UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Uma Abordagem de Processamento de Consultas para Plataformas de Middleware Distribuído e Particionado no Contexto de Cidades Inteligentes

JÚLIO ZINGA SUZUKI LOPES

JOÃO PESSOA-PB
Agosto -2019

Uma Abordagem de Processamento de Consultas para Plataformas de Middleware Distribuído e Particionado no Contexto de Cidades Inteligentes

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba como requisito parcial para obtenção do título de Mestre em Informática.

Área de Concentração: Computação Distribuída

Orientador: Prof. Dr. Gledson Elias.

JOÃO PESSOA-PB

Agosto-2019

Catalogação na publicação
Seção de Catalogação e Classificação

L864a Lopes, Julio Zinga Suzuki.
Uma Abordagem de Processamento de Consultas para Plataformas de Middleware Distribuído e Particionado no Contexto de Cidades Inteligentes / Julio Zinga Suzuki Lopes. - João Pessoa, 2019.
105 f. : il.

Dissertação (Mestrado) - UFPB/CI.

1. Cidade Inteligente. 2. Middleware. 3. Sistema Distribuído. 4. Processamento de Consulta. 5. Linguagem de Consulta de Contexto. I. Título

UFPB/BC

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Julio Zinga Suzuki Lopes, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 30 de agosto de 2019.

Aos trinta dias do mês de agosto, do ano de dois mil e dezenove, às quatorze horas e trinta minutos, no Centro de Informática da Universidade Federal da Paraíba, em Mangabeira, reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do Sr. Julio Zinga Suzuki Lopes, vinculado a esta Universidade sob a matrícula nº 20171004579, candidato ao grau de Mestre em Informática, na área de "Sistemas de Computação", na linha de pesquisa "Computação Distribuída", do Programa de Pós-Graduação em Informática, da Universidade Federal da Paraíba. A comissão examinadora foi composta pelos professores: Gledson Elias da Silveira (PPGI-UFPB) Orientador e Presidente da Banca, Gustavo Henrique Matos Bezerra Motta (PPGI-UFPB), Examinador Interno, Denio Mariz Timoteo de Sousa (IFPB), Examinador Externo à Instituição. Dando início aos trabalhos, o Presidente da Banca cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse a exposição oral do trabalho de dissertação intitulado: "Uma Abordagem de Processamento de Consultas para Plataformas de Middleware Distribuído e Particionado no Contexto de Cidades Inteligentes". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "***aprovado***". Do ocorrido, eu, Alisson Vasconcelos de Brito, Vice-Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai assinada por mim e pelos membros da banca examinadora. João Pessoa, 30 de agosto de 2019.

Prof. Dr. Alisson Vasconcelos de Brito

Prof. Gledson Elias da Silveira
Orientador (PPGI-UFPB)

Prof. Gustavo Henrique Matos Bezerra Motta
Examinador Interno (UFPB)

Prof. Denio Mariz Timoteo de Sousa
Examinador Externo à Instituição (IFPB)

Dedico este estudo aos meus pais, Salvador Lopes e Lia Suzuki.

AGRADECIMENTOS

Aos meus pais, Salvador Lopes e Lia Suzuki, pela dedicação e ensinamentos transmitidos através do exemplo e de princípios.

Ao orientador, Professor Gledson Elias, pelo incentivo, disponibilidade, paciência e valiosas contribuições.

À minha esposa pelo apoio, solidariedade, paciência e cuidado necessários nessa caminhada. Em quem me inspiro para alcançar objetivos e superar obstáculos.

À minha filha pelos momentos de diversão e descontração, que me proporcionaram momentos revigorantes durante os intervalos da pesquisa.

Às minhas irmãs, Iacy Luana e Aruanda Suzuki, pelo imenso apoio e inspiração em ultrapassar limites e como ser um contínuo aprendiz.

À família de Souza Almeida, em especial à Graça de Oliveira, sogra e avó atenciosa, que sempre esteve presente em incontáveis momentos.

À equipe de docentes e técnico-administrativos que formam o Centro de Informática e o PPGI, em especial aos membros da banca de qualificação e de defesa. A(o)s professores(as) Natasha Lino, Gustavo Motta, Ruy Altafim e Fernando Matos, pelos ensinamentos ministrados e conhecimento partilhado nas disciplinas.

Aos amigos: Mayrton Dias, Felipe Melo, Lucas Lopes, Evilásio Silva e Miguel Massera.

Ao amigo Lucas Vale, que participou de várias etapas deste curso de mestrado. Criativo e dedicado à aprendizagem de novos conteúdos, foi um grande parceiro de trabalho.

À equipe da Escola Superior de Redes, Uirá Paiva e Katherin Moneta, pela ajuda e colaboração.

Ao Instituto Federal de Educação da Paraíba (IFPB), em especial ao Departamento de Tecnologia da Informação, que apoiaram a licença de qualificação com afastamento integral. Em especial aos Diretores e Coordenadores Pablo Andrey, Fábio Albuquerque e Ivan Medeiros e aos demais amigos que compõem um agradável e competente ambiente de trabalho.

À Escola Superior de Redes e ao Instituto Nacional de Engenharia de Software, pelo apoio.

A todos os demais que de alguma forma contribuíram para a realização desse trabalho.

RESUMO

Atualmente, mais da metade da população mundial vive em cidades. A expectativa é que este quantitativo alcance cinco bilhões de pessoas em 2030. Essa rápida expansão populacional dos grandes centros apresentará os maiores desafios, como por exemplo: a poluição do ar, a mobilidade urbana, problemas de saúde, energia e gestão de lixo. Soluções para estes desafios exigem a integração de várias Tecnologias de Informação e Comunicação (TIC´s) em diferentes domínios. As cidades que exploram estas soluções, utilizando as TIC´s para auxiliar no enfrentamento de seus problemas, têm sido chamadas de Cidades Inteligentes. Todavia, as primeiras soluções criadas apresentaram limitações, pois em geral eram baseadas em sistemas independentes, com tecnologias diferentes e sem a preocupação com interoperabilidade e escalabilidade. Neste contexto, plataformas de *middleware* têm sido utilizadas como uma infraestrutura de sistemas para auxiliar o desenvolvimento de aplicações, sistemas e serviços de cidades inteligentes. Dentre os principais desafios destas plataformas, a integração de dados é bastante representativa, pois permite fornecer aos usuários informações consolidadas de diferentes provedores com transparência de localização e de formato de representação. Nesta direção, este trabalho propõe uma abordagem de processamento de consultas para plataformas de *middleware* distribuído e particionado, cujas informações estão armazenadas em diferentes provedores de dados e geograficamente distribuídas. Um estudo de caso com significativo volume de dados semirreais de transporte público foi desenvolvido para validar e avaliar a abordagem proposta. No estudo de caso realizado, cinco cenários distribuídos, envolvendo até 10 servidores de plataforma com 10.000 instâncias de entidades de contexto foram utilizados para avaliar essa proposta. Em cada cenário foi avaliada a taxa de transferência no processamento de consultas distribuídas. Dessa forma, comparando com o Orion Context Broker, um provedor de dados largamente utilizado em projetos europeus para Cidades Inteligentes, a abordagem proposta foi capaz de alcançar uma vazão com desempenho superior a 1.600%.

Palavras-chaves: Cidade Inteligente; Middleware; Sistema Distribuído; Processamento de Consulta; Linguagem de Consulta de Contexto.

ABSTRACT

Nowadays, more than half of the world's population lives in cities. It is expected to reach five billion people by 2030. This fast population growth in major centers will present biggest challenges, such as air pollution, urban mobility, health problems, energy and waste management. Solutions to these challenges require the integration of various Information and Communication Technologies (ICTs) in different domains. Cities that exploit these solutions, using ICTs to help solve their problems, have been called Smart Cities. However, the first solutions created had some limitations, as they were generally based on independent systems, with different technologies and without the concern with interoperability and scalability. In this context, middleware platforms have been used as infrastructure to assist the development of Smart City applications, systems and services. Among the main challenges of these platforms, data integration is very representative, as it allows users to provide consolidated information from different providers with transparency of location and representation format. In this sense, this paper proposes a query processing approach for distributed and partitioned middleware platforms, whose information is stored in different data providers and geographically distributed. A case study with significant volume of semi-public transit data was developed to validate and evaluate the proposed approach. In the performed case study, five distributed scenarios involving up to 10 platform servers, with 10,000 instances of context entities were used to evalute this proposal. In each scenario, the throughput in query processing was evaluated. Thus, compared to Orion Context Broker, a data provider widely used in European Smart Cities projects, the proposed approach was able to achieve a throughput exceeding 1.600%.

Keywords: Smart City; Middleware; Distributed Systems; Query Processing; Context Query Language.

Lista de Figuras

Lista de tabelas

Lista de siglas

API	*Application Programming Interface*
BDD	Banco de Dados Distribuído
CQL	*Context Query Language*
CQE	*Context Query Engine*
CIM	*Context Instance Manager*
CSM	*Context Schema Manager*
DCDS	*Distributed Context Data Schema*
DPA	*Data Provider Adapter*
EBNF	*Extended Backus-Naur Form*
HTTP	*Hypertext Transfer Protocol*
IOT	*Internet of Things*
LDP	*Local Data Provider*
JSON	*JavaScript Object Notation*
MPSG	*Mobile Physical Space Gateway*
OASC	*Open and Agile Smart Cities*
OCB	Orion *Context Broker*
OMA	*Open Mobile Alliance*
P2P	*Peer-to-Peer*
RDF	*Resource Description Framework*
REST	*Representational State Transfer*
SGBD	Sistema Gerenciador de Banco de Dados
SGBDD	Sistema Gerenciador de Banco de Dados Distribuído
TIC	Tecnologia da Informação e Comunicação
UFPB	Universidade Federal da Paraíba
XML	*Extensible Markup Language*

SUMÁRIO

1. Introdução

O processo acelerado de urbanização, em nível mundial, resultou na superação do quantitativo populacional que vive nas cidades em relação ao de áreas rurais no ano de 2008, de acordo com as Nações Unidas (UNITED NATIONS, 2012). No período entre 1950 a 2011, a proporção da população urbana passou de 26% para 51% do total. As perspectivas indicam que, em 2030, a população urbana alcançará a quantidade de cinco bilhões de indivíduos (MARTINE; MARSHALL, 2007). Todavia, apenas 2% do território do planeta são ocupados por cidades, fato que acarreta no adensamento cada vez maior, o que aumenta os desafios na administração dessas cidades. O aumento da densidade populacional, que depende dos recursos dos centros urbanos, afeta diretamente serviços e infraestrutura, como por exemplo: saúde, meio ambiente, transporte público, segurança, energia elétrica e água (KHAN, 2012).

Conviver neste cenário desafiador e ainda oportunizar melhoria na qualidade de vida dos cidadãos tem sido um dos obstáculos das grandes cidades. Novas formas de aprimorar o uso dos recursos e da infraestrutura de uma cidade, de forma sustentável, estão sendo pesquisadas, principalmente através da utilização das Tecnologias da Informação e Comunicação (TIC´s). O uso de TIC´s para auxiliar na resolução dos problemas urbanos tem sido chamado de Cidades Inteligentes, do inglês *Smart Cities*, e tem ganhado cada vez mais a atenção de pesquisadores em todo o mundo. Silva (2013), apresenta o conceito de Cidade Inteligente como a necessidade de gerir, automatizar, otimizar e explorar todos os aspectos que poderiam ser aperfeiçoados em uma metrópole. Entre as tecnologias utilizadas, IoT (do inglês, *Internet of Things*) tem sido decisivo na resolução desses problemas, pois a sua adoção desencadeou o surgimento de novos serviços, sistemas e aplicações, principalmente a partir da implantação de tecnologias digitais em objetos físicos (como celulares, veículos e prédios) para compartilhamento de dados e recursos.

Nos últimos anos, inúmeras aplicações surgiram com foco em problemas de domínios específicos, por exemplo: mobilidade urbana, gestão dos resíduos sólidos, saúde e educação (NAPHADE, 2011). Em sua maioria, estas soluções são desenvolvidas e administradas por diferentes gestores públicos ou privados que adotam infraestruturas e plataformas de TIC´s incompatíveis, favorecendo a formação de sistemas independentes com repositórios distintos, dados segmentados e a ausência de padrões, resultando no que foi batizado de ilhas de informação (VILLANUEVA, 2013). Consequentemente, as soluções para cidades inteligentes possuem limitada interoperabilidade, complexa ou ausência de integração de dados e não

exploram as vantagens advindas de estruturas de particionamento de dados, como por exemplo, melhora do desempenho e aumento de escalabilidade.

> *Um problema na maioria das aplicações de cidades inteligentes, é que normalmente os sistemas são direcionados a um problema específico e são desenvolvidos sempre desde o início com pouco reuso de software e sem a comunicação entre esses sistemas. Essa abordagem leva a um maior trabalho, ao uso não otimizado dos recursos e impede a criação de aplicações que necessitem de dados e serviços de diversos domínios, o que é uma das principais características de cidades inteligentes. (KON; SANTANA, 2016, p. 14).*

Como forma de resolver os problemas de integração entras as aplicações, várias pesquisas recentes e experimentos já realizados em algumas cidades apontam como solução a adoção de plataformas de *middleware* (VILLANUEVA, 2013); (HERNÁNDEZ-MUÑOZ, 2011); (FAZIO, 2012). Essas plataformas oferecem mecanismos para explorar plenamente as potencialidades das novas tecnologias, acessando dados e serviços da cidade de uma forma integrada e com o objetivo de potencializar o surgimento de aplicações, serviços e sistemas de Cidades Inteligentes (KON; SANTANA, 2016).

1.1. Motivação

Uma verdadeira cidade inteligente deve prover interoperabilidade entre as diferentes partes dos sistemas, serviços e aplicações que compõem sua infraestrutura. Neste sentido, o conjunto de plataformas de diferentes domínios deveriam seguir os mesmos padrões, de comunicação, modelo de dados, formato de dados e utilizar exclusivamente estes padrões (SILVA, 2016). No entanto, isto não ocorre porque diferentes empresas discordam em relação ao melhor padrão, entre outros interesses comerciais. Esta queda de braço entre empresas mundiais persiste, e todos buscam impor suas especificações proprietárias (SILVA, 2016).

Diversos autores apontam que, entre os desafios ainda não resolvidos pelas plataformas de *middleware*, está o suporte limitado ao problema de integração entre os dados de diferentes plataformas (MINERAUD, 2015; SILVA, 2016; DEL ESPOSTE, 2017). Este problema visa permitir que diversos domínios específicos e suas plataformas sejam tratados como um único sistema. Mineraud (2015) afirma que soluções neste caminho ainda não foram exaustivamente estudadas e o resultado observado tem sido a formação de ecossistemas múltiplos e fragmentados, formando silos verticais.

Entre as lacunas encontradas em plataformas para cidades inteligentes estão: a) precários mecanismos de compartilhamento de dados (DEL ESPOSTE, 2017), inclusive compartilhar informações entre diferentes domínios (MARAN, 2014); b) mecanismos

complexos para acessar dados de múltiplas plataformas (MINERAUD, 2015); c) mecanismos ineficientes para processar dados de formatos e modelos de dados diferentes (MINERAUD, 2015); d) mecanismos de processamento centralizados (CHEN, 2015); e) limitada escalabilidade resultante da adoção de processamento centralizado (RAZZAQUE, 2016).

Para auxiliar no desenvolvimento de serviços, sistemas e aplicações para cidades inteligentes, tendo como horizonte alcançar escalabilidade e interoperabilidade, uma plataforma de *middleware* para cidades inteligentes com arquitetura distribuída pode ser um caminho capaz de ultrapassar as lacunas identificadas. Uma plataforma com este intuito deve possuir alguns requisitos, como:

a) Interagir com diversos servidores geograficamente distribuídos de forma transparente para os usuários, uma vez que podem existir diversos sistemas localizados em diferentes servidores de órgãos ou empresas em regiões distintas. Para conseguir prover este tipo de acesso é preciso obter um mecanismo que resolva o problema de localizar de forma transparente tais servidores, que é uma característica conhecida por transparência de localização.
b) Outro requisito desejável é que a plataforma possa prover interoperabilidade entre diferentes modelos de dados, uma vez que, em sua maioria, os sistemas das cidades inteligentes foram desenvolvidos de forma independente e sem padronização. Neste sentido, é preciso obter um mecanismo para prover transparência de representação de dados.
c) Também é esperado que este tipo de plataforma de *middleware,* no tocante à escalabilidade, proporcione a distribuição do processamento entre diversos servidores, com o intuito de diminuir o tempo de resposta no processamento de consultas.
d) Além da distribuição do processamento entre diversos servidores, outro requisito importante diz respeito ao armazenamento dos dados em mais de um servidor, ou seja, qual o tipo de particionamento de dados que é utilizado pela plataforma de *middleware*. Dentre os benefícios que o particionamento de dados pode oferecer está o acesso concorrente às bases de dados e um tempo menor de resposta das operações, como por exemplo, uma consulta. Desta forma, é desejável que uma plataforma se beneficie das vantagens oriundas do particionamento de dados.

Com a finalidade de contribuir para a superação das lacunas identificadas, além de auxiliar no desenvolvimento de serviços, sistemas e aplicações para cidades inteligentes, a partir da adoção dos requisitos identificados, uma plataforma de *middleware* distribuída e

particionada para cidades inteligentes, denominada *Sirius*, foi desenvolvida pelos pesquisadores do laboratório COMPOSE da UFPB, como parte do projeto INES (Instituto Nacional de Ciência e Tecnologia para Engenharia de Software). Vale ressaltar que a abordagem de processamento de consultas, proposta nesta dissertação, é parte integrante da plataforma *Sirius*, embora possa ser adotada com possíveis adaptações em outras plataformas de *middleware* para cidades inteligentes. No decorrer do trabalho cada servidor da plataforma *Sirius*, em conjunto com o provedor de dados associado, é chamado de servidor *Sirius*, para melhor compreensão

Para viabilizar uma proposta de plataforma de *middleware* capaz de transpor as lacunas identificadas, de forma a atender os requisitos apresentados, foi desenvolvido um metamodelo de dados capaz de representar, tanto a especificação quanto o modelo de dados de uma entidade de contexto - chamado de *Distributed Context Data Schema* (DCDS) (LOPES; SILVA; ELIAS, 2019). Aqui, uma entidade de contexto pode ser entendida como qualquer objeto de uma cidade, como uma pessoa, um ônibus, um celular ou um sensor (ABOWD et al., 1999). Este metamodelo também é decorrente da parceria com os pesquisadores do laboratório COMPOSE da UFPB.

1.2. Objetivos

O objetivo é desenvolver uma abordagem de processamento de consultas para plataformas de *middleware* distribuído e particionado no contexto de Cidades Inteligentes. Esta abordagem compreende mecanismos de descoberta de forma automática e transparente da localização dos provedores e da estratégia de particionamento dos dados. Assim, poderá aumentar o desempenho de tais plataformas, através de acesso concorrente, ao mesmo tempo em que ocultará detalhes da infraestrutura das aplicações externas.

Para alcançar o objetivo geral os seguintes objetivos específicos fazem parte do escopo deste trabalho:

a) definir um mecanismo de localização de recursos distribuídos em estruturas particionadas e que seja transparente para a aplicação consumidora;

b) conceber um mecanismo de consulta que utilize um modelo de dados unificado para integrar resultados de consultas realizadas a múltiplas plataformas;

c) desenvolver um mecanismo que permita a uma plataforma de *middleware* acessar provedores de dados com modelos de dados diferentes;

d) realizar uma análise comparativa de processamento de consultas em diferentes cenários alterando variáveis como quantidade de provedores, tipo de particionamento, e quantidade de instâncias de entidade retornadas.

1.3. Organização do Trabalho

O trabalho encontra-se organizado da seguinte forma: O Capítulo 1 apresentou o contexto, motivação e objetivos da dissertação. O Capítulo 2 apresenta a fundamentação teórica, onde são introduzidos alguns conceitos relacionados à abordagem desenvolvida. A saber: plataformas de *middlewares*, mecanismos de transparência, técnicas de interoperabilidade entre provedores de dados, linguagens de consulta de contexto e estratégias de particionamento de dados. No Capítulo 3, a plataforma de *middleware Sirius* e o metamodelo de dados DCDS são apresentados, o que permite entender os recursos utilizados pela abordagem de processamento de consulta. O Capítulo 4 apresenta a proposta principal deste trabalho, a abordagem de processamento de consulta distribuída e particionada, os componentes relacionados, assim como o detalhamento de suas funcionalidades. O Capitulo 5 apresenta um estudo de caso, no qual um protótipo foi implementado para validar e avaliar a abordagem proposta em diferentes cenários. Experimentos são realizados para analisar o comportamento da abordagem de consulta em cenários com diferentes estruturas de particionamento e diferentes quantidades de servidores. No Capítulo 6 são apresentados os trabalhos relacionados, enfatizando os mecanismos de consultas, destacando as semelhanças e diferenças em relação ao presente trabalho, e, assim, evidenciando suas principais contribuições. Finalmente, no Capitulo 7, são apresentadas as considerações finais, ressaltando as contribuições, limitações e trabalhos futuros.

2. Fundamentação Teórica

Este capítulo se destina a apresentar a fundamentação teórica do presente trabalho, considerando assuntos relacionados às plataformas de *middleware* para cidades inteligentes, problemas de sistemas distribuídos e processamento de consultas. Inicialmente é apresentado o conceito de plataforma de *middleware* e a sua utilização no contexto das cidades inteligentes. Em seguida são apresentados mecanismos de transparência utilizados em sistemas distribuídos, como transparência de localização, de acesso a dados, entre outros. Na sequência, os conceitos abordados são processamento distribuído de consultas, linguagens de consulta e particionamento de dados.

