Online multimedia similarity search with response time-aware parallelism and task granularity auto-tuning. In *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 153–160. IEEE, 2017.

[5] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[6] Bahman Bahmani, Ashish Goel, and Rajendra Shinde. Efficient Distributed Locality Sensitive Hashing. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM)*, pages 2174–2178, 2012.

[7] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[8] Alina Beygelzimer, Sham Kakade, and John Langford. Cover Trees for Nearest Neighbor. In *Proceedings of the 23rd International Conference on Machine learning*, ICML '06, pages 97–104, 2006.

[9] Filip Blagojevic, Dimitrios S Nikolopoulos, Alexandros Stamatakis, Christos D Antonopoulos, and Matthew Curtis-Maury. Runtime scheduling of dynamic parallelism on accelerator-based multi-core systems. *Parallel Computing*, 33(10-11):700–719, 2007.

[10] Sergey Brin. Near neighbor search in large metric spaces. *Proceedings of the 21th International Conference on Very Large Data Bases*, page 574–584, 1995.

[11] Edgar Chávez and Gonzalo Navarro. A Compact Space Decomposition for Effective Metric Indexing. *Pattern Recogn. Lett.*, 26(9):1363–1376, July 2005.

[12] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The mpi message passing interface standard. Technical report, Basel, 1994.

[13] Roland Coghetto. Chebyshev distance. *Formalized Mathematics*, 24, 06 2016.

[14] Per-Erik Danielsson. Euclidean distance mapping. *Computer Graphics and image processing*, 14(3):227–248, 1980.

[15] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[16] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[17] Yang Ding, Mahmut Kandemir, Padma Raghavan, and Mary Jane Irwin. Adapting application execution in cmps using helper threads. *Journal of Parallel and Distributed Computing*, 69(9):790–806, 2009.

[18] Matthijs Douze, Hervé Jégou, Harsimrat Sandhawalia, Laurent Amsaleg, and Cordelia Schmid. Evaluation of GIST Descriptors for Web-scale Image Search. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, CIVR '09, pages 19:1–19:8, New York, NY, USA, 2009. ACM.

[19] Renato Ferreira, Wagner Meira Jr., Dorgival Olavo Guedes Neto, Lúcia Maria de A. Drummond, Bruno Coutinho, George Teodoro, Tulio Tavares, Renata Braga Araújo, and Guilherme T. Ferreira. Anthill: A Scalable Run-Time Environment for Data Mining Applications. In *17th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2005), 24-27 October 2005, Rio de Janeiro, Brazil*, pages 159–167. IEEE Computer Society, 2005.

[20] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977.

[21] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[22] Keinosuke Fukunaga and Patrenahalli M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE transactions on computers*, 100(7):750–753, 1975.

[23] V. Gil-Costa and M. Marin. Load Balancing Query Processing in Metric-Space Similarity Search. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 368–375, 2012.

[24] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[25] Yunchao Gong, Liwei Wang, Ruiqi Guo, and Svetlana Lazebnik. Multi-scale Orderless Pooling of Deep Convolutional Activation Features. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 392–407, Cham, 2014. Springer International Publishing.

[26] Gylfi Pór Gudmundsson, Björn Pór Jónsson, Laurent Amsaleg, and Michael J. Franklin. Prototyping a Web-Scale Multimedia Retrieval Service Using Spark. *ACM Trans. Multimedia Comput. Commun. Appl.*, 14(3s), June 2018.

[27] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350, 2012.

[28] Alexander Hinneburg, Charu C Aggarwal, and Daniel A Keim. What is the nearest neighbor in high dimensional spaces? In *26th Internat. Conference on Very Large Databases*, pages 506–515, 2000.

[29] Tommi S Jaakkola, David Haussler, et al. Exploiting generative models in discriminative classifiers. *Advances in neural information processing systems*, pages 487–493, 1999.

[30] H. Jegou, M. Douze, and C. Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2010.

[31] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid. Aggregating Local Image Descriptors into Compact Codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1704–1716, Sep. 2012.

[32] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *European conference on computer vision*, pages 304–317. Springer, 2008.

[33] Kittisak Kerdprasop, Nittaya Kerdprasop, and Pairote Sattayatham. Weighted K-Means for Density-Biased Clustering. In A. Min Tjoa and Juan Trujillo, editors, *Data Warehousing and Knowledge Discovery*, pages 488–497, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[34] Martin Kruliš, Tomáš Skopal, Jakub Lokoč, and Christian Beecks. Combining CPU and GPU architectures for fast similarity search. *Distributed and Parallel Databases*, 30(3):179–207, Aug 2012.