2.1. Plataformas de *Middleware* para Cidades Inteligentes

No contexto de cidades inteligentes, são encontrados sistemas independentes, geograficamente distribuídos, com diferentes infraestruturas e plataformas tecnológicas. Este cenário complexo dificulta o trabalho de desenvolvedores de sistemas, serviços e aplicativos, uma vez que não existe interoperabilidade ou integração dos dados dos sistemas (JALALI; EL-KHATIB; MCGREGOR, 2015). Como forma de minimizar a complexidade do gerenciamento das informações das entidades que compõem as cidades e disponibilizar ferramentas que facilitem o desenvolvimento de novos sistemas enriquecidos com dados de diversos domínios (MARAN, 2014), pesquisadores têm apresentado soluções de plataforma de *middleware* para atuar como um sistema intermediário e auxiliar no acesso aos diversos sistemas independentes de uma cidade (SILVA, 2016). Neste sentido, uma plataforma de *middleware* para cidades inteligentes pode ser conceituada como um *software* que permite a integração de diferentes subsistemas, com informações provenientes de diversas áreas, buscando alcançar uma visão holística da cidade (BACIKOVÁ; PORUBÄN, 2013).

Os autores Jalali, El-Khatib e Mcgregor (2015) discorrem sobre a dificuldade em se conceber uma arquitetura geral que possa ser amplamente difundida. Parte desta dificuldade é decorrente da heterogeneidade de tecnologias, sistemas e serviços existentes. Os autores apresentam uma arquitetura dividida em camadas, entre as quais a camada de gerenciamento e acesso a dados encontra-se entre a camada de aplicação e as demais camadas de infraestrutura, tais como sensoriamento e rede e controle. Desta forma é proporcionado acesso padronizado aos serviços do sistema (JALALI; EL-KHATIB; MCGREGOR, 2015).

Na literatura de cidades inteligentes é comum a utilização do conceito de entidade de contexto para designar uma pessoa, lugar ou algum dispositivo que é relevante para a

interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação (ABOWD et al., 1999). Qualquer informação que auxilie na identificação ou caracterização de uma entidade é denominada de informação de contexto. Plataformas para Cidades Inteligentes utilizam informações de contexto para diversos fins, incluindo os processos de análise, tomada de decisão e planejamento. Dentre os objetivos destas plataformas destaca-se o objetivo de fornecer um conjunto de funcionalidades que auxiliem desenvolvedores na produção de aplicações em diversas áreas relacionadas a centros urbanos.

2.1.1. Métodos de Acesso aos Dados

Um dos requisitos de uma plataforma de *middleware* é o gerenciamento de dados, que compreende os processos de aquisição, remoção, acesso e armazenamento de dados. Para permitir o desenvolvimento de aplicações, é importante que os dados coletados e armazenados possam ser acessados por aplicações e serviços externos à infraestrutura da cidade. Os métodos de acesso aos dados da plataforma podem ser:

- **Consulta direta (*Pull* ou *Request-Response*):** neste método, a aplicação acessa os dados definindo parâmetros da consulta, e, em seguida, o *middleware* retorna o resultado encontrado. Este processamento é realizado pelo *middleware* no momento seguinte ao recebimento da requisição, podendo ser classificada como uma resposta instantânea (PERERA et al., 2014). Um exemplo é o conjunto de dados referentes às posições dos ônibus de uma linha, representando a posição atual de cada ônibus.
- **Subscrição (*Publish/Subscribe*):** através do método por subscrição (também chamado de publicação/subscrição ou, em inglês, *publish/subscribe*), uma aplicação sinaliza o interesse em certas informações de contexto, e, sempre que tais informações forem modificadas, os respectivos dados são enviados a todos os clientes que realizaram a assinatura (KON; SANTANA, 2016). O *middleware* então retornará o resultado de acordo com o que foi especificado apenas quando a condição sinalizada ocorrer. Este método também é chamado de método baseado em evento.

A atual proposta tem foco em explorar os mecanismos necessários para preencher a lacuna existente em relação ao limitado suporte para integração entre dados de plataformas diferentes. Como primeiro método de acesso foi escolhido, o mecanismo de consulta direta, que é muito utilizado pelas demais plataformas (RAZZAQUE, 2016), assim como também fora encontrado na literatura da área como método escolhido para estudos de casos.

2.2. Mecanismos de Transparência

Uma das funções de uma plataforma de *middleware* distribuído é permitir que os conjuntos de dados correlacionados que estão distribuídos geograficamente sejam vistos de forma integrada, como se todos os dados estivessem localizados localmente na plataforma de *middleware* consultada. Assim, com o intuito de minimizar a complexidade, do ponto de vista dos desenvolvedores, em localizar as fontes de armazenamento dos dados, ou ter o domínio de todos os modelos de dados utilizados pelos provedores independentes alguns tipos de transparência precisam ser providos pela plataforma. Uma vez que estes problemas já foram exaustivamente estudados nas áreas de Bancos de Dados Distribuídos e Sistemas Distribuídos os conceitos amplamente disponíveis na literatura dessas áreas são utilizados para a fundamentação deste trabalho.

Coulouris (2007, p. 37) apresenta uma definição para transparência em sistemas distribuídos como sendo “a ocultação, para um usuário final ou para o desenvolvedor do sistema, da separação dos componentes em um sistema distribuído de modo que o sistema seja percebido como um todo, em vez de uma coleção de componentes independentes”. Entre os tipos de transparência pode-se citar:

a) transparência de acesso: permite que recursos locais e remotos sejam acessados com o uso de operações idênticas (COULOURIS, 2007).

b) transparência de localização: garante que a localização das várias partes do sistema fique oculta dos usuários (CASANOVA, 1999).

c) transparência de representação: permite que diferentes formatos de dados ou nomes diferentes com o mesmo significado possam ser representados de forma transparente (HURSON; JIAO, 2005).

d) transparência de escalabilidade: permite que o sistema se expanda em escala, sem alterar a estrutura do sistema ou os algoritmos (COULOURIS, 2007).

e) transparência de particionamento: características de particionamento dos dados deve ficar oculta, tornando transparente, por exemplo, em quantos provedores os dados estão particionados e a forma de particionamento (ÖZSU; VALDURIEZ, 2011).

As características provenientes de mecanismos de transparência são desejáveis para os recursos de processamento de consultas distribuídas e particionadas. A próxima seção aborda um dos mecanismos utilizados para alcançar a transparência de representação, chamado de adaptador**.**

2.2.1. Adaptadores

Um dos padrões de projetos estruturais descritos em Gamma (1995) é o padrão *Adapter* (também chamado de *Wrapper*). Este padrão permite a comunicação entre classes que não podem trabalhar juntas devido à incompatibilidade de suas interfaces. Como solução, uma classe intermediária realiza os ajustes necessários para a comunicação.

Adaptadores são utilizados em diversas áreas como Banco de Dados Distribuídos (OLIVEIRA, 2007) e Sistemas de Integração de Dados na Web (AYRES, 2003). Para cada fonte de dados, ou provedor de dados, existe um adaptador com capacidade de acessar e retornar a resposta correspondente a uma consulta. Estes adaptadores são responsáveis pela tradução de modelos de dados específicos de cada provedor em um modelo de dados unificado (NADAI, 2016), que é utilizado pela plataforma de *middleware*. Este processo de tradução resulta na homogeneização da representação das informações, o que permite a interoperabilidade com dados de diferentes representações.

No processamento de consultas em sistemas de integração de dados na web, adaptadores são utilizados por sistemas baseados na arquitetura mediador-adaptador (*mediator—wrapper*), proposta por Wiederhold (1992), onde um mediador (*mediator*) centraliza as informações de fontes de dados distribuídas e heterogêneas, e, através dos adaptadores, acessa dados em diferentes formatos de dados (AYRES, 2003). Neste tipo de abordagem, para cada fonte de dados existe um adaptador específico, responsável por converter o modelo dos dados específico da fonte de dados no formato convencionado pelo sistema e compreendido pelo mediador centralizado. Desta forma, é possível ter fontes de dados web com modelos de dados heterogêneos sendo consultados durante o processamento de consultas com integração de dados na web.

2.3. Processamento Distribuído de Consultas

Em Banco de Dados Distribuídos (BDDs), o processamento de consultas distribuídas pode ser dividido em camadas, que tratam problemas bem definidos, como: decomposição de consultas, localização dos dados, otimização global e otimização local. Estas camadas são chamadas em inglês de *Generic Layering Scheme for Distributed Query Processing* (ÖZSU; VALDURIEZ, 2011, p. 226).

Dentre as vantagens de se utilizar BDDs está a possibilidade de explorar a organização geograficamente dispersa (CODATO, 2001). Uma das possibilidades advindas é a capacidade de administrar diversos microcomputadores interligados e a facilidade em adicionar ou excluir

microcomputadores em diferentes localidades, quando o volume de dados apresenta tendência de crescimento, provendo mecanismos de escalabilidade (CODATO, 2001).

Um BDD também possui suas limitações e desvantagens, como por exemplo: a) a taxa de acesso na obtenção de informações em ambientes geograficamente distintos não pode ser comparada com a taxa de acesso a discos locais, pois os fatores pertinentes à comunicação exercem influência (tempo de transmissão, congestionamento, etc.); b) as etapas envolvidas resultam em *overhead* de processamento, que pode ser significativo; c) exige maior complexidade para permitir a coordenação de comunicação entre os microcomputadores interligados (CODATO, 2001).

Por estes motivos, o processamento de consulta em BDDs possui complexidade maior que em banco de dados centralizados. De forma análoga, o processamento de uma consulta em uma plataforma com estrutura distribuída possui uma complexidade maior que em estruturas centralizadas (CODATO, 2001).

A atual proposta utiliza o conceito de processamento distribuído em sua abordagem distribuída, composta por vários servidores *Sirius*, nos quais o processamento de consultas pode ser distribuído em cada servidor, e, assim, diminuir o tempo de processamento, facilitar o gerenciamento dos dados de cada servidor *Sirius* e aumentar a escalabilidade. Pode-se dizer que a razão principal por trás de um processamento distribuído é prover facilidade para a necessidade de gerenciamento de grandes volumes de dados que aflige a realidade atual, através da conhecida filosofia do “dividir para conquistar” (ÖZSU; VALDURIEZ, 2011).

2.3.1. Linguagens de Consulta de Contexto

Uma linguagem de consulta de contexto (CQL, do inglês *Context Query Language*) é utilizada para acessar (REICHLE, 2008) e compartilhar dados de contexto (HASSANI, 2016). Este tipo de linguagem deve expressar consultas, através de restrições e filtros, para a requisição de informações contextuais, sem revelar detalhes da infraestrutura que provê suporte ao sistema (HASSANI, 2016; PERTTUNEN; RIEKKI; LASSILA, 2009). Como toda linguagem de alto nível, deve ser não-procedimental no sentido do usuário especificar características dos dados que deseja acessar e não como eles devem ser acessados (CASANOVA, 1999).

Uma CQL também pode ser considerada uma espécie de um mecanismo que provê transparência (HASSANI, 2016), uma vez que oculta os detalhes de baixo nível dos provedores de dados e das técnicas de armazenamento. Através de uma única CQL, com

abstrações de alto nível, é possível acessar informações representadas em diferentes modelos de dados.

Entre as características e requisitos de uma CQL (CHEN, 2014), destacam-se:

- A linguagem deve expressar consultas em um nível abstrato, de forma a desacoplar detalhes dos provedores de dados, tais como, sua localização e mecanismos de armazenamento.
- Deve prover suporte não apenas ao acesso de um único elemento, ou dos elementos de certo tipo, mas também de retornar um conjunto de elementos que atendam a uma determinada restrição expressa em uma única consulta (REICHLE, 2008).
- A linguagem deve expressar consultas geográficas possibilitando expressar uma localização, proximidade espacial e área geográfica;
- A linguagem deve possuir múltiplos operadores, por exemplo, utilizando um conjunto de operadores lógicos que combinam condições simples em outras mais complexas.

Atender aos requisitos listados acima exige um alto nível de complexidade, principalmente ao se tratar de uma abordagem distribuída. Com o propósito de atender aos requisitos listados acima, este trabalho utilizou como base a sintaxe da CQL do sistema gerenciador de contexto do projeto FIWARE (FIWARE, 2019d), chamado de *Orion Context Broker* (OCB) (FIWARE, 2019b). O OCB utiliza a interface NGSI-10 (OMA, 2012), definida por uma associação que desenvolve padrões abertos de comunicação, a *Open Mobile Alliance*. A implementação do NGSI pelo OCB sofreu modificações e foi chamada de FIWARE NGSI-v2 (ou apenas de NGSI-v2) (FIWARE, 2019a), incorporando além da interface, a definição de uma CQL (ausente na definição original) e um modelo de dados aperfeiçoado. Atualmente, esta interface tem ampliado sua aceitação por projetos e órgãos de padronização como *Open and Agile Smart Cities*[1] (OASC) (SILVA, 2016).

A interface NGSI-v2 permite o gerenciamento das informações de contexto do OCB com o suporte de uma API RESTful, através de requisições HTTP. A interface permite dois métodos de acesso (FIWARE, 2019c): consultas diretas ou assinatura de interesse por modificações (subscrição). Conforme apresentado na seção 2.1.1, a abordagem aqui proposta adota o primeiro método.

A linguagem de consulta definida na NGSI-v2, ao mesmo tempo em que é simples, possui expressividade o suficiente para permitir que representações de diferentes domínios de cidades inteligentes possam ser consultados. Este trabalho introduziu pequenas mudanças à

[1] https://oascities.org

CQL da NGSI-v2, como uma forma de contribuição para a sua evolução, em relação à adição do operador lógico disjuntivo, que será detalhado no Capítulo 4.

2.4. Particionamento de Dados

Inicialmente são abordados alguns conceitos de particionamento de dados utilizados em SGBDs relacionais e SGBDs Distribuídos. Nestes sistemas a utilização de estratégias de particionamento de dados possui impacto positivo no desempenho e no gerenciamento.

Entre as estratégias de particionamento existentes, duas são utilizadas por banco de dados relacionais (ÖZSU; VALDURIEZ, 2011): particionamento vertical e particionamento horizontal. Por um lado, no particionamento vertical, uma tabela T é dividida em duas ou mais tabelas (chamadas aqui de sub-tabelas), cada uma contendo um subconjunto das colunas da tabela T. Com a divisão, os vários acessos ao conjunto de sub-tabelas exigirá menor quantidade de dados a serem lidos, proporcionando benefícios como menor processamento e menor uso de memória (AGRAWAL; NARASAYYA; YANG, 2004).

Por outro lado, o particionamento horizontal adota outra estratégia, na qual as linhas de uma tabela T são fisicamente divididas em duas ou mais sub-tabelas, possibilitando que sejam acessadas separadamente. O particionamento horizontal foi largamente empregado para facilitar o gerenciamento de SGBDs por parte dos administradores (AGRAWAL; NARASAYYA; YANG, 2004).

Mesmo com a utilização de particionamento, a visão que um usuário ou aplicação possui de uma base particionada deve ser a mesma que ele possui de uma relação centralizada (ELLER, 1997).

De volta ao contexto de plataformas para cidades inteligentes, requisitos importantes podem ser alcançados com o emprego de estratégias de particionamento, como por exemplo:

a) Desempenho: os sistemas para cidades inteligentes devem lidar cada vez mais com novas aplicações, que, apesar do aumento do volume de dados, exigem cada vez mais velocidade em oferecer serviços e respostas (da SILVA, 2013).
b) Escalabilidade: devido ao crescimento do volume de dados, as plataformas de *middleware* de cidades inteligentes necessitam de recursos que possibilitem expandir seus sistemas (hardware e software) para acompanhar este crescimento (da SILVA, 2013).

Por estes motivos, o atual trabalho pretende explorar a utilização de estratégias de particionamento como forma de proporcionar escalabilidade e desempenho para uma abordagem de consulta distribuída em plataformas de *middleware* de cidades inteligentes.

Além de explicitar as estratégias de particionamento que podem ser adotadas, o presente trabalho também realiza uma avaliação de desempenho, evidenciando as diferenças entre as diferentes estratégias de particionamento adotadas.

2.5. Considerações Finais

Este capítulo apresentou os principais fundamentos e conceitos utilizados na abordagem de processamento de consulta proposta. Inicialmente foi apresentado o conceito de plataforma de middleware e como as pesquisas têm enfrentado os problemas no contexto das cidades inteligentes. De forma complementar, também foi apresentado os principais métodos de acesso a dados. Em seguida foi discorrido sobre os diferentes mecanismos de transparência utilizados por sistemas distribuídos, dando ênfase aos mais importantes para a abordagem proposta. Por fim, foram abordados dois tópicos pouco encontrados em outros trabalhos da área, que são o processamento distribuído de consultas e o particionamento de dados.

A abordagem de processamento distribuído de consulta proposta neste trabalho apresenta um método de distribuição de dados classificado de *request-response* (ou consulta direta), utiliza uma linguagem de consulta que fora adaptada de iniciativas de padronização de protocolos e dados, tal linguagem apresenta-se simples, porém expressiva. Através dela é possível consultar entidades de um esquema específico, com restrições de atributos que se deseja consultar. Após uma etapa de otimização das consultas, em que provedores de dados são comparados com os atributos consultados, as consultas são distribuídas apenas para aqueles provedores que contêm dados associados às respectivas consultas, o que permite dividir a carga de processamento entre a rede de provedores da plataforma. Tanto a representação de uma entidade, como os mecanismos de localização e integração são baseados em um metamodelo desenvolvido durante o trabalho, denominado DCDS (LOPES; SILVA; ELIAS, 2019).

Para prover a possibilidade de trabalhar com provedores de dados que utilizam modelos de dados heterogêneos, é utilizado o conceito de adaptador para traduzir estes diferentes tipos de modelos para um modelo de dados único (derivado do metamodelo DCDS).

Em busca de permitir que grandes volumes de dados sejam consultados, na direção de alcançar a escalabilidade que é cada vez mais exigida às aplicações de cidades inteligentes, a proposta deste trabalho possibilita que diferentes estratégias de particionamento sejam adotadas.

Na medida em que a literatura é ampla acerca de tais áreas, o capítulo focou nas temáticas relevantes para o trabalho, mas leituras adicionais são recomendadas para um estudo aprofundado sobre cada um dos conceitos.

3. Sirius: Uma Plataforma de *Middleware* Distribuído e Particionado

O presente trabalho compreende uma parte de uma proposta de plataforma de *middleware* para cidades inteligentes chamada *Sirius*. A plataforma Sirius tem como objetivo auxiliar desenvolvedores que necessitam produzir serviços, sistemas e aplicações que utilizam uma infraestrutura distribuída e particionada. Este capítulo apresenta uma visão geral desta plataforma, conceituando os principais componentes e suas funcionalidades. Também é apresentado o metamodelo de dados que fornece os alicerces necessários para integração e interoperabilidade entre provedores heterogêneos distribuídos.

3.1. Introdução

O desenvolvimento de serviços, sistemas e aplicações para cidades inteligentes é uma tarefa complexa, a ponto de ainda não ter uma plataforma universalmente padronizada (PCAST, 2016; HALEVY, 2003). Entre as dificuldades encontradas para o desenvolvimento destes serviços está o uso de diferentes tecnologias e repositórios incompatíveis com diferentes interfaces e modelos de dados heterogêneos.

A arquitetura apresentada neste capítulo consiste em uma abordagem para criação, gerenciamento e consulta de dados em plataformas de *middleware* distribuído para este cenário, visando minimizar a dificuldade em se trabalhar com provedores de dados heterogêneos, independentes e distribuídos geograficamente, através do desacoplamento, tanto da localização, como de especificidades destes provedores (como por exemplo linguagem de consulta e modelo de dados).

Para enfrentar estes problemas, durante o percurso desta pesquisa foi desenvolvido um metamodelo de dados chamado *Distributed Context Data Schema* (DCDS) (LOPES; SILVA; ELIAS, 2019), que auxilia e provê recursos no sentido de plataformas compatíveis conseguirem realizar operações complexas como integração, interoperabilidade, particionamento, entre outros. Este metamodelo será apresentado com mais detalhes, onde seus principais elementos são descritos.

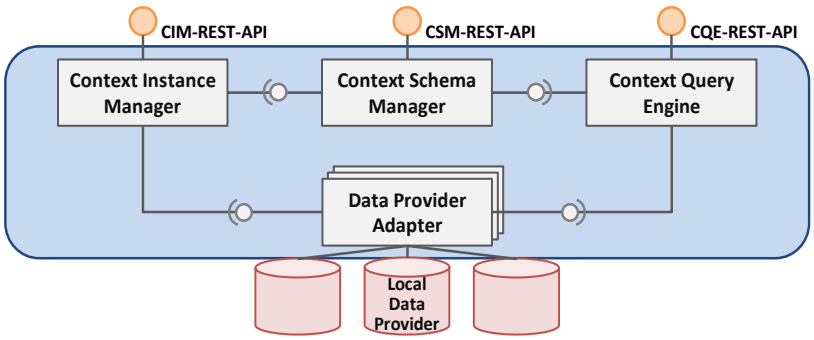
3.2. Sirius

A Plataforma *Sirius* foi desenvolvida visando auxiliar o desenvolvedor de sistemas, serviços e aplicações no contexto de cidades inteligentes. Sua arquitetura consiste em 4 componentes, conforme ilustrado pela Figura 1. Em conjunto, estes componentes realizam quatro grandes grupos de funcionalidades:

a) Gerenciamento de especificações de entidades de contexto;
b) Gerenciamento de instâncias de entidades de contexto;
c) Processamento de consultas às informações contextuais;
d) Adaptação de modelos de dados e linguagens de consulta para manipulação e acesso de forma transparente à diferentes tipos de provedores.

As próximas seções apresentam o detalhamento de cada um dos componentes.

Figura 1 : Arquitetura Baseada em Componentes da Plataforma Sirius



Fonte: adaptado de (LOPES; SILVA; ELIAS, 2019).

3.2.1. Context Schema Manager - CSM

O componente *Context Schema Manager* (CSM) recebe como entrada requisições de criação, atualização, leitura e exclusão de esquemas de entidades de contexto e de seus elementos. Além de realizar as operações sobre um esquema propriamente, é possível atuar sobre seus atributos e metadados. Um esquema de entidade especifica os atributos de uma entidade, assim como também possui os nomes dos domínios em que se localizam os servidores *Sirius* (em conjunto com o provedor de dados local) que armazenam as instâncias da entidade de contexto. No contexto deste trabalho uma entidade pode ser qualquer objeto existente nas cidades, como uma pessoa, um carro, uma lixeira, ou outro sistema.

O propósito deste componente é gerenciar o ciclo evolutivo de um esquema de entidade. Através deste gerenciamento, um esquema pode evoluir e refletir a dinamicidade dos objetos e entidades urbanas, que estão em constantes mudanças, refletindo a adoção e captação de novos tipos de informações a partir de mais sensores ou outros acessórios tecnológicos.

Algumas soluções de plataformas desenvolvidas para o contexto de Cidades Inteligentes utilizam estratégias centralizadas para armazenar dados como o repositório de referências para outros serviços (um esquema DCDS desempenha esta função de referenciar serviços e servidores), como é o caso do NEXUS (DÜRR et al., 2004). No sistema NEXUS,

existe um componente chamado *Area Service Register*, que possui estrutura centralizada, sendo acessado por todos os demais servidores do sistema. Estratégias centralizadas resultam em limitações de escalabilidade, que são indesejadas para plataformas planejadas para terem milhares de acessos, como é o caso no contexto de Cidades Inteligentes.

No sentido de propor uma solução para o problema identificado de armazenar as informações de repositórios de referências de forma centralizada, a plataforma *Sirius* trabalha com uma perspectiva distribuída, na qual, cada um de seus servidores tem a capacidade de gerenciar esquemas de entidades, através do componente CSM.

Em cada servidor *Sirius*, o componente CSM pode gerenciar e armazenar diversos esquemas de entidades de contexto. Vale ressaltar que a decisão de qual servidor da plataforma armazena o esquema de uma dada entidade de contexto depende exclusivamente da estratégia das organizações envolvidas na concepção, desenvolvimento e uso dos sistemas de software que fazem uso das respectivas informações contextuais da referida entidade de contexto. Desta forma, em um sistema com diversos servidores da plataforma, cada servidor pode ser responsável por armazenar diversos esquemas diferentes, evitando os problemas conhecidos, como um único ponto de falha, além de permitir vantagens como acesso concorrente, que possibilitam desempenho e escala.

Desta forma, o propósito deste componente é fornecer uma forma fácil para gerenciar um esquema de entidade, adotando uma estratégia que permite que diferentes servidores da plataforma gerenciem os esquemas sob sua responsabilidade, e que estes esquemas não estejam concentrados em um único e centralizado servidor.

3.2.2. Context Instance Manager - CIM

O componente *Context Instance Manager* (CIM) realiza as operações básicas de criar, acessar, atualizar e deletar instâncias de entidades de contexto em uma plataforma *Sirius* (operações conhecidas como CRUD – *Create, Read, Update, Delete*). Este componente atua como um gerenciador de instâncias de entidades de contexto, onde cada instância é definida por um esquema de entidade, que, por sua vez, é gerenciado pelo componente CSM, como já descrito anteriormente. Ou seja, enquanto o CSM é responsável por gerenciar os esquemas de entidade, o CIM é responsável por gerenciar as instâncias de entidade de contexto.

Antes de qualquer operação ser realizada com uma dada instância de entidade contexto, o CIM precisa recuperar o respectivo esquema da entidade de contexto. O esquema da entidade define os atributos e seus respectivos tipos de dados. Além disso, o esquema também sinaliza como os atributos podem ser particionados pelos provedores. Durante as

operações de criação e atualização, o componente CIM tem a responsabilidade de garantir a integridade dos dados, evitando a geração de informações contextuais inválidas ou inconsistentes.

O CIM detém funcionalidades importantes na plataforma *Sirius*, uma vez que realiza as operações responsáveis pela criação e atualização dos dados que representam as entidades de uma cidade.

3.2.3. Context Query Engine - CQE

O propósito do *Context Query Engine* (CQE) é realizar o processamento de consultas de um conjunto de entidades correlacionadas que se encontram em provedores de dados distribuídos e heterogêneos. Explorando os mecanismos decorrentes da utilização dos conceitos do metamodelo DCDS, o CQE provê um processo de integração de dados que consegue descobrir de forma automática e transparente a localização dos provedores e a estratégia de particionamento dos dados. Este componente também tem capacidade de realizar consultas geográficas, que representa um importante recurso no contexto de cidades inteligentes.

O CQE recebe como entrada informações básicas para a realização de uma consulta: o identificador do esquema da entidade (*schemaId*) de contexto da consulta; o nome do servidor (*ownerName*) *Sirius* que armazena o referido esquema; e a consulta propriamente dita na forma de uma cadeia de caracteres de consulta (*queryString*). A *queryString* expressa restrições que são aplicadas aos atributos de uma entidade de contexto. Ao final do processamento, este componente devolve como resposta as informações associadas às instâncias das entidades de contexto que atendem às restrições definidas.

A fim de alcançar seus objetivos o componente CQE necessita de um analisador léxico e sintático que permite reconhecer a linguagem de consulta da plataforma e conferir os atributos consultados de acordo com o esquema da entidade de contexto. A utilização de uma linguagem de consulta provê uma forma transparente e não ambígua de acessar os dados distribuídos. Outra função desempenhada pelo componente é o gerenciamento de várias requisições a outros servidores *Sirius*, que podem estar organizadas em duas estratégias diferentes de particionamento. Por fim, o CQE precisa ser capaz de realizar a decomposição de consultas em subconsultas, e, posteriormente, realizar a integração dos resultados das subconsultas retornadas pelos servidores da plataforma que participaram do processamento da consulta original.

Estes procedimentos realizados pelo componente CQE permitem que um servidor *Sirius*, ao receber uma requisição de consulta de uma aplicação externa entregue uma resposta única e completa como se os dados se encontrassem armazenados localmente. Porém, o processamento exige a localização, acesso e integração de dados armazenados em diversos provedores distribuídos geograficamente de forma transparente. Desta forma, o CQE proporciona transparência de localização à plataforma *Sirius*. Este componente também permite que recursos locais e remotos sejam acessados com o uso de operações idênticas, o que caracteriza a transparência de acesso.

3.2.4. Data Provider Adapter - DPA

O *Data Provider Adapter* (DPA) é um componente que recebe exclusivamente requisições do CQE local do servidor *Sirius*. A entrada é uma consulta definida pela sintaxe da linguagem de consulta adotada pelo componente CQE da plataforma, enquanto que a saída é o resultado da consulta ao provedor de dados local (que neste trabalho passa a ser chamado LDP – do inglês, *Local Data Provider*), com o formato de dados do provedor convertido para o modelo unificado de dados DCDS.

O propósito deste módulo é proporcionar interoperabilidade entre LDP e o servidor *Sirius*, no sentido de permitir que diferentes tipos de LDP´s sejam utilizados. Para que isto ocorra, o DPA provê transparência de representação de dados (HURSON; JIAO, 2005), conforme explicado na seção 2.2.

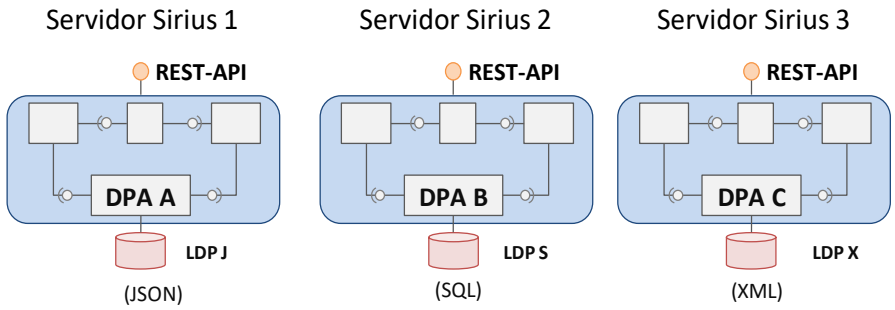
A fim de alcançar este objetivo o componente DPA exerce o papel de um adaptador (GAMMA, 1995) entre o LDP e o componente CQE da plataforma *Sirius*. Assim, o DPA precisa traduzir a consulta recebida do componente CQE para a linguagem de consulta do LDP. Através desta tradução, o DPA provê transparência de acesso às diferentes linguagens de consultas utilizadas pelos LDPs acoplados aos respectivos servidores *Sirius*. Ou seja, o componente CQE não se preocupa sobre o tipo de LDP que está sendo utilizado, apenas realiza consultas com a linguagem de consulta da plataforma *Sirius* e o componente DPA intermedia a interação e demais procedimentos, como tradução de linguagem de consulta e formato de dados.

Após receber o resultado de uma consulta do LDP o componente adaptador traduz as informações recebidas no formato do modelo de dados do LDP para o formato do modelo de dados DCDS. Através destes procedimentos, o DPA provê a transparência de representação em relação aos modelos de dados heterogêneos utilizados na estrutura de armazenamento.

Através destes procedimentos, que compreendem tradução da linguagem de consulta, gerenciamento de requisições e tradução do modelo de dados, o DPA provê os mecanismos para a utilização do modelo de dados DCDS como modelo de dados unificado da plataforma *Sirius*. A utilização de um modelo de dados unificado permite que os dados armazenados em LDP heterogêneos sejam consultados e integrados pelo CQE da plataforma *Sirius*, formando uma visão única do resultado que representam instâncias de entidades.

Como pode ser visto no exemplo da Figura 2, três diferentes LDP´s são utilizados por três servidores *Sirius*. Para cada tipo de LDP, existe um componente DPA específico. Cada DPA converte os dados dos formatos heterogêneos para o formato unificado, garantindo interoperabilidade para a plataforma.

Figura 2 : Servidores da plataforma de *middleware* com diferentes adaptadores (DPA)



3.3. Distributed Context Data Schema - DCDS

O metamodelo de dados *Distributed Context Data Schema* (DCDS) caracteriza-se por possibilitar tanto a especificação quanto a instanciação de dados de contexto associados a entidades do mundo real, compreendidas nos diversos domínios existentes das cidades inteligentes. Conforme apresentado nesta seção, o DCDS (*Distributed Context Data Schema*) pode ser adotado como um metamodelo de dados que provê interoperabilidade às plataformas de *middleware* distribuídas, proporcionando integração e interoperabilidade de dados correlacionados persistidos em provedores de dados distribuídos e independentes.

Para contextualizar os desafios existentes, uma breve descrição elencando os requisitos iniciais que nortearam o desenvolvimento do metamodelo DCDS pode ser assim resumida:

- Expressividade – está relacionada com a característica de resolver e representar elementos genéricos (BELLAVISTA, 2012), garantido uma vasta opção de

instanciação dos diversos tipos de dados encontrados nas entidades das cidades inteligentes.

- Simplicidade – é a característica de pouca exigência do número de estruturas, composição e regras de controle, tornando o processo de modelagem fácil e inteligível (JARA, 2014).
- Granularidade – é a característica do dado de contexto ser representado em diferentes níveis de detalhes, (BOLCHINI, 2007), incluindo entidades compostas, tipos de dados estruturados e exposição parcial de atributos.
- Interoperabilidade – permite a definição de esquemas não ambíguos para definir entidades e seus atributos (MCGEE, 1976), permitindo a troca de informações de forma transparente entre plataformas distribuídas.
- Reuso – potencializado por esquemas estruturados que agem como contratos de dados estruturados, incentivando o desenvolvimento de serviços baseados em dados nos quais os agentes do ecossistema podem publicar e compartilhar conjuntos de dados reutilizáveis (HAVELY, 2003).
- Integração – representa um passo além da reutilização, no qual os atores do ecossistema podem integrar outros conjuntos de dados externos, fornecendo serviços de valor agregado ou adaptando-se a diferentes propósitos (ABELLÁ-GARCÍA; ORTIZ-DE-URBINA-CRIADO; DE-PABLOS-HEREDERO, 2015).
- Particionamento – permite que os dados de contexto relacionados às entidades de cidades inteligentes sejam particionados ou divididos em vários provedores de dados (WHITE; TANTSURA, 2015), oportunizando o acesso simultâneo para aumentar o desempenho e a escalabilidade (BOUSSARD, 2013).

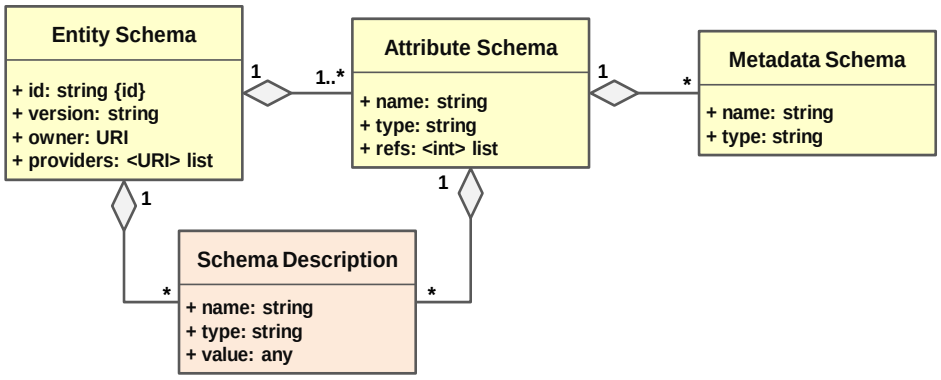
O metamodelo DCDS utiliza dois conceitos, esquema de entidade e instância de entidade para representar de forma não ambígua e única cada especificação e as múltiplas instâncias associadas a cada entidade de contexto, respectivamente.

A Figura 3 apresenta a representação UML (do inglês, *Unified Modeling Language*) de um esquema DCDS, no qual é possível observar dois elementos principais: *Entity Schema* e *Attribute Schema*.

O elemento *Entity Schema* corresponde a um esquema de entidade e possui os termos: *id* - provê um identificador único e global para cada esquema de entidade; *version* - indica a versão do respectivo esquema de entidade; *owner* - identifica de forma única o provedor

proprietário do esquema da entidade de contexto (ex: uma empresa, uma secretaria); *providers* - define a lista de múltiplos provedores nas quais as instâncias da entidade de contexto podem ser integralmente ou parcialmente armazenadas ou acessadas.

Figura 3 : Representação UML de esquema DCDS



Fonte: (LOPES; SILVA; ELIAS, 2019)

Cada esquema de entidade pode ter vários atributos, cada atributo é representado na perspectiva do metamodelo pelo elemento *Attribute Schema*. Este pode conter as seguintes partes: *name* - provê uma identificação única para cada atributo; *type* - indica o tipo do respectivo atributo (ex.: texto, inteiro e real); e *refs* - define a lista de referências para provedores nos quais o atributo pode ser armazenado ou acessado. A lista de referências tem um papel fundamental na localização dos servidores, provedores ou serviços que armazenam os dados dos atributos das instâncias de uma entidade de contexto.

Tanto o elemento *Entity Schema*, quanto o elemento *Attribute Schema* são essenciais para prover as funcionalidades de uma plataforma de *middleware* distribuído e particionado. É através das informações contidas na lista de provedores (*providers*) do *Entity Schema* e na lista de referências de cada atributo (*refs*) do *Attribute Schema* que componentes de integração e consulta, por exemplo, como o CQE, conseguem prover a transparência de localização em relação aos provedores distribuídos. Uma vez que a referência para provedores de dados independentes é organizada de forma separada da estrutura e do código do sistema (encontra-se no esquema DCDS), é possível realizar a reconfiguração dinâmica de novos provedores de dados em tempo de execução, ou seja, prover a adição, alteração e remoção de provedores sem a necessidade de interromper o funcionamento da plataforma de middleware e dos sistemas de software que fazem uso das informações contextuais.

É importante destacar que, através do elemento *Attribute Schema*, uma plataforma de *middleware* que adota o DCDS pode descobrir todos os atributos que compõem uma instância

de entidade de contexto e assim pode realizar a correta integração destes atributos que se encontram distribuídos.

Os outros dois elementos do metamodelo são *Metadata Schema* e *Schema Description*. Ambos são opcionais e permitem o enriquecimento de informações de contexto, tanto para o esquema, como para as instâncias, desde simples descrições textuais a registros de métricas e informações do contexto de captura. O elemento *Metadata Schema* permite que diferentes tipos de metadados sejam associados aos atributos das instâncias de entidades. Enquanto o elemento *Schema Description* representa os metadados do próprio esquema (*Entity* ou *Attribute*). Cada elemento *Metadata Schema* e *Schema Description* possuem dois termos, *name* e *type*, que atribuem um identificador único para cada metadado e indica o tipo de metadados utilizado, respectivamente. Cada *Schema Description* possui o termo *value* para representar o conteúdo específico do metadados.

A partir da utilização de esquemas, o DCDS provê uma forma simples, flexível e expressiva de descrever de forma não ambígua uma entidade de contexto no cenário das cidades inteligentes. Como outra vantagem advinda da adoção de esquemas, cada esquema de entidade pode servir como referência para analisadores sintáticos verificarem a validade de dados de entidades de contexto distribuídas, tornando possível a interoperabilidade e integração de dados reutilizados a partir de diferentes provedores.

Os elementos do DCDS permitem a construção de uma plataforma que gerencia dados distribuídos, pois a partir da lista de provedores relacionados a cada atributo é possível localizar estes dados de forma automática e transparente.

Para ser persistido e utilizado por um serviço, sistema ou aplicativo, o DCDS precisa ser convertido para uma linguagem de representação. Atualmente existe uma grande diversidade de linguagens, como XML, CSV (*Comma-Separated Values*) e RDF (*Resource Description Framework*). Foi escolhido a linguagem de representação JSON (*JavaScript Object Notation*) (JSON, 2019), que tem sido muito utilizada em projetos IoT, pela sua simplicidade, tamanho, leveza e inteligibilidade, principalmente se comparado com o XML (ZUNKE; D'SOUZA, 2014). A Figura 4 mostra um esquema de DCDS representado com JSON.

O exemplo de representação JSON do esquema DCDS (Figura 4) possui o parâmetro 'bus' (*id*) como identificador do esquema de entidade; '1.0' (*version*) indica a versão do esquema; 'dcds.provider.br' (*owner*) define o nome do provedor do esquema da entidade de contexto (utilizando o *id* e o *owner* se tem a identificação única e global de um esquema de entidade DCDS); '["dcds.provider1.br", "dcds.provider2.br", "dcds.provider3.br"]' (*providers*)

indica os nomes dos provedores em que as instâncias da entidade de contexto podem ser integralmente ou parcialmente armazenadas ou acessadas. Os outros quatro termos presentes na representação referem-se aos atributos do esquema (*Atribute Schema*), cada atributo possui os termos *name*, *type* e *refs*. Observe que, na representação JSON adotada, o termo *name* não aparece de forma explicita, porém ele está representado como o nome do atributo (*location*, *speed*, *people-amount* e *temperature*).

Figura 4 : Representação JSON de um Esquema DCDS

```
{
"id":"bus",
"version":"1.0",
"owner":"dcds.provider1.br",
"providers":["dcds.provider1.br", "dcds.provider2.br",
"dcds.provider3.br"],
"location":{
   "type":"point",
   "refs":[1, 2]},
"speed":{
   "type":"double",
   "refs":[1, 3]},
"people-ammount":{
   "type":"integer",
   "refs":[1, 2, 3]},
"temperature":{
   "type":"double",
   "refs":[1, 3]}
}
```

Fonte: (LOPES; SILVA; ELIAS, 2019).

Para melhor entendimento da representação JSON de um esquema, é detalhado o primeiro atributo, este possui o nome ‘location’ e é do tipo *point*, ou seja, é uma coordenada geográfica que corresponde à localização da entidade. Este atributo pode estar armazenado em dois provedores, definidos pela lista do termo *refs*, indicando a referência para os provedores contidos no termo *providers*, neste caso aos provedores ‘dcds.provider1.br’ e ‘dcds.provider2.br’.

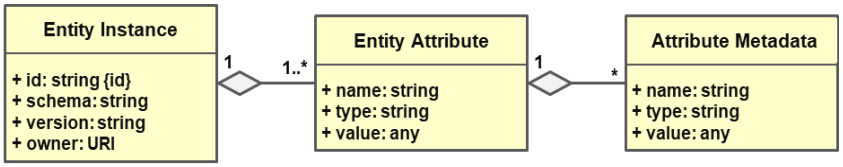
A partir desta representação de esquema de entidade, uma aplicação que adota esquemas DCDS e deseja acessar os dados do atributo ‘location’ das instâncias da entidade definida pelo esquema DCDS ‘bus’ cujo proprietário é ‘dcds.provider1.br’ deve realizar as seguintes etapas. Acessar o provedor ‘dcds.provider1.br’, carregar o esquema de entidade identificado por ‘bus’, confirmar a existência de um atributo identificado por ‘location’, identificar os provedores utilizados para armazenar os dados deste atributo, de acordo com as informações contidas em ‘refs’ e ‘providers’. Após ler todas estas informações estará ciente dos provedores que contém os dados do atributo ‘location’ das instâncias da entidade ‘bus’.

O Esquema DCDS possibilita a automação das etapas envolvidas na recuperação de informações distribuídas, facilitando o trabalho de desenvolvedores de sistemas para cidades inteligentes. Uma aplicação externa que solicita informações da entidade ‘bus’ do proprietário

‘dcds.provider1.br’ não toma conhecimento em relação aos procedimentos realizados e a infraestrutura distribuída existente para a entrega do resultado.

Para plataformas, sistemas e serviços de cidades inteligentes, cada entidade física do mundo real (por exemplo, celulares, pessoas e sensores) pode ser modelada e representada como uma entidade de contexto virtual, a qual é chamada de instância de entidade DCDS. Através de representação UML, como exposto na Figura 5, uma instância de entidade é formada por três elementos. Sendo que, cada instância de entidade, segue as especificações de seu respectivo esquema de entidade.

Figura 5 : Representação UML de entidade de contexto



Fonte: (LOPES; SILVA; ELIAS, 2019).

Utilizando-se novamente das informações da Figura 5, uma instância de entidade possui os seguintes elementos: a) *Entity Instance*, constituído pelos termos: *id* - provê um identificador para cada instância de entidade; *schema* - o respectivo esquema que especifica o tipo de instância de entidade; *version* - a versão do respectivo esquema de entidade; *owner* - identifica o provedor proprietário que armazena o esquema (por exemplo, o endereço URL). A partir da utilização em conjunto dos termos *id*, *schema*, *version* e *owner*, é possível definir um identificador global único para cada instância de entidade, permitindo assim que plataformas de middleware acessem de forma transparente as instâncias de entidades.