[35] Yossi Lev, Victor Luchangco, and Marek Olszewski. Scalable reader-writer locks. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 101–110, 2009.

[36] Tony Lindeberg. Scale invariant feature transform. *Scholarpedia*, 7, 05 2012.

[37] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[38] Diana Moise, Denis Shestakov, Gylfi Gudmundsson, and Laurent Amsaleg. Indexing and Searching 100M Images with Map-reduce. In *Proceedings of the 3rd ACM Conference on International Conference on Multimedia Retrieval*, ICMR '13, pages 17–24, 2013.

[39] M. Muja and D. G. Lowe. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, Nov 2014.

[40] Marius Muja. *Scalable nearest neighbour methods for high dimensional data*. PhD thesis, University of British Columbia, 2013.

[41] Marius Muja and David Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP 2009 - Proceedings of the 4th International Conference on Computer Vision Theory and Applications*, 1:331–340, 01 2009.

[42] Marco Patella and Paolo Ciaccia. Approximate similarity search: A multi-faceted problem. *Journal of Discrete Algorithms*, 7(1):36–48, 2009.

[43] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[44] Chanop Silpa-Anan and Richard Hartley. Optimised KD-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 06 2008.

[45] Archana Singh, Avantika Yadav, and Ajay Rana. K-means with three different distance metrics. *International Journal of Computer Applications*, 67(10), 2013.

[46] Deepak Sinwar and Rahul Kaushik. Study of euclidean and manhattan distance metrics using simple k-means clustering. *Int. J. Res. Appl. Sci. Eng. Technol*, 2(5):270–274, 2014.

[47] Rafael Souza, André Fernandes, Thiago S. F. X. Teixeira, George Teodoro, and Renato Ferreira. Online multimedia retrieval on CPU-GPU platforms with adaptive work partition. *J. Parallel Distributed Comput.*, 148:31–45, 2021.

[48] Aleksandar Stupar, Sebastian Michel, and Ralf Schenkel. RankReduce - processing K-Nearest Neighbor queries on top of MapReduce. In *Proceedings of the 8th Workshop on Large-Scale Distributed Systems for Information Retrieval (LSDS-IR'10)*, pages 1–6, 2010.

[49] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. Streaming Similarity Search over One Billion Tweets Using Parallel Locality-Sensitive Hashing. *Proc. VLDB Endow.*, 6(14):1930–1941, September 2013.

[50] Xun Tang, Maha Alabduljalil, Xin Jin, and Tao Yang. Load Balancing for Partition-Based Similarity Search. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '14, pages 193–202, New York, NY, USA, 2014. Association for Computing Machinery.

[51] George Teodoro, Eduardo Valle, Nathan Mariano, Ricardo Torres, Wagner Meira, and Joel H. Saltz. Approximate similarity search for online multimedia services on distributed CPU–GPU platforms. *The VLDB Journal*, 23(3):427–448, Jun 2014.

[52] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. Storm@twitter. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 147–156, New York, NY, USA, 2014. Association for Computing Machinery.

[53] Ji Wan, Dayong Wang, Steven Chu Hong Hoi, Pengcheng Wu, Jianke Zhu, Yongdong Zhang, and Jintao Li. Deep Learning for Content-Based Image Retrieval: A Comprehensive Study. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 157–166, New York, NY, USA, 2014. ACM.

[54] Zheng Wang and Michael FP O'Boyle. Mapping parallelism to multi-cores: a machine learning based approach. In *Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 75–84, 2009.

[55] Roger Weber and Klemens Böhm. Trading quality for time with nearest-neighbor search. In *International Conference on Extending Database Technology*, pages 21–35. Springer, 2000.

[56] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[57] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, volume 98, pages 194–205, 1998.

[58] Yair Weiss, Antonio Torralba, Robert Fergus, et al. Spectral hashing. In *Nips*, volume 1, page 4. Citeseer, 2008.

[59] Peipei Xia, Li Zhang, and Fanzhang Li. Learning similarity with cosine similarity ensemble. *Information Sciences*, 307:39–52, 2015.

[60] Keyu Yang, Xin Ding, Yuanliang Zhang, Lu Chen, Baihua Zheng, and Yunjun Gao. Distributed similarity queries in metric spaces. *Data Science and Engineering*, 4:1–16, 06 2019.

[61] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, page 10, 2010.

[62] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity search: the metric space approach*, volume 32. Springer Science & Business Media, 2006.

[63] Pavel Zezula, Pasquale Savino, Giuseppe Amato, and Fausto Rabitti. Approximate similarity retrieval with m-trees. *The VLDB Journal*, 7(4):275–293, 1998.