Para cada instância de entidade podem existir diversos atributos, que são representados a partir dos elementos *Entity Attribute*. Os termos presentes em um *Entity Attribute* são: *name* - indica o nome do atributo; *type* - define o tipo do respectivo atributo (por exemplo, texto, coordenadas geográficas); e *value* - representa o valor atual de um atributo de uma determinada instância de entidade.

Como especificado em um *Entity Schema*, cada instância de entidade pode ter, de forma opcional, metadados associados para prover informações contextuais sobre certo atributo. Para isto, o DCDS adota o elemento *Attribute Metadata*, com seus termos *name*, *type* e *value*, que representam respectivamente o nome, o tipo e o valor do metadado. Observe que os termos *name* e *type* são previamente definidos no esquema da entidade.

A Figura 6Figura 6 apresenta um exemplo simples de uma instância de entidade DCDS representada em JSON, de acordo com o esquema de entidade DCDS representado em JSON na Figura 4. Esta representa uma instância de entidade que possui como identificador ‘CT01-1 Circular-Tourism’ (*id*), que pertence ao esquema de entidade ‘bus’ (*schema*), enquanto a entidade ‘bus’ pertence a um provedor de nome ‘dcds.provider1.br’ (*owner*) e a versão do esquema é ‘1.0’ (*version*). Esta instância de entidade possui quatro atributos, cada atributo possui o seu nome identificador e os termos *type* e *value*, o primeiro caracteriza o tipo de atributo e foi especificado no esquema da entidade, enquanto o segundo corresponde ao valor do atributo e deve obedecer ao tipo definido. Como exemplo, é detalhado o primeiro atributo, que possui como nome ‘location’, é do tipo *point* (o tipo *point* determina uma localização geográfica representada por dois ou três números reais - latitude, longitude e elevação, onde o valor da elevação é opcional) e possui como valor a coordenada geográfica ‘-23.590, -46.610’.

Figura 6 : Representação JSON de uma instância de entidade DCDS

```
{
"id":"CT01-1 Circular-Tourism",
"schema":"bus",
"version":"1.0",
"owner":"dcds.provider1.br",
"location":[-23.590, -46.610],
"speed":60.35,
"people-ammount":32,
"temperature":22.25
}
```

Fonte: adaptado de (LOPES; SILVA; ELIAS, 2019)

A partir do modelo de dados derivado de um esquema DCDS (chamado neste trabalho de modelo de dados DCDS), uma plataforma de *middleware* pode alcançar transparência de representação ao adotá-lo como modelo de dados unificado, ou seja, consegue interoperar os dados armazenados em diferentes provedores de dados, mesmo quando adotam modelo de dados heterogêneos.

3.4. Considerações Finais

Neste capítulo foi apresentada a visão geral da plataforma *Sirius*, uma plataforma de *middleware* ao qual a abordagem de processamento de consulta proposta neste trabalho se integra. Foram apresentados os principais componentes e suas funcionalidades.

Um aspecto interessante da arquitetura da plataforma *Sirius* é a independência entre os quatro componentes que a compõe. Levando em consideração que as interfaces dos componentes são claramente definidas, é possível a mudança, por exemplo, das técnicas e algoritmos empregados na implementação de qualquer componente, sem causar impacto nos demais. Por outro lado, novos componentes podem ser adicionados à arquitetura, tendo a opção de usufruir dos componentes já existentes, para agregar novos recursos, por exemplo, adicionando novas funcionalidades específicas de um determinado domínio de aplicação no contexto de Cidades Inteligentes.

Foi apresentado também o metamodelo DCDS e as características que nortearam o seu desenvolvimento, incluindo os dois conceitos introduzidos de esquema e instância de entidade, bem como os elementos que integram cada um destes conceitos. O esquema de entidade identifica de forma única uma entidade de contexto e define sua estrutura, seus atributos e a localização dos provedores que podem armazenar as instâncias de entidades. Um esquema DCDS possui como característica permitir que as plataformas que o utiliza alcancem mecanismos almejados para sistemas distribuídos, como a transparência de localização e a transparência de particionamento.

O outro conceito do metamodelo DCDS apresentado foi o conceito de instância de entidade, que é um modelo de dados capaz de representar as múltiplas instâncias associadas a cada entidade de contexto. O modelo de dados DCDS pode ser adotado como modelo de dados unificado de uma plataforma distribuída e permitir que, com o uso de adaptadores, diferentes modelos de dados possam ser traduzidos e, assim, se alcance a transparência de representação, possibilitando interoperar os dados armazenados em diferentes provedores de dados. Para cada conceito apontado, foi apresentada uma representação com diagrama UML e sua respectiva conversão para a linguagem de representação JSON.

Tanto a proposta de plataforma de *middleware*, quanto o metamodelo DCDS retratados no decorrer deste capítulo, foram concebidos em parceria com os pesquisadores do Laboratório COMPOSE da UFPB. Isto posto, o presente trabalho possui como escopo uma abordagem de consulta para plataformas de *middleware* distribuído e particionado no contexto de Cidades Inteligentes. No próximo capítulo serão detalhados a abordagem de processamento de consulta, a interação entre os componentes da plataforma *Sirius* em uma consulta distribuída e particionada, e, por fim, o processamento interno dos componentes (CQE e DPA) envolvidos na consulta.

4. Uma Abordagem para Processamento de Consultas Distribuídas e Particionadas

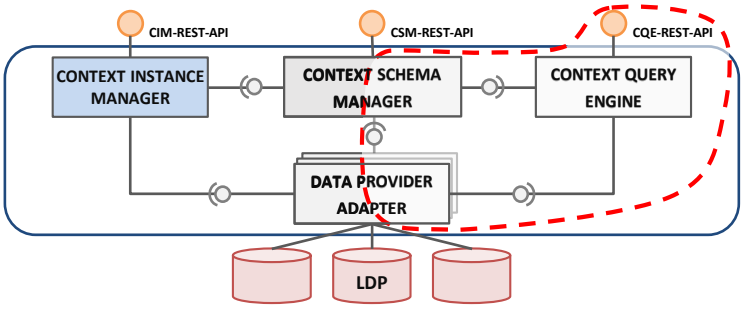
Este capítulo apresenta a abordagem de processamento de consultas distribuídas e particionadas para plataformas de *middleware* no contexto das cidades inteligentes. Esta abordagem faz parte da plataforma *Sirius* e utiliza esquemas e instâncias de entidades de contexto, especificadas e representadas usando o metamodelo de dados DCDS (LOPES; SILVA; ELIAS, 2019), para transpor os problemas de interoperabilidade e escalabilidade encontrados no cenário de sistemas, ferramentas e aplicativos para cidades inteligentes.

Inicialmente, é apresentada a interação existente entre os componentes da plataforma *Sirius* (Figura 8Figura 8) durante um processamento de uma consulta envolvendo vários servidores da plataforma. Na sequência, são apresentados os componentes CQE e DPA, assim como os detalhes da estrutura interna de cada um.

O objetivo da abordagem proposta é apoiar o desenvolvimento de serviços, sistemas e aplicações para cidades inteligentes, para que desenvolvedores tenham maior facilidade nos procedimentos de consultas em cenários em que envolve a integração de sistemas heterogêneos e distribuídos, através da interoperabilidade de provedores.

A Figura 7 apresenta com a linha tracejada os componentes desenvolvidos durante este trabalho. É possível identificar um componente totalmente dentro desta linha tracejada e outros com parte de sua área intersecionando com esta linha. Os componentes que estão com parte de sua área demarcada foram idealizados em parceria com a equipe do laboratório COMPOSE da UFPB. Na Figura 7 está representado três componentes: CQE, CSM e DPA. O componente CQE (responsável pelo processamento de consultas distribuídas) está dentro da área tracejada e foi desenvolvido no atual trabalho. No caso do componente DPA, este trabalho contribuiu parcialmente com as funcionalidades relacionadas a processamento de consulta, mais especificamente realizando a tradução da expressão de consulta da linguagem adotada na plataforma *Sirius* para a linguagem de consulta adotada pelo provedor. Outro módulo que teve contribuição parcial deste trabalho foi o componente CSM, que contribuiu na etapa de formulação do metamodelo DCDS e dos seus elementos (esquemas e instâncias de entidade).

Figura 7 : Componentes relacionados a abordagem de consulta distribuída em destaque

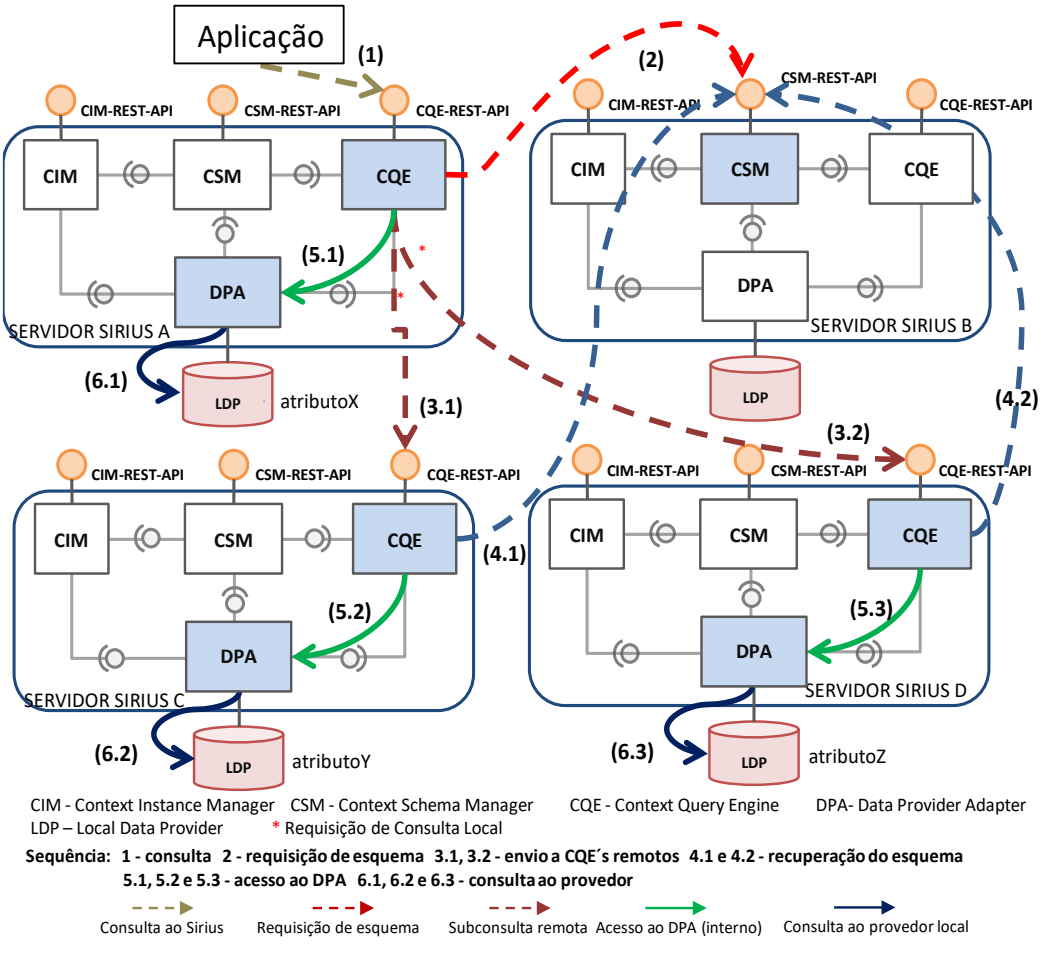


Fonte: adaptado de (LOPES; SILVA; ELIAS, 2019)

4.1. Interação entre Componentes

Esta seção apresenta as interações existentes entre os componentes da plataforma *Sirius* durante o processamento de uma consulta. No exemplo da Figura 8, uma aplicação cliente realiza uma consulta para um servidor *Sirius* que, durante o processamento, necessita da colaboração de outros servidores *Sirius* para entregar a resposta correspondente.

Figura 8 : Interação de componentes durante processamento de consulta



Primeiro, a aplicação realiza a consulta para o servidor *Sirius* A através da interface do componente CQE (1), que deve seguir padrões de comunicação como uma API REST. Esta operação contém três parâmetros: o domínio ou endereço (chamado de *ownerName*) do servidor *Sirius* (B) que gerencia e armazena o esquema da entidade de contexto utilizada na consulta; o identificador do referido esquema da entidade de contexto (*schemaId*); e a expressão de consulta (*querystring*).

A próxima interação ocorre entre o componente CQE do servidor A e o componente CSM do servidor *Sirius* (B) que gerencia o esquema da entidade de contexto utilizada na consulta. Nesta interação, o componente CQE do servidor A acessa (2) o esquema de entidade *schemaId*. O esquema recebido como resposta contém, como todo esquema do DCDS, duas informações fundamentais para o processamento de uma consulta distribuída: os atributos que formam a entidade e os endereços dos provedores de dados responsáveis por armazenar os dados destes atributos. Todo provedor de dados é acessado por intermédio de um servidor *Sirius*, e, por isto, neste trabalho, ao se referenciar um servidor *Sirius*, está implícito a existência do provedor de dados.

De posse do esquema DCDS, o CQE realiza os seguintes procedimentos:

1) Valida se os atributos presentes na *querystring* correspondem aos especificados no esquema;
2) Quebra a consulta em subconsultas;
3) Identifica os servidores *Sirius* que armazenam de forma distribuída e particionada dados das instâncias da entidade de contexto;
4) Prepara subconsultas específicas para os respectivos servidores *Sirius*.

Após a preparação das subconsultas, o CQE gerencia e encaminha cada subconsulta para os respectivos servidores *Sirius* (C e D) (3.1 e 3.2) que armazenam atributos referenciados pela consulta original. Nestes dois servidores, o componente que recebe a subconsulta é o CQE. Porém, neste caso, o processamento da consulta ocorre de forma diferente do que ocorreu no servidor A e retorna apenas informações existentes no próprio servidor, em virtude da utilização de um parâmetro de cabeçalho (processingType=localData) do CQE que determina o processamento exclusivamente local. Ao disparar as duas subconsultas remotas, o servidor A adiciona este parâmetro. O parâmetro ‘processingType=localData’ evita que ocorra ciclos de requisições, que podem gerar *deadlock* de comunicação (DASZCZUK, 2017). Em sistemas distribuídos, *deadlock* de comunicação ocorre quando um sistema S encaminha uma requisição ao sistema R e aguarda a resposta. Por sua vez, o sistema R encaminha uma requisição ao sistema T e aguarda a resposta. Na

sequência, o sistema T encaminha outra requisição ao sistema S e também aguarda a resposta, gerando um ciclo e inviabilizando a comunicação por *deadlock*.

Ao receber as subconsultas com escopo local, cada servidor remoto *Sirius* (C e D) solicita o esquema da entidade de contexto referenciada na subconsulta para o servidor *Sirius* (B) (4.1 e 4.2) e em seguida é realizada a validação da subconsulta com base no esquema da entidade. Os passos executados até aqui são os mesmos realizados pelo servidor (A), alterando a expressão (subconsulta) a ser validada. Na sequência, o CQE de cada servidor *Sirius* envia para o seu próprio DPA (5.1, 5.2, 5.3) uma requisição contendo as respectivas subconsultas na linguagem de consulta do CQE *Sirius*. Cada DPA traduz a subconsulta para a linguagem do provedor de dados local e envia para cada LDP a consulta traduzida (6.1, 6.2, 6.3). Cada LDP encaminha o resultado em seu próprio modelo de dados, sendo de responsabilidade do DPA realizar a tradução para o modelo de dados DCDS. Terminada a tradução, o DPA encaminha o resultado para o respectivo CQE local (esta etapa não está representada Figura 8).

Após estas etapas envolvendo o servidor A e três servidores remotos, o componente CQE do servidor A recebe três conjuntos de dados no formato do modelo de dados DCDS. Estes são integrados em apenas um conjunto coeso, ou seja, uma resposta correspondente com a consulta enviada pela aplicação cliente. Por fim, uma mensagem no formato JSON é encaminhada para a aplicação cliente com o resultado. A plataforma *Sirius* devolve uma resposta completa, ou seja, atômica. Caso um único servidor *Sirius*, presente como destinatário das subconsultas ou como responsável por gerenciar o esquema de entidade (Servidores B,C, D) esteja inacessível, uma resposta de erro é retornada.

O comportamento do primeiro servidor *Sirius* a receber a requisição da aplicação externa (servidor A), e, em seguida, enviar novas requisições a outros dois servidores *Sirius* que contêm as informações desejadas, é semelhante a um *mediator* da arquitetura de sistemas de integração de dados na web chamada *mediator-wrapper* (AYRES, 2003), onde um mediador centraliza as informações de fontes de dados distribuídas e heterogênea (AYRES, 2003).

As próximas seções abordam com mais detalhes os componentes CQE e DPA, responsáveis pelo processamento da consulta distribuída.

4.2. CQE

O componente CQE (*Context Query Engine*) da plataforma *Sirius* utiliza o esquema de entidade DCDS como uma espécie de mapa para proporcionar a localização dos dados nos

servidores distribuídos. Também é a partir do DCDS que é derivado um modelo de dados unificado, utilizado pelos diversos componentes, peça chave para a interoperabilidade entre modelos heterogêneos de dados.

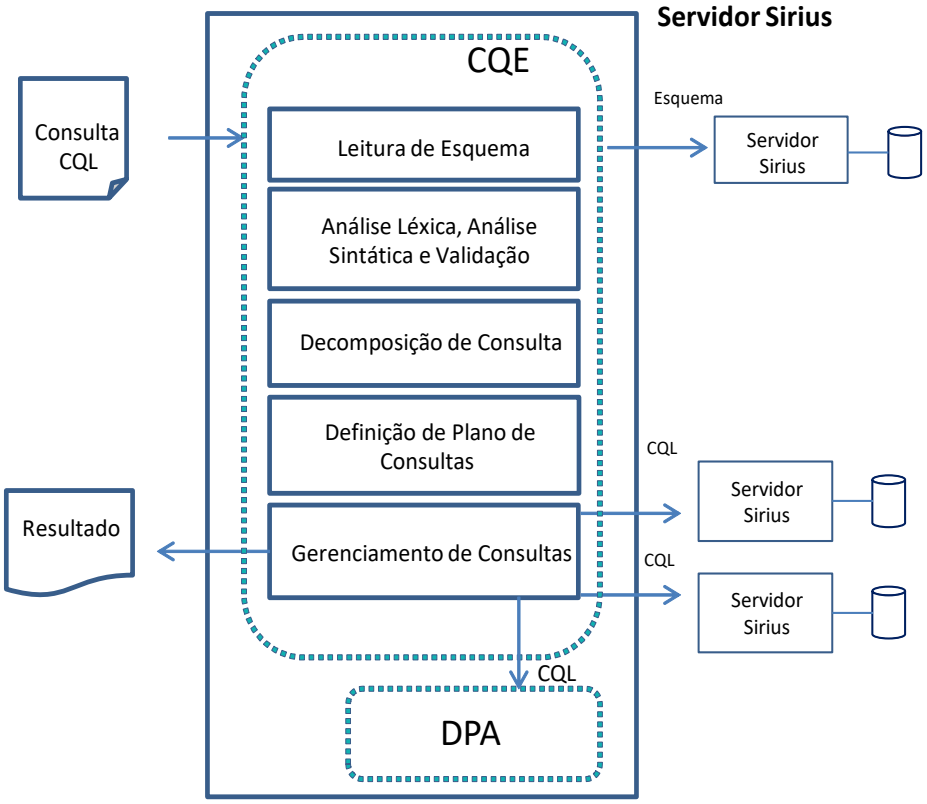
O objetivo do CQE é prover acesso através de consultas a um conjunto de informações inter-relacionadas, dispersas por provedores de dados independentes e geograficamente distribuídos. Uma linguagem de consulta de contexto (CQL) de alto nível é utilizada para acessar de forma transparente os dados das instâncias de contexto, ocultando detalhes como quantidade, estrutura de particionamento e localização dos servidores *Sirius* utilizados para armazenamento dos dados.

A Figura 9 ilustra a arquitetura modular do CQE. Resumidamente, o processamento de consultas distribuídas do CQE segue o seguinte fluxo:

a) O módulo de *Leitura de Esquema* acessa o servidor proprietário do esquema de entidade consultado;
b) O módulo de *Análise Léxica, Análise Sintática e Validação* analisa a consulta recebida em termos léxicos e sintáticos, e, em seguida, valida os atributos de acordo com o esquema DCDS;
c) O módulo de *Decomposição de Consulta* cria subconsultas para cada atributo da consulta;
d) O módulo de *Definição de Plano de Consultas* formula o plano de consultas considerando a localização das informações distribuídas e a estrutura de particionamento;
e) O módulo de *Gerenciamento de Consultas Distribuídas* distribui as subconsultas com servidores remotos e para o módulo local DPA, integra as respostas e devolve o resultado para a aplicação externa.

O processamento de consultas em sistemas distribuídos tem uma complexidade maior que em sistemas centralizados, envolve maior número de procedimentos, como, por exemplo, comunicação entre servidores, divisão de consulta, localização de servidores, e integração dos dados.

Figura 9 : Módulos internos do componente CQE



4.2.1. Interface do CQE

O CQE provê uma API RESTFul com o protocolo HTTP para a realização de consultas diretas, permitindo assim o acesso de aplicações externas aos dados das instâncias de entidades armazenadas em diversos servidores *Sirius*.

O acesso a consulta é fornecida através do método GET com a seguinte estrutura:

GET /entities/{ownerName}/{schemaId}?q={queryContext}&
georel={georelCont}&geometry={geometryCont}&coords={coordsCont}

De acordo com esta estrutura, *entities* é o recurso RESTful (MASSE, 2011) que se está requerendo a informação. Na sequência, *ownerName* refere-se ao nome do servidor *Sirius* proprietário do esquema da entidade que se quer informações (por exemplo 'dcds.provider1.br'), *schemaId* é o identificador do esquema de entidade. Em seguida, o parâmetro 'q' é utilizado para identificar a expressão de consulta, chamada aqui de *queryContext*. Um exemplo de uma consulta através desta interface que retorna apenas as entidades do tipo 'bus' cujo esquema é gerenciado pelo servidor *Sirius* 'dcds.provider1.br' está representado a seguir. Observe que aqui é representado uma consulta de forma a facilitar

compreensão, com termos que são traduzidos pelos *browser* antes de realizar uma requisição HTTP, como é o caso do caractere '>', que é substituído por '%3E':

GET /entities/dcds.provider1.br/bus?q=speed>90

Os próximos três parâmetros *georel*, *geometry* e *coords* fazem parte das *Geographical Queries*. Podem expressar uma consulta que retorna as entidades que se localizam a um raio mínimo de uma coordenada geográfica, ou as entidades que encontram-se fora de uma área de um polígono, ao se determinar as extremidades. Um exemplo deste tipo de consulta, retornando as entidades que estão a menos de 1.000 metros de um ponto de referência é:

GET /entities/dcds.provider1.br/bus?georel=near;maxDistance:1000&
geometry=point&coords=-23.590,-46.610

Outra característica desta interface diz respeito a um parâmetro de cabeçalho. Durante o desenvolvimento da abordagem de consulta foi identificada a necessidade da plataforma *Sirius* ter um mecanismo para impedir a formação de *deadlock* de comunicação (DASZCZUK, 2017). Em decorrência disto, foi definido um parâmetro de cabeçalho HTTP (MASSE, 2011) para informar que um servidor *Sirius* deve realizar apenas processamento de seus dados locais. Para isto, o parâmetro 'Processing-Type' deve estar atribuído com valor 'localData' e ser encaminhado na mesma mensagem GET da consulta, conforme discutido na seção 4.1.

Cada consulta retorna os atributos presentes na consulta, desta forma, o resultado da primeira consulta é composta pelos identificadores da entidade, além do atributo *speed*, com o respectivo valor (conforme pode ser visto na Figura 10). Na seção 4.2.3.1 a linguagem de consulta de contexto da plataforma *Sirius* é apresentada com mais detalhes.

Figura 10 : Exemplo da representação JSON de um resultado de consulta não geográfica


```
[{
"id":"CT01-5 Circular-Tourism",
"schema":"bus",
"version":"1.0",
"owner":"dcds.provider1.br",
"speed":92.35
},
{
"id":"CT01-9 Circular-Tourism",
"schema":"bus",
"version":"1.0",
"owner":"dcds.provider1.br",
"speed":95.00
}]
```

4.2.2. Leitura de Esquema de Entidade

O módulo de *Leitura de Esquema de Entidade* é responsável por acessar o servidor *Sirius* que gerencia o esquema de entidade da consulta, informado na interface de consulta (*ownerName*). Ele se comunica com o componente CSM do servidor *Sirius* remoto através do protocolo HTTP. No entanto, o *ownerName* pode também referenciar o próprio servidor, e, neste caso, é realizado uma requisição interna.

Após ter acesso ao esquema de entidade, este esquema fica disponível localmente para os demais módulos durante o processamento da consulta.

4.2.3. Análise Léxica, Análise Sintática e Validação

O módulo de *Análise Léxica, Análise Sintática e Validação* verifica se a consulta atende aos critérios da linguagem de consulta de contexto (CQL) da plataforma *Sirius* e se está de acordo com o esquema de entidade acessado previamente pelo módulo de *Leitura de Esquema de Entidade*. Quando um erro léxico ou sintático é detectado, uma resposta é devolvida à aplicação cliente com uma descrição do erro, caso contrário, o processamento da consulta continua.

Uma consulta é classificada como inválida se qualquer de seus atributos não for definido no esquema DCDS da respectiva entidade de contexto, ou ainda se forem aplicadas operações não suportadas pelos atributos. Como exemplo, considere que um esquema DCDS define que o atributo ‘temperature’ pertence a entidade ‘bus’. Se uma queryContext expressa ‘temp>10’, o servidor *Sirius* retornará uma mensagem de erro informando que o atributo ‘temp’ não existe no esquema ‘bus’.

4.2.3.1. Linguagem de Consulta de Contexto

Com o objetivo de oferecer uma linguagem de consulta de alto nível, conforme os requisitos elencados no Capítulo 2, este trabalho utiliza uma sintaxe adaptada da linguagem de consulta da plataforma para internet do futuro FIWARE (FIWARE, 2019d). Esta plataforma utiliza parte dos conceitos da linguagem definida como *Simple Query Language* e dos parâmetros *Geographical Queries*, que pertencem à especificação NGSI-v2 (FIWARE, 2019a) para a realização de consultas e acesso aos dados do *Orion Context Broker* (FIWARE, 2019b). Vale ressaltar que, nos últimos anos, a especificação NGSI-v2 tem sido adotada por vários projetos, em um esforço de se estabelecer um padrão de comunicação para projetos no contexto de Cidades Inteligentes.

A linguagem *Simple Query Language* possui a expressividade necessária para realizar consultas e retornar as entidades que atendem simultaneamente às restrições especificadas em um conjunto de declarações não geográficas e separadas pelo operador lógico AND. Ou seja, a linguagem é capaz de reconhecer uma sequência de predicados conjuntivos, que podem ser expressos da seguinte maneira: $(p_{11} \wedge p_{12} \wedge p_{13} \wedge ... \wedge p_{1n})$. Para expressar consultas geográficas são especificados parâmetros chamados de *Geographical Queries*.

O atual trabalho realizou uma pequena evolução na sintaxe da linhagem *Simple Query Language* e, com o intuito agregar mais expressividade, incorporou o operador lógico disjuntivo. Com isto, a CQL do atual trabalho permite que sentenças conjuntivas sejam sequenciadas pelo operador disjuntivo. Assim, para reaproveitar as operações nativas da linguagem original, é dada a precedência para o operador conjuntivo. Em SGBDs relacionais esta forma de predicado é chamada de forma disjuntiva normal e pode ser representado da seguinte forma: $(p_{11} \wedge p_{12} \wedge p_{13} \wedge ... \wedge p_{1n}) \vee ... \vee (p_{m1} \wedge p_{m2} \wedge p_{11} \wedge p_{m3} \wedge ... \wedge p_{mn})$.

Para permitir um melhor entendimento da linguagem de consulta adaptada foi elaborada sua representação formal através de uma gramática EBNF (do inglês *Extended Backus-Naur Form*), conforme mostrado na Figura 11. A partir desta representação formal é possível o desenvolvimento de interpretadores para a linguagem de consulta proposta.

A linguagem de consulta em questão ainda está em evolução e o seu atual estágio contempla consultas a atributos, assim como também já trata sequências de sentenças conjuntivas unidas pelo operador disjuntivo. Consultas relacionadas a metadados de atributos fazem parte de trabalhos futuros.

Figura 11 : Representação da gramática EBNF para linguagem de consulta

```
queryContext ::= conjuntive_query_1 disjuntive_query_1
disjuntive_query_1 ::= (or_operator queryContext )*
conjuntive_query_1 ::= simple_query conjuntive_query_2
conjuntive_query_2 ::= (and_operator conjuntive_query_1)*
simple_query ::= attribute_name op_1 value_t_1
                 | attribute_name  op_2 value_t_2
                 | attribute_name  op_3 value_t_3
or_operator ::= "|"
and_operator ::= ";"
op_1 ::= "==" | "!="
op_2 ::= "<" | "<=" | ">" | ">="
op_3 ::= "~="
value_t_1 ::= string | float (".." float)? | integer (".." integer)?
value_t_2 ::= string | float | integer
value_t_3 ::= string
```

O símbolo “*queryContext*” representa como uma consulta pode ser composta, que, de acordo com a gramática apresentada, é formado por uma sequência de predicados conjuntivos ou conjunto de predicados conjuntivos sequenciados pelo operador disjuntivo. Isto porque o símbolo “disjuntive_query_1” pode ser uma expressão vazia (o que ocasiona uma sequência de predicados conjuntivos) ou derivar em novos símbolos “*queryContext*”, podendo se repetir várias vezes, o que permite que se forme um conjunto de predicados conjuntivos sequenciados pelo operador disjuntivo. O operador disjuntivo (*or_operator*) é representado pelo caractere “|”, enquanto o operador conjuntivo (*and_operator*) é representado pelo caractere “;”.

O símbolo “conjuntive_query_1” representa um predicado simples ou um conjunto de predicados conjuntivos. Um predicado simples (“*simple_query*”) é formado por um atributo, um operador e um valor. Os operadores podem ser:

- Igual: ‘==’. Este operador pode ser utilizado com valores do tipo texto, inteiro e real, das seguintes formas:
    - Forma simples, por exemplo *speed==10*. Para as entidades cujo atributo (*speed*) contém o valor *value_t_1* (10).
    - Forma tipo ‘*range’*, por exemplo *speed==10..50*. É especificado um valor mínimo e um valor máximo, separado por ‘..’. As entidades devem possuir o valor do atributo consultado entre o valor máximo e mínimo (incluindo os próprios valores).
- Diferente: ‘!=’. Este operador pode ser utilizado com texto, inteiro e real e é semelhante ao operador igual. Porém o resultado é inverso.
    - Forma simples, por exemplo *speed!=10*. Para todas as entidades cujo atributo (*speed*) não contém o valor *value_t_1* (10).
    - Forma tipo ‘*range’*, por exemplo *speed==10..50*. É especificado um valor mínimo e um valor máximo, separado por ‘..’. As entidades devem possuir o valor do atributo consultado fora da faixa dos valores mínimo e máximo (incluindo os próprios valores).
- Operadores de comparação: Maior que ‘>’, Maior ou igual ‘>=’, Menor que ‘<’, Menor ou igual ‘<=’. Estes operadores provêm o comportamento esperado e comumente utilizado de comparação. São aplicados para valores do tipo texto, real e inteiro.

- Identificador de padrão textual ‘~=’. O valor deve seguir um determinado padrão, sinalizado em expressão regular, por exemplo ‘source~=praça’. Retorna as entidades (bus), cuja origem da linha contém o nome ‘praça’ (‘Praça da República’ e ‘Praça da Sé’ seriam valores válidos, enquanto ‘Liberdade’ não seria).

O símbolo “attribute_name” é utilizado em uma “simple_query” e corresponde aos atributos pertencentes a uma entidade de contexto, que são definidos pelo respectivo esquema DCDS. Então, os nomes correspondentes ao símbolo “attribute_name” são validados com os nomes dos atributos da entidade consultada pelo módulo Análise Léxica, Análise Sintática e Validação. Além dos nomes dos atributos, também são validados os valores “value_t_1”, “value_t_2” e “value_t_3”, com processo semelhante, a partir dos tipos de valores especificados no esquema.

Para representar consultas geográficas foi utilizado os parâmetros *Geographical Queries* do NGSI-v2: *georel*, *geometry* e *coords*. A expressividade destes parâmetros permite realizar as principais operações geográficas, como por exemplo retornar as entidades situadas dentro de um raio de uma coordenada geográfica.

O parâmetro *georel* permite especificar uma relação espacial (por exemplo relação de proximidade ou de interseção) entre instâncias de entidade de contexto e uma forma referenciada (que será visto no parâmetro *geometry*). É composto por uma lista de termos separados por ‘;’. O primeiro termo é relacionado ao nome, e os demais termos (se forem necessários) provêm mais informações sobre esta relação (por exemplo, a distância máxima). O parâmetro *georel* contém os seguintes valores: *near, coveredBy, intersects, equals, disjoint*. Apenas *near* exige mais termos, que são *maxDistance* e *minDistance*. Como exemplo, uma consulta que expressa as entidades que estão localizadas ao menos 1.000 metros de um ponto de referência é representada da seguinte forma:

georel=near;maxDistance:1000&geometry=point&coords=-40.4,-3.5

Os outros parâmetros são *geometry* e *coords*. *Geometry* define a forma que será considerada para a resolução da consulta. Para isto permite que seja utilizado os seguintes termos: *point, line, polygon, box*. Enquanto *coords* contém pares de coordenadas geográficas em concordância com a forma geográfica utilizada. Outro exemplo de consulta geográfica, em que são retornadas entidades localizadas dentro de um polígono especificado:

georel=coveredBy&geometry=polygon&coords=25.774,-80.190;18.466,-66.118;32.321,-64.757;25.774,-80.190

4.2.4. Decomposição de Consulta

O módulo de *Decomposição de Consulta* tem como objetivo dividir a *queryContext* recebida em subconsultas formadas por um único atributo. Este processo permite que os próximos módulos manipulem subconsultas e explorem melhor a estrutura distribuída de servidores *Sirius*. Portanto, uma *queryContext* composta por dois atributos, por exemplo envolvendo os atributos 'speed' e 'temperature', formando uma única consulta 'speed<30;temperature>30', deve ser decomposta em duas subconsultas.Assim, ao sair da Decomposição de Consulta formam duas subconsultas, cada uma com um atributo: 'speed>10' e 'temperature>30*'*.

Além da decomposição apresentada acima, o módulo organiza as sentenças obedecendo a forma disjuntiva normal, ou seja, dando precedência para o operador AND. Como exemplo, uma *queryContext* composta por quatro atributos, envolvendo os atributos 'speed', 'humidity', 'temperature' e 'people-amount', formando uma consulta com um operador disjuntivo e dois conjuntivos 'speed<30;humidity>70|temperature>30;people-amount>20', que representa uma consulta para ônibus com velocidade inferior a 30 e umidade maior que 70 ou temperatura maior que 30 e quantidade de passageiros maior que 20, é processada em dois blocos de operações conjuntivas. Ou seja, 'speed<30;humidtiy>70' e outra 'temperature>30;people-amount>20', que repetindo o processo de decomposição para cada sequência conjuntiva, gera quatro subconsultas com apenas um único atributo. Esta estratégia aproveita a gramática da linguagem de consulta original do OCB.

4.2.5. Definição de Planos de Consultas

O componente CQE necessita acessar os dados que se encontram espalhados por diferentes provedores. Para isto o módulo *Definição de Planos de Consultas* utiliza-se de um recurso semelhante ao procedimento de Sistemas Gerenciadores de Bancos de Dados Distribuídos (SGBDDs), chamado de localização de dados distribuídos. Em SGBDDs, este procedimento utiliza esquemas que armazenam os endereços de fontes de dados distribuídas para descobrir a localização de atributos consultados. Aqui são utilizadas as informações do esquema de entidade DCDS.

Este módulo produz uma estrutura que vincula os endereços a serem consultados com a respectiva subconsulta. Um segundo processamento realiza a reformulação das subconsultas, como um tipo simplificado de otimização, no intuito de que se realize o mínimo de requisições. Desta forma, as subconsultas que possuem os endereços de destino repetidos

são agrupadas em uma única subconsulta, desde que façam parte de um mesmo bloco de operações conjuntivas.

Conforme os requisitos desejáveis para plataformas de *middleware* (seção 1.1), deve ser considerado cenários e arquiteturas em que os dados de entidades de contexto estejam armazenados de forma particionada em diferentes servidores. Em virtude disto, o módulo *Definição de Plano de Consultas* precisa considerar a existência de diferentes tipos de particionamento nos quais os dados estejam organizados, para então elaborar o plano de consultas.

4.2.5.1. Estratégia de Particionamento de Dados

A adoção de estratégias de armazenamento de dados de forma particionada tem o potencial de proporcionar melhorias como: aumento do desempenho, da escalabilidade e facilidade de gerenciamento. A partir das informações contidas no esquema DCDS, a plataforma *Sirius* provê duas estratégias: *particionamento por entidade* e *particionamento por atributo*.

4.2.5.1.1. Particionamento por Entidade

No particionamento de dados por entidade, múltiplos provedores de dados independentes podem armazenar e gerenciar subconjuntos de instâncias de entidades relacionadas a um mesmo esquema. Nesta estratégia de particionamento de dados, ao invés de armazenar o conjunto completo de instâncias da entidade de contexto em um único provedor, subconjuntos dessa entidade são particionadas em múltiplos provedores com base em diferentes políticas de cunho organizacional, financeiro ou geográfico. Uma característica desta estratégia é que, para cada instância de entidade de contexto, todos os atributos relacionados a ela estão armazenados de forma completa em um único provedor, embora não exista a obrigatoriedade de atribuir valores para todos os atributos.

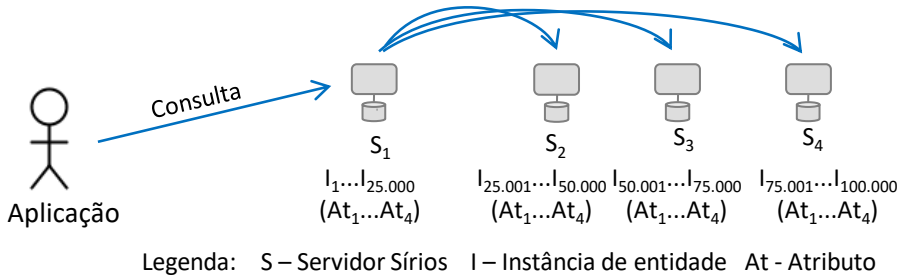
Tendo como referência as estratégias de particionamento em SGBDs relacionais, é possível identificar uma relação de semelhança entre o particionamento horizontal e o particionamento de dados por entidade. Como já conceituado no Capítulo 2, o particionamento horizontal (AGRAWAL; NARASAYYA; YANG, 2004) define que as linhas de uma tabela (aqui, cada linha de uma tabela de um SGBD relacional está relacionado a uma instância de entidade de contexto completa, com todos os atributos) são fisicamente divididas em duas ou mais sub-tabelas. Durante o decorrer deste trabalho, ao invés de invocar a

nomenclatura utilizada por SGBDs relacionais, particionamento horizontal, será referenciado por particionamento de dados por entidade.

Por exemplo, em uma grande cidade, é comum existir diferentes empresas de ônibus. Em cenários como este, é de se esperar que cada empresa deseje administrar diretamente o conjunto de ônibus pertencentes à sua frota. Desta forma, a estratégia de particionamento de dados por entidade garante que cada empresa armazene e gerencie os dados referentes aos ônibus de sua responsabilidade.

A Figura 12 apresenta um exemplo de particionamento de dados por entidade, no qual quatro servidores *Sirius* (S_1, S_2, S_3, S_4) gerenciam dados de contexto. Cada servidor *Sirius* possui uma representação completa dos atributos (At_1, At_2, At_3, At_4) de um conjunto de instâncias de entidades (25.000 instâncias por provedor, de um universo de 100.000 instâncias, representadas por I_1 até $I_{100.000}$). Assim, cada provedor de dados independente é associado a um servidor *Sirius*, que, auxiliado pelos mecanismos de localização e integração de dados que utilizam informações da especificação DCDS, proveem uma visão unificada das entidades de contexto.

Figura 12 : Exemplo de Particionamento de dados por Entidade



O processamento de uma consulta recebida por um servidor *Sirius* resulta na criação de subconsultas que são encaminhadas para outros três servidores da plataforma e para componentes locais para poder, ao receber todas as respectivas respostas, entregar uma única resposta à aplicação solicitante. Neste exemplo, as subconsultas encaminhadas para cada servidor *Sirius* são idênticas, uma vez que todas as instâncias possuem os mesmos atributos armazenados. Através desta operação, de consultar quatro servidores *Sirius* simultaneamente, a plataforma está provendo acesso concorrente a cada servidor, dividindo a carga de processamento da consulta em cada servidor.

Levando em consideração as peculiaridades do cenário de exemplo, no qual todos os servidores possuem a mesma quantidade de instâncias de entidades de contexto armazenadas, como também se pressupõe que cada servidor possui configuração idêntica aos demais, tais

como capacidade processamento e banda de tráfego de rede, é esperado que a adoção desta estratégia de processamento distribuído resulte na diminuição do tempo de resposta. É salutar ressaltar que este cenário é hipotético e nem sempre encontrado em sistemas reais, sendo utilizado apenas como exemplo.

Assumindo como exemplo o transporte público, esta estratégia de particionamento de dados por entidade ainda permite a expansão do conjunto de servidores *Sirius*, de acordo com o surgimento de novos bairros e a expansão da malha de ônibus, ou a participação de novas empresas. A adoção de particionamento de dados por entidade permite a escalabilidade dos dados, sem que a aplicação consumidora tome ciência destas mudanças. Desta forma, uma consulta não é alterada em decorrência da quantidade de servidores *Sirius* utilizados na infraestrutura de armazenamento e gerenciamento dos dados.

A estratégia de particionamento de dados por entidade, além de permitir o aumento de escala do sistema, facilita o gerenciamento das instâncias de entidade de contexto armazenadas em cada servidor *Sirius*. Com menos dados em cada servidor, o consumo de recursos de processamento, memória e comunicação é menor.

4.2.5.1.2. Particionamento por Atributo

No particionamento de dados por atributo, múltiplos provedores de dados independentes podem armazenar e gerenciar subconjuntos de atributos relacionados a uma mesma instância de entidade. Ou seja, ao invés de armazenar todos os atributos de uma determinada instância de entidade em um único provedor, subconjuntos de atributos são particionados em múltiplos provedores. Da mesma forma que no particionamento de dados por entidade, não há a exigência de se atribuir valores a todos os atributos.

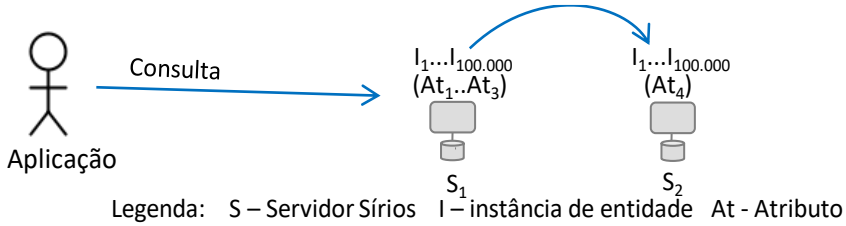
Recorrendo às estratégias de particionamento em SGBDs relacionais, o particionamento por atributo aqui apresentado possui grande semelhança ao particionamento vertical (AGRAWAL; NARASAYYA; YANG, 2004). Como fora conceituado no Capítulo 2, o particionamento vertical define que uma tabela T pode ter subconjunto de suas colunas fisicamente divididas, formando duas ou mais sub-tabelas (aqui, cada coluna de uma tabela de um SGBD relacional está relacionada a um atributo de todas as instâncias de entidades). Ao longo deste trabalho, ao invés de utilizar a nomenclatura adotada por SGBDs relacionais, particionamento vertical, será referenciado por particionamento de dados por atributo.

O particionamento de dados por atributo permite que empresas especializadas fiquem responsáveis por armazenar, gerenciar e processar aqueles atributos em que possuem expertise. Desta forma, retornando ao exemplo anterior, em cidades em que todo o

contingente de ônibus é controlado pelo poder municipal, tipos de dados relevantes e que necessitam de processamento especial podem ficar na responsabilidade de empresas de determinado nicho de especialidade. Como por exemplo, serviços de monitoramento em tempo real, mensuração da qualidade do ar ou cálculo da quantidade de passageiros ficam em provedores de empresas especializadas, enquanto que outras informações como o número da linha ou a identificação do motorista permanecem nos servidores governamentais.

A Figura 13 apresenta um exemplo de dois servidores *Sirius* (S_1 e S_2) com particionamento de dados por atributo. O servidor S_1 armazena três atributos (At_1 ,At_2 e At_3) e o servidor S_2 armazena um único atributo (At_4). Observe que todas as 100.000 instâncias de entidade (I_1 até $I_{100.000}$) são armazenadas tanto em S_1 quanto em S_2, porém apenas os atributos sob as respectivas responsabilidades encontram-se armazenados em cada servidor. Considere que uma aplicação externa realiza uma consulta relacionando os atributos At_1 e At_4, direcionada para o servidor S_1. Este servidor (S_1) processa a consulta recebida, gera duas subconsultas diferentes e as encaminha. O primeiro destino é o próprio Servidor S_1, que recebe uma subconsulta com o atributo At_1, enquanto que o outro destino é o servidor S_2, que recebe uma subconsulta apenas com o atributo At_4. Por último ocorre a integração da duas respostas no servidor S_1.

Figura 13 : Exemplo de particionamento de dados por atributo



A estratégia de particionamento de dados por atributo permite uma segmentação dos atributos de acordo com especificidades próprias, a partir de critérios organizacionais, financeiros ou geográficos.

4.2.5.2. Gerenciamento de Consultas

O módulo de *Gerenciamento de Consultas* tem a função de encaminhar o conjunto de subconsultas para os servidores *Sirius*, considerando o Plano de Consultas estruturado anteriormente. Como parte integrante do gerenciamento das consultas, os resultados das subconsultas devem ser integrados de forma que represente coerentemente a consulta com as restrições definidas.

Além das subconsultas para servidores *Sirius* remotos, este módulo encaminha subconsultas para o componente DPA local do próprio servidor *Sirius*, mas apenas quando a mesma é responsável por armazenar atributos presentes na consulta.

Todas as subconsultas realizadas pelo CQE, tanto as internas quanto as remotas, obedecem a sintaxe da linguagem de consulta do *Sirius*. Assim, as diferentes linguagens de consultas dos provedores de dados independentes são transparentes para o CQE. Desta forma, não apenas a aplicação cliente desconhece quais as diferentes linguagens adotadas pelos provedores que estão sendo acessados, como também o módulo gerenciador de consultas.

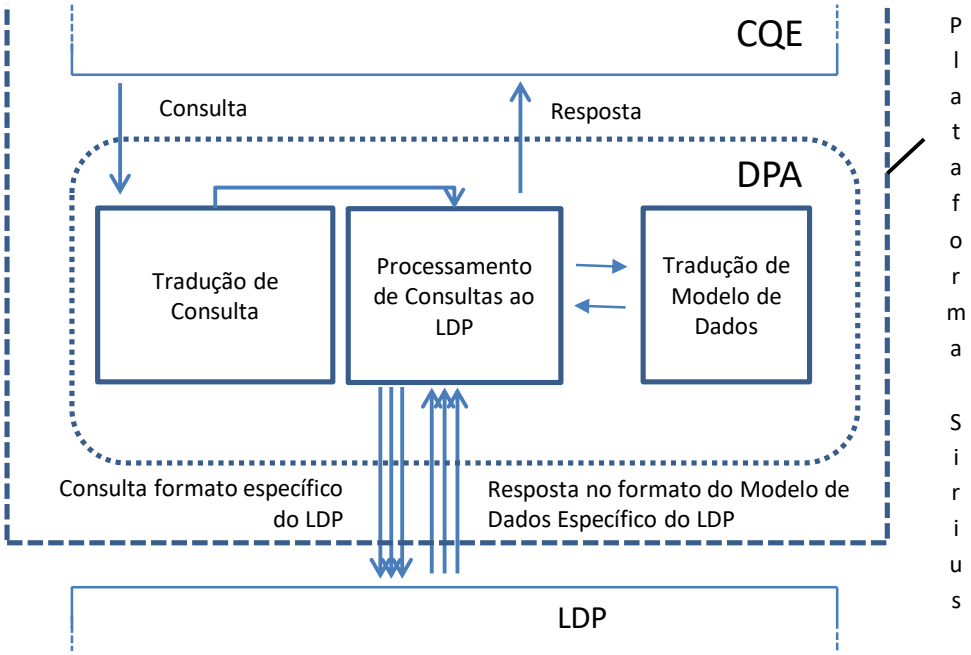
Por último, o gerenciador de consultas precisa realizar a integração das subrespostas recebidas de cada servidor *Sirius*. Dois processos principais de integração devem ser adotados, de acordo com a estratégia de particionamento utilizada pela entidade de contexto associada à consulta. Para as entidades de contexto organizadas com particionamento por entidade, a operação de junção (*join*) das subrespostas deve ser adotada, enquanto para as organizadas com particionamento por atributo, a operação de fusão (*merge*) das sobrepostas deve ser aplicada.

4.3. DPA

O componente DPA (*Data Provider Adapter*) provê o mecanismo de interoperabilidade que permite à plataforma *Sirius* acessar, de forma transparente, diferentes tipos de provedores, com diferentes linguagens de consulta e modelos de dados. O componente DPA é um módulo interno que intermedia a comunicação com o provedor de dados local. Quando um servidor *Sirius* precisa acessar seus dados locais, realiza uma consulta ao DPA, para então receber a resposta correspondente. O DPA recebe uma consulta no formato da linguagem de consulta do *Sirius*, traduz para a linguagem de consulta do provedor de dados local, realiza a consulta ao provedor de dados local, traduz a resposta do modelo de dados do provedor para o modelo de dados unificado DCDS, e, por fim, devolve a resposta para o componente CQE.

A Figura 14 apresenta os módulos internos do DPA que realizam estas funcionalidades. Primeiro, ocorre a *Tradução de Consultas,* que traduz a consulta recebida do formato da CQL da plataforma *Sirius* para o formato do LDP. O processo de tradução exige o conhecimento detalhado das duas linguagens envolvidas, como sintaxe, tipos de atributos e operadores. Portanto, o componente DPA é específico para cada tipo de LDP.

Figura 14 : Módulos internos do componente DPA



O módulo de *Processamento de Consultas*, entre outras funções, gerencia as requisições de consultas ao LDP e entrega o resultado para o CQE no modelo de dados DCDS. Alguns provedores de dados utilizam mecanismos de paginação com uma quantidade máxima de entidades por resposta. Nestes casos, em que são necessárias várias consultas até que se obtenha o universo de entidades que satisfazem a consulta realizada, além de gerenciar as requisições é preciso realizar a integração destas respostas parciais. O processo de integração de resultados parciais para estes casos é realizado pelo próprio módulo *Processador de Consultas*, adotando a estratégia de traduzir cada resposta parcial, para em seguida realizar o procedimento de integração. Desta forma os desenvolvedores de DPAs não precisam entender o mecanismo de integração de todos os modelos de dados utilizados, apenas do modelo de dados DCDS. Após a tradução dos fragmentos manipulados, e da integração, este módulo retorna para o CQE o resultado de seu processamento no modelo de dados DCDS.

O módulo de *Tradução de Modelo de Dados* é responsável por traduzir o modelo de dados do LDP para o modelo de dados DCDS. Os procedimentos realizados neste componente exigem conhecimento detalhado dos dois modelos em questão. Ele é chamado pelo Processamento de Consulta e devolve para este mesmo módulo o conjunto de dados traduzido.

O componente DPA provê transparência de representação para o acesso a diferentes tipos de provedores de dados, desde provedores que utilizam SBGDs relacionais, ou do tipo

não relacional. Graças a este módulo, a plataforma *Sirius* é definida como independente de provedor de dados.

4.4. Considerações Finais

Neste capítulo foi apresentada a abordagem para processamento de consultas distribuídas e particionadas para plataformas de *middleware*. Esta abordagem permite que plataformas tenham capacidade de acessar dados inter-relacionados, localizados em provedores de dados independentes e distribuídos e tem como objetivo auxiliar no desenvolvimento de serviços, aplicações e plataformas no contexto das cidades inteligentes.

A abordagem em questão é parte da plataforma de *middleware Sirius*, e tem suas funcionalidades expressas em dois componentes: o CQE e o DPA. No capítulo foram apresentadas as interações existentes entre os componentes durante o processamento de consultas distribuídas. Também foi explicado como as informações contidas no esquema DCDS são utilizadas durante este processamento.

O CQE proporciona transparência de localização dos servidores *Sirius* distribuídos. Este componente possui um módulo de Análise Léxica, Análise Sintática e Validação que certifica a sintaxe da *queryContext* recebida e a existência em um esquema DCDS dos atributos consultados. Em seguida cria novas subconsultas e as organiza em um plano de consultas, levando em consideração a estrutura como os atributos estão distribuídos pelos servidores *Sirius*. Os atributos podem estar particionados de duas formas diferentes: particionamento por entidade e por atributo. A utilização de particionamento permite que a plataforma acesse cada servidor de forma concorrente. As últimas etapas ocorridas neste componente acontecem com o gerenciamento das requisições e a integração dos resultados recebidos.

Este capítulo também abordou a linguagem de consulta utilizada pela plataforma, uma adaptação da linguagem de consulta utilizada pelo *Orion Context Broker*. Foi apresentada a gramática formal EBNF da linguagem do *Sirius* para melhor compreensão da sintaxe.

O outro componente apresentado foi o DPA, que provê a interoperabilidade necessária para o acesso a diferentes tipos de provedores de dados. Este componente realiza a tradução de uma consulta na linguagem de consulta do *Sirius* para a linguagem específica do provedor de dados local. Este mesmo componente é o responsável por traduzir o resultado da consulta para o modelo de dados DCDS, que funciona coma cola, permitindo a interoperabilidade entre diferentes sistemas.

Nos dois componentes, o DCDS é uma peça chave. No CQE, o esquema DCDS é utilizado para a localização dos dados distribuídos e a validação da *queryContext*. No DPA, o modelo de dados unificado utilizado é o próprio DCDS. Com o trabalho conjunto destes dois componentes, os provedores de contexto são encapsulados e acessados através de uma mesma interface de acesso.

Analisando o processamento discutido de um modo geral, o CQE possibilita o gerenciamento de requisições para diversos servidores *Sirius* distribuídos, inclusive promove a criação de novas subconsultas considerando cada tipo de particionamento dos dados. Desta forma, este componente provê transparência de localização e de particionamento. O CQE faz com que o acesso a recursos locais e remotos ocorram com o uso de operações idênticas, o que caracteriza a transparência de acesso. Enquanto isto, a transparência em relação aos diferentes modelos de dados (interoperabilidade) é provida pelo DPA. Estes tipos de transparências providos pela plataforma *Sirius* são fundamentais para o desenvolvimento de serviços, sistemas e aplicações para Cidades Inteligentes, no qual atores de diferentes domínios de aplicações no contexto de cidades inteligentes podem publicar e compartilhar seus dados sem se ater aos detalhes de localização e formato dos dados.

5. Estudo de Caso

Neste capítulo é apresentado o estudo de caso realizado com o propósito de validar a abordagem desenvolvida, além de analisar o desempenho em diferentes cenários. Para isto, é apresentado um estudo de caso utilizando dados semirreais do sistema de transporte público da cidade de São Paulo, no qual foram utilizadas informações de 10.000 ônibus. Para a análise de desempenho, foram organizados vários cenários, com diferentes tipos de particionamento, assim como também diferentes quantidades de provedores de dados.

5.1. Experimentos

Para validar a abordagem de processamento de consultas distribuídas e particionadas, um protótipo da plataforma *Sirius* foi desenvolvido. Este protótipo é baseado em requisições HTTP (do inglês, *Hypertext Transfer Protocol*) (FIELDING, 1997) e serviços REST (do inglês, *Representational State Transfer*) (MASSE, 2011). Através de um conjunto de serviços RESTful, a implementação do protótipo provê consultas de forma integrada e interoperável entre provedores de dados distribuídos.

O protótipo foi desenvolvido com Flask (GRINBERG, 2018), um *framework* Python utilizado para desenvolver aplicações web e fornecer serviços RESTful. O protótipo adotou como provedor de dados local o *Orion Context Broker* (FIWARE, 2019b) para persistir as instâncias de entidades de contexto. Para simular o ambiente distribuído durante o desenvolvimento foram utilizados containers Docker[2] (DOCKER, 2019). Todas estas tecnologias também têm sido utilizadas por outros projetos de pesquisa em cidades inteligentes, a exemplo do provedor de dados históricos compatível com NGSI-v2 chamado de QuantumLeap (QUANTUMLEAP, 2019).

Nos experimentos realizados foram utilizados dados semirreais, partindo inicialmente da base de dados em tempo real do sistema de transporte urbano da cidade de São Paulo - Olho Vivo (SPTRANS, 2019). Este sistema provê uma API RESTful para acesso a dados relacionados a todos os ônibus em operação na cidade, onde são disponibilizados dados como coordenadas de localização e sentido da linha. Originalmente, foi capturada uma base de dados superior a 11.000 instâncias da entidade ônibus. Em seguida foram selecionadas as 10.000 primeiras instâncias. Este ajuste foi realizado para simplificar o processo de distribuir 10.000 instâncias da entidade de contexto ônibus entre quantidades de servidores diferentes (por exemplo, 10, 5 e 2 servidores), e assim ter quantidades equânimes em cada servidor,

[2] https://www.docker.com

representado na Tabela 1. Esta base de dados inicial provê informações relacionadas ao nome da linha, localização e o sentido da viagem. No entanto, para permitir que este experimento explore melhor as potencialidades da plataforma e para melhorar a explicação dos cenários, foram também adicionados atributos de contexto sintéticos. Desta forma, cada instância da entidade ônibus teve adicionado os seguintes atributos: total de passageiros, temperatura externa, velocidade, código motorista, temperatura interna, nível de gasolina e nível de óleo.

5.1.1. Cenários dos Experimentos

Vários cenários foram organizados para testar as funcionalidades da plataforma e também para a realização da avaliação de desempenho. A organização das instâncias de entidade por servidor está representada na Tabela 1. O ambiente de execução do experimento foi no Laboratório COMPOSE da UFPB, onde foram utilizados 12 microcomputadores com configurações idênticas. Sendo que, 10 microcomputadores foram utilizados para os servidores *Sirius*, um microcomputador para a instalação do OCB (*Orion Context Broker*) e um microcomputador com uma aplicação teste para realizar as requisições de consulta. Os computadores utilizados possuem as seguintes características:

Hardware: Dell OptiPlex 990 (Intel Core I5-2400 / 3.10 GHz, 16GB RAM 6MB Cache)

Software: Ubuntu Server 18.04 LTS 64-bit

Tabela 1 : Divisão de instâncias de entidades pelos servidores Sirius

Quantidade de servidores *Sirius*	Quantidade de instâncias de entidades por servidor
1 servidor	10.000
2 servidores	5.000
5 servidores	2.000
10 servidores	1.000

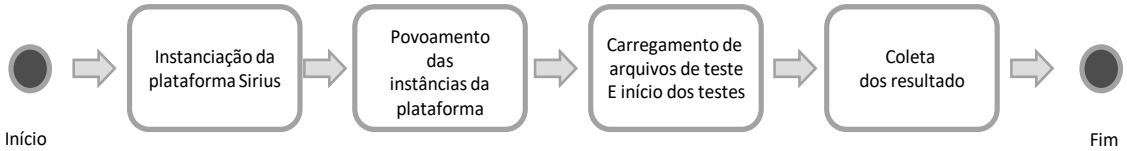
Para configuração do ambiente de testes, conforme citado anteriormente, foram utilizados dados obtidos do sistema de transporte público da cidade de São Paulo. Após a coleta desses dados foi realizada a tradução para o modelo de dados DCDS (LOPES; SILVA; ELIAS, 2019) e em seguida a adição dos atributos sintéticos. O resultado deste processamento serviu como base para uma série de cenários de teste, e, para cada cenário foram gerados os

dados de povoamento com 10.000 instâncias armazenadas nas diferentes configurações da plataforma, variando o número de provedores de dados e os tipos de particionamento.

Os diferentes cenários permitem observar o comportamento da plataforma para uma determinada configuração. Foram utilizadas as seguintes quantidades de servidores *Sirius*: 1, 2, 5 ou 10. A mudança de cada cenário durante a realização dos testes obrigou a realização de procedimentos de persistência do esquema de entidade e povoamento dos servidores *Sirius* (e seus respectivos provedores de dados), de acordo com a estrutura de particionamento e da quantidade de servidores utilizados. Portanto, cada cenário exige diferentes formas de distribuição das instâncias de entidades e dos atributos das entidades entre os servidores *Sirius*. Logo, para cada cenário existe um esquema de entidade específico.

Para testar diferentes configurações de cenários, foi necessário executar uma sequência de etapas resumidas na Figura 15. Para prover estas configurações, considerando um exemplo em que se utiliza 10 servidores *Sirius*, primeiro é executado nos 10 microcomputadores um *script* para iniciar os serviços relacionados a um servidor *Sirius*. Para cada microcomputador que funciona com um servidor *Sirius*, é preciso iniciar os seguintes serviços: a) Gunicorn (GUNICORN, 2019)- servidor web que oferece suporte ao ambiente de produção do *framework* Flask, permitindo o uso de múltiplos *workers* para disparar requisições de consulta (seguindo o manual e após testes para encontrar a melhor configuração, foram utilizados 9 *workers*); b) *Framework* Flask - provê os serviços da plataforma *Sirius*, incluindo todos os componentes internos; c) OCB (*Orion Context Broker*) - LDP, é o provedor de dado local, a quantidade de *workers* utilizada pelo OCB seguiu a recomendação da documentação, 4 *workers*).

Figura 15 : Fluxograma das etapas realizadas na organização de cada cenário



Para realizar o povoamento dos servidores *Sirius* foram utilizados componentes da plataforma *Sirius* que não fazem parte do escopo deste trabalho, a saber: o componente CIM e o componente CSM. Ambos foram brevemente descritos no Capítulo 3. Assim, foram utilizados protótipos de componentes desenvolvidos por pesquisadores do COMPOSE. Através destes componentes, para cada cenário, foi registrado o esquema da entidade

correspondente ao cenário avaliado, assim como a base de dados com as 10.000 instâncias de ônibus personalizadas. Ao final desta etapa de povoamento, os servidores *Sirius* de cada cenário continham ao total 10.000 instâncias de entidade, representando ônibus da cidade de São Paulo.

Continuando a descrição das etapas de configuração, do ponto de vista da aplicação que realiza os testes, é criado um arquivo de teste específico para cada cenário, que considera as peculiaridades de configuração, como tipo de particionamento e quantidade de servidores utilizados, fazendo referência a um esquema específico. As últimas etapas correspondem à execução dos testes e à coleta dos dados.

A seguir são detalhados oito cenários organizados para a avaliação de desempenho, conforme indicado na Tabela 2.

Tabela 2 : Cenários de avaliação de desempenho

Cenários de avaliação de desempenho
Cenário de referência
Cenário centralizado
Cenário Particionamento por entidade – 2 servidores *Sirius*
Cenário Particionamento por entidade – 5 servidores *Sirius*
Cenário Particionamento por entidade – 10 servidores *Sirius*
Cenário Particionamento por atributo – 2 servidores *Sirius*
Cenário Particionamento por atributo – 5 servidores *Sirius*
Cenário Particionamento por atributo – 10 servidores *Sirius*

5.1.1.1. Cenário de Referência

Como base de referência para as avaliações de desempenho foi escolhido o *Orion Context Broker*, que é o mesmo provedor de dados utilizado no protótipo da plataforma *Sirius*. Um dos fatores que contribuíram para a sua escolha é o fato do crescente número de projetos para cidades inteligentes que o tem adotado como componente na função de gerenciador de contexto. Então, na prática, foi confrontado um sistema de arquitetura centralizada e uma abordagem distribuída e particionada.

5.1.1.2. Cenário com Estrutura Centralizada

A plataforma *Sirius* permite a utilização de um único servidor *Sirius* para armazenar todos os dados de todas as instâncias de entidade referentes a um esquema específico (ou seja, todos os atributos e todas as instâncias de entidade em um único servidor *Sirius*). A esta

configuração, com a utilização de apenas um servidor *Sirius*, foi chamado cenário com estrutura centralizada. Desta forma são 10.000 instâncias de entidades armazenadas em um único servidor *Sirius*.

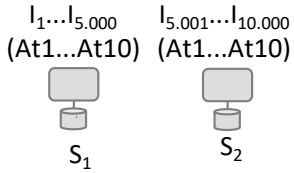
Os dois cenários apresentados até aqui (de referência e de estrutura centralizada) utilizam o OCB para armazenar as instâncias de entidade. A diferença entre os dois é que o segundo possui uma camada de software extra, correspondente à plataforma *Sirius*, enquanto que o primeiro é o OCB propriamente dito. O cenário de estrutura centralizada exigirá procedimentos a mais, como chamadas ao OCB, leitura do esquema de entidade e validação da *querystring* de consulta.

5.1.1.3. Cenários com Particionamento de Dados por Entidade

O objetivo deste conjunto de testes é observar o efeito no desempenho da plataforma para responder a uma consulta. Para esta observação, foi elevado gradativamente a escala de servidores *Sirius* organizados com estrutura de particionamento de dados por entidade.

Para o grupo de experimentos de análise do desempenho do particionamento por entidade foram organizados três cenários com 2, 5 e 10 servidores *Sirius*. A Figura 16 representa como as instâncias de entidades ficam distribuídas no cenário com dois servidores *Sirius*. Como pode ser observado, cada servidor *Sirius* (S_1 e S_2) gerencia 5.000 instâncias da entidade ônibus.

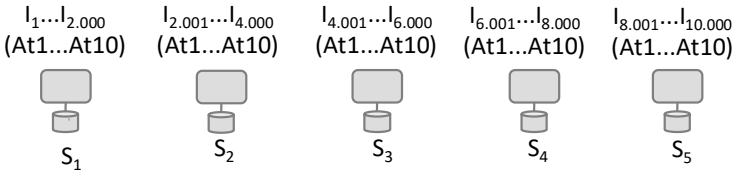
Figura 16 : Particionamento de Dados por Entidade com 2 servidores Sirius – 5.000 instâncias de entidade por servidor.



Legenda: S – Servidor Sirius I – instância de entidade At - Atributo

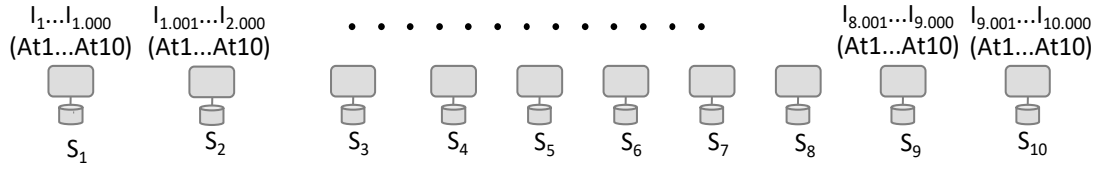
A Figura 17 apresenta o cenário com 5 servidores, cada servidor possui 2.000 instâncias de entidade armazenadas. Cada instância detêm todos os atributos definidos no esquema da entidade.

Figura 17 : Particionamento de Dados por Entidade com 5 servidores Sirius – 2.000 instâncias de entidade por servidor.



A última representação para o cenário com particionamento de dados por entidade está na Figura 18, que apresenta 10 servidores *Sirius*, cada um com 1.000 instâncias.

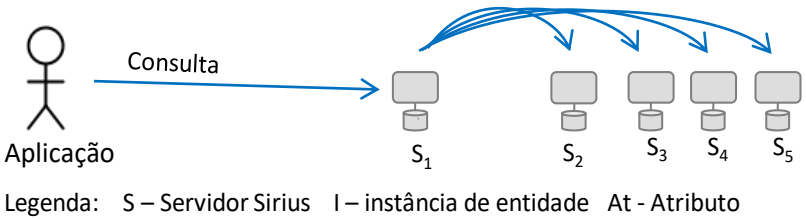
Figura 18 : Particionamento de Dados por Entidade com 10 servidores Sirius – 1.000 instâncias de entidade por servidor.



Em qualquer destes cenários ilustrados, pelo fato de estarem organizados seguindo a estratégia de particionamento de dados por entidade, o processamento de uma consulta exige que todos os servidores *Sirius* que estão registrados no esquema DCDS correspondente sejam examinados para verificar a presença de instâncias de entidade que atendem à consulta. Ou seja, a simples leitura do esquema e da lista de servidores não permite identificar em qual dos servidores está a informação consultada. A Figura 19, representa o fluxo de uma consulta envolvendo cinco servidores *Sirius*, seguindo o particionamento de dados por entidade. Observe que o primeiro servidor precisa realizar quatro consultas para os outros servidores *Sirius* antes de devolver o resultado.

A divisão da quantidade de instâncias de entidade por servidor observada nas representações das Figuras 16, 17 e 18 obedeceu a uma lógica exclusivamente de equidade. Porém é sabido que em cenários reais esta divisão ocorre de forma, muitas vezes, desbalanceada e incerta, considerando aspectos como custo, segurança, entre outros. Em virtude desta escolha observa-se que, quanto maior o número de servidores *Sirius* de um cenário, menor é a quantidade de instâncias de entidades armazenadas em cada servidor.

Figura 19 : Particionamento de Dados por Entidade com 5 servidores Sirius - Exemplo requisições durante consulta ao At5, que encontra-se em todos os servidores.



A Figura 20 apresenta a representação em JSON de um esquema de entidade, em que 10 atributos são organizados através da estrutura de particionamento de dados por entidade, em 10 servidores *Sirius*. Como é possível observar, cada atributo do esquema referencia todos os servidores *Sirius* cadastrados no esquema. Desta forma, cada instância de entidade pode ter todos os seus atributos em um mesmo servidor *Sirius*. Esta é a representação JSON de uma estrutura de particionamento com 10 servidores *Sirius* por entidade, apresentado anteriormente na Figura 18.

Figura 20 : Representação JSON de um esquema de entidade que utiliza particionamento de dados por entidade

```
{
"id":"bus_entity_partition_10",
"version":"1.0",
"owner":"dcds.i1.br",
"providers":["dcds.i1.br", "dcds.i2.br", "dcds.i3.br", "dcds.i4.br",
"dcds.i5.br", "dcds.i6.br", "dcds.i7.br", "dcds.i8.br", "dcds.i9.br",
"dcds.i10.br"],
"at1":{
   "type":"point",
   "refs":[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
"at2":{
   "type":"double",
   "refs":[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
"at3":{
   "type":"integer",
   "refs":[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
...
"at10":{
   "type":"double",
   "refs":[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}
}
```

Outros dois fatores que impactam no processamento distribuído são a latência de comunicação e o processamento de subconsultas. A latência é influenciada por diversos fatores, incluindo congestionamento da rede, número de roteadores no caminho e taxa de transmissão dos enlaces. No cenário do experimento, os microcomputadores estão interconectados por um *switch* com portas de 1 (um) Gbps, em um ambiente controlado, o que deve resultar em um baixo impacto da comunicação da rede, mas, de toda forma, é uma variável que possui interferência no tempo de resposta. O segundo fator, representado pelo processamento de subconsultas, inclui a divisão da consulta em subconsultas e a integração de resultados parciais que o primeiro servidor *Sirius* precisa realizar quando possui uma estrutura

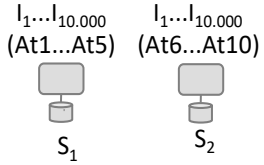
particionada. Desta forma, quanto maior a escala de servidores *Sirius*, mais custoso deve ser o processamento de integração, ao mesmo tempo que também há ganho no desempenho de cada servidor, em virtude da diminuição da quantidade de instâncias de entidades locais processadas.

5.1.1.4. Cenários com Particionamento de Dados por Atributo

O objetivo deste conjunto de testes é analisar o cenário com servidores *Sirius* organizados com estrutura de particionamento de dados por atributo.

Para este grupo de experimentos foram organizados três cenários com particionamento de dados por atributo com as seguintes quantidades de servidores *Sirius*: 2, 5 e 10 servidores. Porém, a quantidade de atributos armazenada em cada servidor *Sirius* obedece a uma lógica diferente da utilizada pelos cenários com particionamento de dados por entidade. Para o cenário com dois servidores *Sirius* foi dividido igualmente a quantidade de atributos por cada servidor, conforme ilustrado na Figura 21. Considerando que a entidade possui 10 atributos, cada servidor *Sirius* (S1 e S2) ficou com 5 atributos de todas as instâncias de entidade. No caso do cenário com cinco servidores *Sirius*, mostrado na Figura 22, cada servidor recebeu dois atributos. E, conforme ilustrado no cenário com 10 servidores, que pode ser observado na Figura 23, cada servidor recebeu um único atributo. Esta divisão pode variar a critério do administrador do sistema, porém, com estes exemplos é possível ter uma ideia da diferença existente entre as formas de particionamento de dados no que diz respeito a forma de organização dos atributos.

Figura 21 : Particionamento de Dados por Atributo com 2 servidores Sirius. 5 atributos por servidor.



Legenda: S – Servidor Sirius I – instância de entidade At - Atributo

Diferente dos cenários do particionamento de dados por entidade, no particionamento de dados por atributo o processamento de uma consulta é destinado apenas para aqueles servidores listados no esquema DCDS do atributo consultado. Desta forma, com o cenário ilustrado na Figura 24, uma consulta sobre o atributo At_5 faz com que o primeiro servidor (S_1) que recebe a consulta da aplicação externa realize apenas uma consulta ao servidor detentor do atributo (servidor S_3), e, logo em seguida, entregue o resultado. Uma representação em

JSON de um esquema de entidade para esta distribuição de atributos por servidor *Sirius* é apresentada na Figura 25. Neste esquema DCDS, uma entidade possui atributos distribuídos por 10 servidores *Sirius*, sendo que cada atributo possui referência para apenas um servidor *Sirius*.

Figura 22 : Particionamento de Dados por Atributo com 5 servidores Sirius. 2 atributos por servidor.

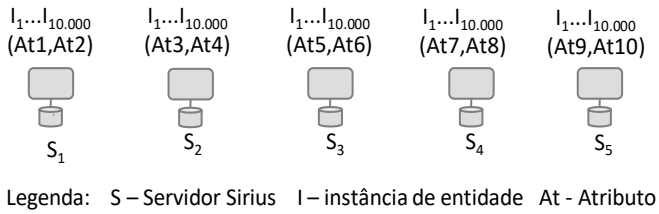


Figura 23 : Particionamento de Dados por Atributo com 10 servidores Sirius. 1 atributo por servidor.

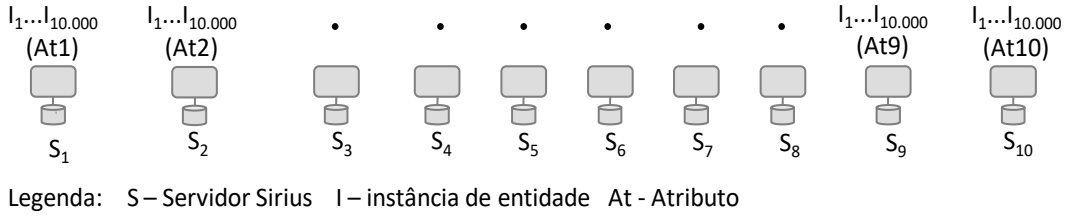
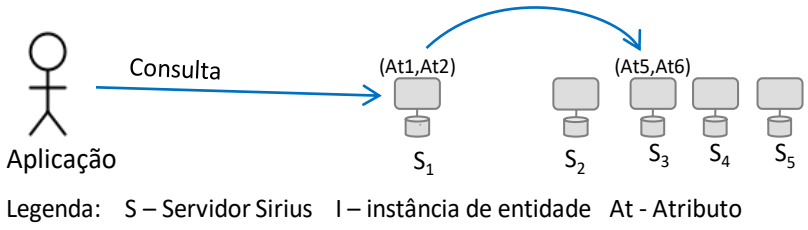


Figura 24 : Particionamento por Atributo com 5 servidores Sirius - Exemplo de requisições durante consulta ao At5, que encontra-se no servidor S5.



A organização dos dados entre os vários servidores *Sirius* também difere entre os particionamento de dados por entidade e por atributo. No particionamento de dados por atributo, o aumento da quantidade de servidores *Sirius* não altera a quantidade de instâncias de entidade por servidor. Como pode ser visto nas três representações de particionamento de dados por atributo (Figura 21, Figura 22 e Figura 23), mesmo com diferentes quantidades de Servidores a mesma quantidade de instâncias de entidade são gerenciadas por cada servidor *Sirius*, ou seja, existem 10.000 instâncias de entidade em cada um dos servidores. No particionamento de dados por atributo, a diferença entre uma estrutura com maior quantidade

de servidores em relação a outra estrutura com menor quantidade de servidores é a quantidade de atributos que cada servidor armazena.

Em um cenário com a distribuição de atributos entre servidores obedecendo a forma exposta, ou seja, quanto mais servidores, menor a quantidade de atributos por servidor, a redução da quantidade de atributos deve ter um impacto sobre o tempo de consulta. Assim, é esperada uma melhora na vazão ao se aumentar a quantidade de servidores *Sirius*, no caso da estrutura de particionamento de dados por atributo.

Figura 25 : Representação JSON de um esquema de entidade que utiliza particionamento por atributo

```
{
"id":"bus_attribute_partition_10",
"version":"1.0",
"owner":"dcds.i1.br",
"providers":["dcds.i1.br", "dcds.i2.br", "dcds.i3.br",
"dcds.i4.br", "dcds.i5.br", "dcds.i6.br", "dcds.i7.br",
"dcds.i8.br", "dcds.i9.br", "dcds.i10.br"],
"at1":{
   "type":"point",
   "refs":[1]},
"at2":{
   "type":"double",
   "refs":[2]},
"at3":{
   "type":"integer",
   "refs":[3]},
"at4":{
   "type":"double",
   "refs":[4]},
"at5":{
   "type":"integer",
   "refs":[5]},
...
"at10":{
   "type": "double",
   "refs":[10]}
}
```

5.1.2. Plano de Testes

Para cada cenário de teste são realizados quatro tipos de consultas. Estas consultas exploram como a plataforma se comporta em relação a retornar uma quantidade de instâncias de entidades diferentes como resultado, assim como também permite que seja verificado diferenças de desempenho entre consultas geográficas e não geográficas. Para melhor compreensão do plano de testes, foram elaboradas a Tabela 3 e a Tabela 4. A Tabela 3 representa a identificação das consultas, o tipo das consultas (geográfica e não geográfica) e a quantidade de instâncias de entidades retornadas. Já a Tabela 4 identifica a sintaxe de cada consulta.

O objetivo da análise de desempenho aqui proposta é avaliar a capacidade máxima suportada pela plataforma *Sirius* (fazendo também uma comparação com o OCB) em relação ao processamento de consultas. Para tal, a vazão de mensagens (*throughput*) da plataforma é analisada. Estas análises adotam uma perspectiva externa, isto é, a partir de aplicações

externas que realizam consultas a um servidor *Sirius*. Desta forma, é possível identificar a carga máxima suportada pelo sistema. Para isso, foi medida a vazão na troca de mensagens nos oito cenários considerados neste capítulo, tanto explorando abordagem centralizada, como explorando abordagens distribuídas e particionadas.

Tabela 3 : Consultas – Quantidade de instâncias de entidades retornadas

#	Tipo	Quantidade de instâncias retornadas
1	Não geográfica	993
2	Não geográfica	9.068
3	Geográfica	999
4	Geográfica	9.056

A análise de desempenho é feita utilizando quatro consultas. Duas exploram características não geográficas (as consultas 1 e 2 da Tabela 3) e diferem entre si pela quantidade de instâncias de entidade presentes na resposta. Os dados referentes ao atributo destas consultas são dados sintéticos que foram gerados de forma aleatória, compondo a base de 10.000 instâncias de entidades. Cada consulta foi expressa conforme exposto na Tabela 4. As outras duas consultas exploram o recurso de consulta geográfica e diferem entre si pela quantidade de instâncias de entidades retornadas. Já os dados destas duas consultas geográficas foram adquiridos da plataforma Olho Vivo, representando a distribuição geográfica de ônibus na cidade de São Paulo no momento em que os dados foram capturados. A consulta #1, conforme Tabela 4, faz referência ao esquema ‘bus’ definido no próprio servidor *Sirius* que é consultado ‘sirius.s1.br’ (*ownerName*) e atua em relação a um único atributo ‘speed’ com o operador maior que ‘>’ e tem como valor ‘90’ (*value*), ou seja, espera-se como resultado todas as instâncias de entidade do esquema ‘bus’ (cujo proprietário do esquema é o ‘sirius.s1.br’) que possuam o atributo ‘speed’ com valor maior que 90. Esta consulta #1 retorna 993 instâncias de entidades, conforme a Tabela 3. A consulta #2 utiliza os mesmos parâmetros da consulta #1, mudando apenas o valor do atributo ‘speed’ para 10. Esta consulta #2 retorna 9.068 instâncias de entidade (Tabela 3).

As consultas #3 e #4 são consultas geográficas que retornam as instâncias de entidade que estão dentro de um raio a uma distância informada em metros de um ponto de referência. As duas consultas utilizam o mesmo ponto de referência, que são as coordenadas geográficas -23.59085762 e -46.61044925 (latitude e longitude). Estas coordenadas localizam-se na cidade de São Paulo, como pode ser visto através da ferramenta de mapas do Google[3], Google

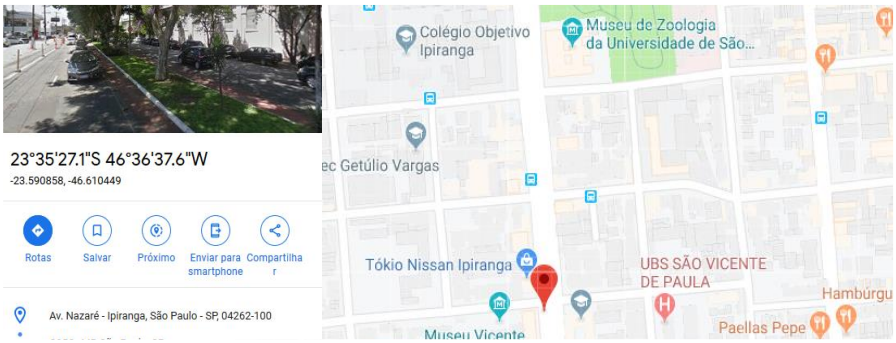
[3] https://www.google.com/maps/place/23%C2%B035'27.1%22S+46%C2%B036'37.6%22W/@-23.5902034,-46.6125812,16.75z/data=!4m5!3m4!1s0x0:0x0!8m2!3d-23.5908576!4d-46.6104492

Maps (GOOGLE, 2019), na Figura 26. A diferença entre as duas consultas é em relação ao raio utilizado (termo *maxDistance* da sintaxe apresentada na Tabela 4). A consulta #3 utiliza 5.400 metros como raio, enquanto que a consulta #4 utiliza 20.000 metros.

Tabela 4 : Consultas – Sintaxe utilizada para representar consultas do plano de teste

#	Tipo	Sintaxe da consulta
1	Não geográfica	sirius.s1.br/entities/sirius.s1.br/bus?speed>90
2	Não geográfica	sirius.s1.br/entities/sirius.s1.br/bus?speed>10
3	Geográfica	sirius.s1.br/entities/sirius.s1.br/bus?georel=near;maxDistance:5400&geometry=point&coords=-23.59085762,-46.61044925
4	Geográfica	sirius.s1.br/entities/sirius.s1.br/bus?georel=near;maxDistance:20000&geometry=point&coords=-23.59085762,-46.61044925

Figura 26 : Exibição no Google Maps de coordenada geográfica utilizada no plano de testes



Fonte: (GOOGLE, 2019).

Os experimentos foram realizados através de repetidas requisições de consultas, enviadas para a plataforma *Sirius*. Assim, foi analisado o comportamento da plataforma para diferentes taxas de requisição (requisição por segundo) enviadas para um servidor *Sirius*, por exemplo, para uma taxa de 10 requisições por segundo, a aplicação externa realizou uma média de 10 requisições por segundo durante 300 segundos. Para cada taxa enviada foi coletada como métrica a vazão média de resposta do servidor *Sirius* durante o envio de consultas neste período de 300 segundos, ou seja, a quantidade média de respostas recebidas por segundo em decorrência das consultas realizadas.

Foi utilizado um período de duração de 300 segundos para o envio das requisições de consultas para o servidor *Sirius* com a finalidade de identificar a vazão do sistema mais próxima do real e evitar contabilizar, de forma errônea, os pacotes da fila de espera. Antes do início de cada medição, as 10 primeiras consultas foram descartadas, a fim de que o sistema

atingisse um estado estável. Outra medida aplicada para evitar ruídos na aferição do experimento (como, por exemplo que uma requisição atrasada do teste anterior adulterasse o resultado) foi o de adotar um período de intervalo de 90 segundos entre todos os testes. Nestes experimentos foi utilizado a aplicação de testes JMeter (versão 5.1.1) (APACHE, 2019), para disparar e receber as respostas, assim como para gerar a tabela dos índices analisados.

A Tabela 5 apresenta um resumo dos fatores variáveis nos cenários dos testes. Entre estes fatores estão: tipos de consulta, tipos de particionamento de dados, número de servidores utilizados, plataformas e taxas de requisições. Sobre as taxas de requisições, estas dizem respeito a quantidade de consultas enviadas pela aplicação cliente (aplicação que realiza o teste) para a plataforma (*Sirius* ou OCB). Como é possível observar, as taxas de requisição empregadas não foram iguais para todos os cenários. A razão em não se repetir um único conjunto de taxas de requisição para todos os cenários ocorreu em virtude do comportamento da plataforma testada atingir pontos de saturação em diferentes níveis de taxa de requisição, dependendo do cenário. Como o objetivo destes testes foi identificar a carga máxima suportada pela plataforma, os testes foram encerrados um pouco depois do ponto de saturação.

Tabela 5 : Resumo dos fatores presentes nos cenários de teste

Fatores	Níveis
Tipo Consulta	Geográfica, Não Geográfica
Tipo Particionamento de dados	Entidade, Atributo
Número Servidores	1, 2, 5, 10
Tamanho Resposta	~1.000 (menor), ~ 9.000 (maior)
Plataforma	*Sirius*, OCB
Taxa de requisição	Com menor volume de resposta e particionamento de dados por entidade: 10, 20, 30, 40, 60, 60, 70, 80.
	Com menor volume de resposta e particionamento de dados por atributo: 1, 2, 5, 10, 15, 20, 25
	Com maior volume de resposta e particionamento de dados por entidade: 1, 2, 4, 6, 8, 10, 12, 14, 16.
	Com maior volume de resposta e particionamento de dados por atributo: 1, 2, 34, 15, 20, 80

A próxima Seção destina-se a apresentação e discussão dos resultados obtidos com as análises realizadas. Serão abordados apenas os resultados das duas consultas não geográficas,

uma vez que os resultados encontrados para as consultas geográficas foram muito semelhantes e pouco acrescentam na discussão. Os resultados e as discussões sobre as consultas geográficas encontram-se nos Apêndices A e B.

5.2. Resultados e Discussões

Esta seção apresenta a avaliação de desempenho da abordagem de consultas distribuídas e particionadas da Plataforma *Sirius* e do *Orion Context Broker* (OCB). Os resultados são apresentados seguindo as consultas não geográficas apresentadas no plano de testes da seção 5.1.2. Cada consulta foi executada nos oito cenários estabelecidos na seção 5.1.1. Os resultados de cada consulta são apresentados em dois grupos, um com o particionamento por entidade e outro com o particionamento por atributo. Para cada um destes grupos é apresentado o gráfico correspondente ao resultado da vazão dos cenários com diferentes quantidades de servidores *Sirius* (2, 5, 10), o cenário centralizado e o cenário de referência (que utiliza OCB).

A avaliação de desempenho tem por finalidade verificar se a abordagem proposta trouxe escalabilidade para a plataforma e procura identificar a carga máxima suportada pela plataforma *Sirius* e pelo OCB nos cenários analisados e com a distribuição e quantidade de instâncias de entidades persistidas.

5.3. Consulta não Geográfica com Menor Volume de Resposta

Esta seção apresenta a discussão dos resultados para a consulta não geográfica que retorna o menor volume de dados em sua resposta (993 instâncias de entidade). São analisados os cenários com particionamento de dados por entidade e com particionamento de dados por atributos. Para cada tipo de particionamento existem cinco cenários, compostos por 2, 5 e 10 servidores *Sirius*, o cenário com estrutura centralizada e o cenário de referência, cujas requisições são endereçadas ao OCB.

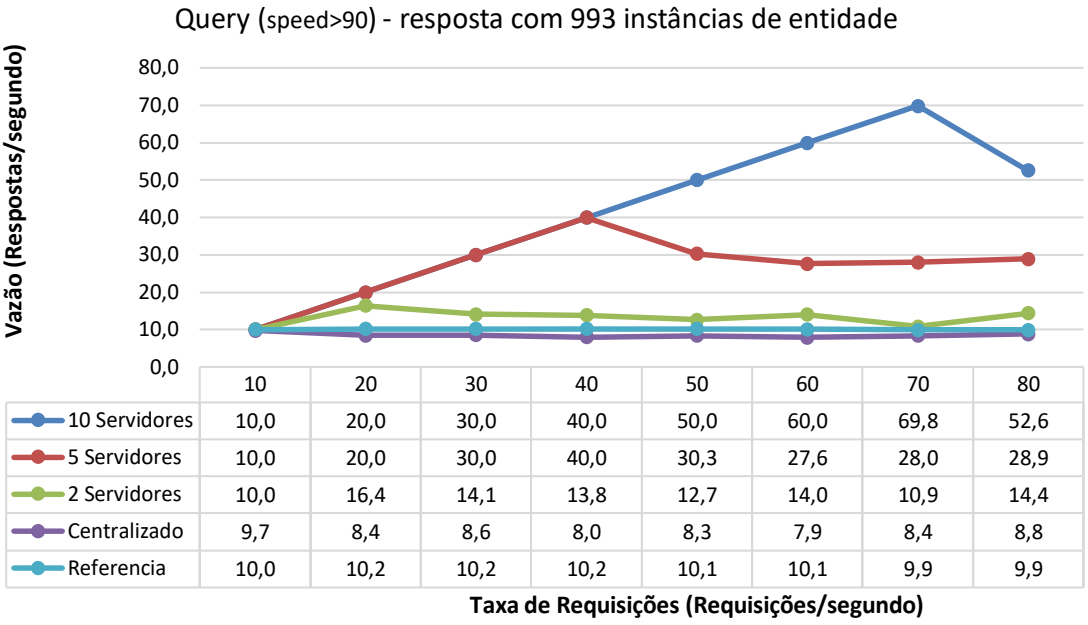
5.3.1. Particionamento de Dados por Entidade

A Figura 27 apresenta a avaliação de desempenho para uma consulta não geográfica no cenário com particionamento por entidade com menor volume de resposta. Esta consulta retorna 993 instâncias de um universo de 10.000 instâncias de entidade. Nesta figura é possível acompanhar a evolução da vazão da plataforma *Sirius* quando é aumentado a taxa de requisições por segundo (req/s). Foram analisados oito testes, com a taxa de requisição variando de forma crescente de 10 a 80 requisições por segundo, com intervalos de 10

requisições. O eixo X representa a taxa de requisição de dados à plataforma (req/s), enquanto que o eixo Y representa a vazão medida em respostas por segundo (resp/s). A Figura 27 representa o resultado da análise de cinco cenários, compostos por particionamento de dados por entidade para 2, 5 e 10 servidores, além do cenário com estrutura centralizada e o cenário de referência, cujas requisições são endereçadas ao OCB.

Seguindo a ordem crescente do desempenho de cada cenário presente na Figura 27, o cenário centralizado da plataforma *Sirius* foi o que obteve menor vazão ao longo dos registros. Neste cenário, o melhor resultado ocorreu quando a taxa de requisições foi de 10 req/s e se obteve uma vazão média de 9,7 resp/s. Após este ponto, se verifica a piora da vazão, chegando a 7,9 resp/s quando a taxa de requisições chega a 60 req/s. O próximo cenário da sequência foi o cenário de referência (utilizando o OCB), onde é possível verificar que a variação da vazão ao longo do gráfico é muito pequena, chegando a 10,2 resp/s para uma taxa de 20 req/s. Porém apresentou um desempenho superior ao cenário centralizado do *Sirius*. Esta performance superior do cenário de referência em relação ao cenário centralizado é justificado pelo motivo de que a plataforma *Sirius* utiliza o OCB como LDP, sendo normal que a plataforma *Sirius* com estrutura centralizada apresente desempenho um pouco abaixo do OCB devido ao tempo decorrido de processamentos excedentes, como por exemplo requisições ao OCB, consulta do esquema, validação da consulta, etc.

Figura 27 : Vazão - Particionamento por Entidade. Consulta não geográfica com menor volume de resposta



	10	20	30	40	50	60	70	80
10 Servidores	10,0	20,0	30,0	40,0	50,0	60,0	69,8	52,6
5 Servidores	10,0	20,0	30,0	40,0	30,3	27,6	28,0	28,9
2 Servidores	10,0	16,4	14,1	13,8	12,7	14,0	10,9	14,4
Centralizado	9,7	8,4	8,6	8,0	8,3	7,9	8,4	8,8
Referencia	10,0	10,2	10,2	10,2	10,1	10,1	9,9	9,9

O cenário de particionamento de dados por entidade com dois servidores *Sirius* ficou com o terceiro melhor resultado, obtendo assim um desempenho superior ao OCB que, no teste com 20 req/s chegou a ser 60,7% melhor que o cenário de referência. Neste cenário particionado, cada servidor *Sirius* possui 5.000 instâncias de entidade, que corresponde à metade das instâncias de entidade do servidor no cenário de referência. No entanto, o *Sirius* precisou realizar procedimentos a mais que o OCB, como criação de subconsultas e integração de dados. Este resultado demonstra como o volume total de dados presentes no servidor *Sirius* tem influência em relação ao tempo de processamento de uma consulta.

O cenário de particionamento por entidade com cinco servidores apresentou um ganho expressivo em relação aos três primeiros cenários, tendo o seu melhor desempenho com a taxa de requisições de 40 req/s para a plataforma, na qual apresentou igual valor de vazão (40 resp/s). Neste ponto, o desempenho em relação ao cenário de referência foi superior em 292,1% e em relação ao cenário centralizado o desempenho foi superior em 400%.

O melhor cenário testado com o particionamento de dados por entidade foi com 10 servidores *Sirius*. Este cenário conseguiu atingir a vazão de 69,8 respostas por segundo, e, neste ponto, superou o cenário de referência em 584,3%. A vazão apresentada por este cenário teve um crescimento linear até atingir 69,8 resp/s, quando apresentou acentuada perda de desempenho, porém, mesmo com esta perda de desempenho manteve-se superior à vazão registrada pelo cenário de referência.

A plataforma *Sirius*, no cenário com 10 servidores, apresenta como ponto de saturação 68,8 resp/s, ou seja, depois deste ponto, não consegue processar um volume maior de requisições. Após atingir o ponto de saturação, em virtude do processamento ser distribuído, tanto novas requisições da aplicação de teste como respostas dos servidores remotos passam a ser descartadas, pois a plataforma encontra-se saturada. Desta forma, o desempenho da plataforma apresenta uma diminuição acentuada de forma mais perceptível para os cenários mais distribuídos, como é observada na Figura 27 para os cenários com 10 e 5 servidores.

No gráfico da Figura 27 é possível perceber um padrão de comportamento na linha que representa a vazão do sistema. Primeiro, a vazão do sistema cresce à medida que a taxa de requisições aumenta, o que significa que a plataforma *Sirius* ainda não chegou ao seu processamento máximo. Segundo, a partir de certo ponto, o gráfico decresce rapidamente, representando que o processamento máximo da plataforma foi atingido. Por fim, o gráfico se mantém estável mesmo com o aumento do fluxo de requisições. Neste ponto a plataforma *Sirius* já encontra-se saturada e descarta as requisições em excesso. Embora exista uma fila de espera para novas requisições, experimentos empíricos realizados durante os testes mostraram

que a fila de espera foi preenchida por completo durante os 300 segundos de envio contínuo de requisições em cenários com saturação.

Os experimentos demonstraram que a plataforma *Sirius* provê escalabilidade para aplicações, serviços e sistemas, a partir da estratégia adotada. Entre os cenários analisados, quanto maior a quantidade de servidores utilizados, maior vazão. Nesta avaliação é levado em consideração a especificidade dos cenários, cuja distribuição dos dados entre os servidores foi homogênea e as operações atômicas. Embora o cenário centralizado tenha registrado os valores mais baixos de vazão, a plataforma no cenário distribuído teve expressivo ganho proporcional de vazão. O cenário com 10 servidores e o cenário centralizado tiveram um diferença de 730,9% (para uma taxa de 70 req/s), enquanto que o cenário 5 foi superior em 400% ao cenário centralizado (para uma taxa de 40 req/s).

5.3.2. Particionamento de Dados por Atributo

Para a mesma consulta adotada na seção anterior foram organizados outros cenários com a estratégia de particionamento de dados por atributo, cujos resultados estão representados na Figura 28. Vale ressaltar que, por ser o mesmo universo de dados e a mesma consulta, a quantidade de instâncias retornadas é idêntica aos resultados da seção 5.3.1. No particionamento de dados por atributo cada servidor *Sirius* é povoado por todas as instâncias de entidades (10.000), porém possui apenas um subconjunto de seus atributos. Enquanto que no particionamento de dados por entidade, a quantidade de instâncias de entidade é variável (entre 1.000 e 10.000 instâncias por servidor), porém cada instância de entidade possui a totalidade dos atributos.

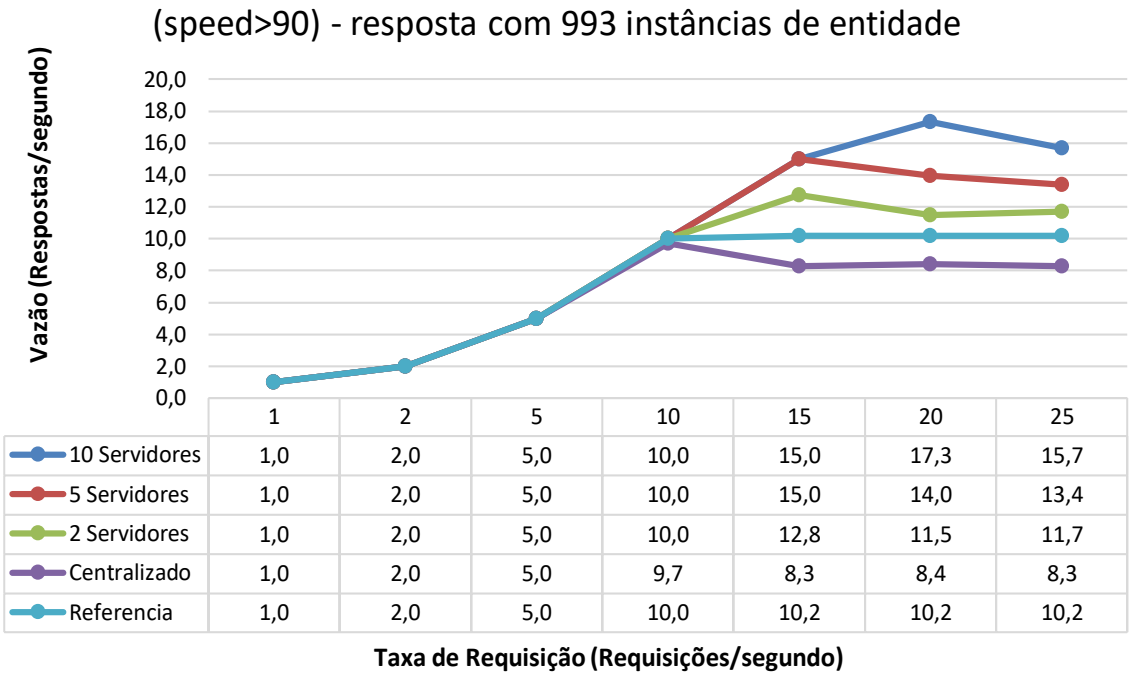
Na Figura 28 observa-se que o cenário que obteve a pior vazão foi o centralizado, registrando vazão de 9,7 resp/seg para uma taxa de 10 req/seg. Em seguida, o cenário de referência registrou vazão de 10,2 resp/seg para uma taxa de 15 req/seg. O cenário distribuído com dois servidores aparece com a terceira melhor vazão, alcançando 12,8 resq/seg quando a taxa foi de 15 req/seg. Em seguida, o cenário com cinco servidores registrou como melhor vazão 15 resp/seg para taxa de igual valor. Repetindo a mesma posição registrada no particionamento de dados por entidade, o cenário com 10 servidores atingiu a melhor vazão, chegando a 17,3 resp/seg para a taxa de 20 req/seg.

Nestes experimentos, o melhor resultado foi o cenário com 10 servidores *Sirius* (que possui um atributo por instância de entidade), que, para uma taxa de 20 req/seg apresentou vazão 69,6% superior ao cenário de referência, 23,5% superior ao cenário com 5 servidores (2 atributos por instância de entidade), 50,4 % superior ao cenário com 2 servidores (5 atributos

por instância de entidade) e 105,9% superior ao cenário centralizado (10 atributos por instância de entidade). Esta análise demonstra que, embora a totalidade dos servidores *Sirius* (considerando todos os cenários distribuídos e o cenário centralizado) possuam a mesma quantidade de instâncias de entidades (10.000), o fato de diminuir a granularidade de atributos por instância de entidade resulta em um desempenho melhor.

No gráfico da Figura 28 é possível observar que nenhum dos cenários chegou perto de alcançar a vazão dos cenários de particionamento de dados por entidade (Seção 5.3.1). O experimento mostrou que o servidor *Sirius* apresentou um desempenho melhor com particionamento de dados por entidade do que com o particionamento de dados por atributo, levando em conta exclusivamente o valor da vazão alcançada nestes cenários. Para os dois tipos de particionamento de dados utilizados nos experimentos, por entidade e por atributo, é possível observar a característica de escalabilidade, pois quanto maior a quantidade de servidores utilizada melhor é a vazão registrada.

Figura 28 : Vazão - Particionamento por Atributo. Consulta não geográfica com menor volume de resposta



	1	2	5	10	15	20	25
10 Servidores	1,0	2,0	5,0	10,0	15,0	17,3	15,7
5 Servidores	1,0	2,0	5,0	10,0	15,0	14,0	13,4
2 Servidores	1,0	2,0	5,0	10,0	12,8	11,5	11,7
Centralizado	1,0	2,0	5,0	9,7	8,3	8,4	8,3
Referencia	1,0	2,0	5,0	10,0	10,2	10,2	10,2

5.4. Consulta Não Geográfica com Maior Volume de Resposta

Esta seção apresenta a discussão dos resultados para a consulta não geográfica que retorna o maior volume de dados em sua resposta (9.068 instâncias de entidade). São analisados os cenários com particionamento de dados por entidade e por atributo. Para cada

tipo de particionamento existem três cenários, compostos por particionamento por entidade para 2, 5 e 10 servidores *Sirius*, além do cenário com estrutura centralizada e o cenário de referência, cujas requisições são endereçadas ao OCB.

5.4.1. Particionamento de Dados por Entidade

Os resultados desta seção representam a avaliação de desempenho de uma consulta não geográfica que retorna 9.068 entidades. Para avaliar o cenário de referência (avaliação da vazão do *Orion Context Broker*) foi desenvolvido uma ferramenta para realizar as consultas com as taxas de requisição desejadas, coletar dados e calcular a vazão do OCB. A necessidade do desenvolvimento de uma ferramenta específica para auxiliar na avaliação de desempenho ocorreu em virtude de que o OCB tem capacidade de retornar no máximo 1.000 instâncias de entidades como resposta em cada consulta e a consulta em questão corresponde a 9.068 instâncias de entidade, o que exige no mínimo a realização de dez consultas ao OCB. Desta forma, o cenário de referência teve a avaliação realizada com a ferramenta de teste específica, enquanto que para os demais cenários foi utilizada a ferramenta de teste Jmeter.

Uma breve observação sobre o mecanismo de gerenciamento de requisições para o OCB merece ser registrada, uma vez que a ferramenta de teste desenvolvida utiliza um mecanismo diferente da forma que um servidor *Sirius* acessa o OCB. Os dois mecanismos utilizados realizam requisições de consulta de forma assíncrona para o OCB. A diferença é que a ferramenta desenvolvida realiza a primeira consulta perguntando apenas quantas instâncias de entidade o OCB retorna para um conjunto de restrições, e, em decorrência desta quantidade e sabendo que o máximo em cada resposta é de 1.000 instâncias, são criadas as dez consultas para retornar 9.068 instâncias de entidade, que, ao final, são integradas em uma única resposta. Já um servidor *Sirius*, na primeira requisição de consulta solicita ao OCB a quantidade de instâncias existentes para aquela consulta, porém, nesta mesma requisição, solicita as primeiras 1.000 instâncias de entidades que satisfazem à consulta, e, em seguida, realiza 9 requisições para acessar as instâncias que faltam. São duas formas diferentes para a solução de um mesmo problema. A estratégia utilizada pelo servidor *Sirius* tem a probabilidade de ser mais eficiente, uma vez que exige do OCB o processamento de 9 consultas, enquanto que o primeiro mecanismo solicita o processamento de 10 consultas.

Estes dois mecanismos de gerenciamento de requisições a um mesmo LDP (*Local Data Provider*) evidenciam a importância de conhecer em detalhes cada LDP utilizado pela plataforma *Sirius* a fim de se obter o melhor desempenho. É possível que existam

mecanismos mais eficientes, no entanto, apenas análises mais detalhadas podem de fato identificá-los.

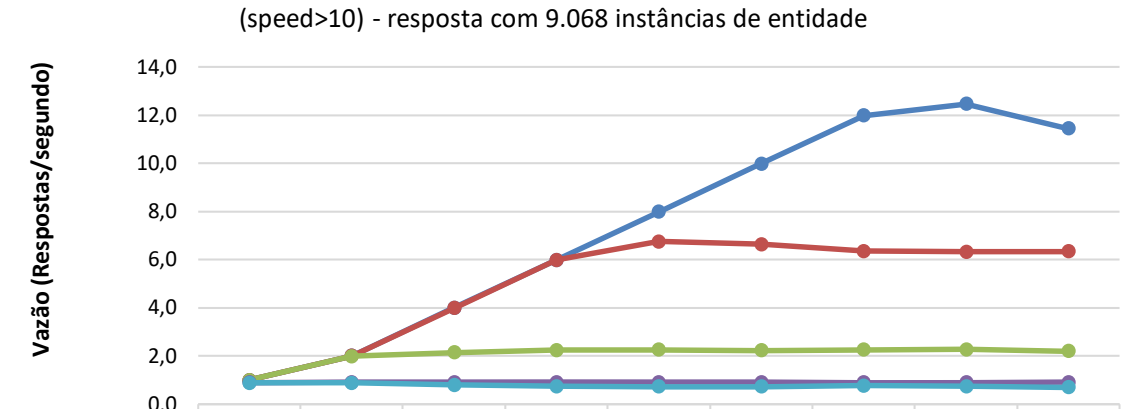
Após estas considerações, é apresentado o resultado da consulta não geográfica que retorna 9.068 instâncias de entidade. Na Figura 29 é possível acompanhar a evolução da vazão da plataforma *Sirius* quando a taxa de requisições é aumentada. Foram realizados testes com as seguintes taxas: 1, 2, 4, 6, 8, 10, 12, 14 e 16 requisições por segundo. Estas taxas diferem das taxas da análise empregadas na consulta não geográfica com menor quantidade de instâncias de entidade em virtude do desempenho da plataforma ser diferente. Na consulta não geográfica com maior volume de resposta, o ponto de saturação ocorre antes que na consulta com menor volume. E, a partir do ponto de saturação, não foi observado mudança significativa no comportamento da vazão, sendo este o motivo de não aparecer nas figuras seguintes. A figura representa o comportamento da plataforma para os 5 cenários: 3 configurações para o particionamento por entidade, com 2, 5 e 10 servidores, o cenário centralizado e o cenário de referência.

Seguindo a ordem crescente do desempenho dos cenários da Figura 29, o cenário que obteve a pior vazão foi o cenário de referência (OCB). Neste cenário, a melhor vazão foi com taxa de requisições igual a 2 req/s, obtendo vazão de 0,9 resp/s. Os outros registros de vazão para este cenário mantiveram-se estáveis em 0,7 resp/s. O próximo cenário na ordem crescente é o centralizado, apresentando vazão constante em 0,9 resp/s em todos os testes. Diferente dos resultados alcançados pela consulta não geográfica no cenário com particionamento por entidade com menor volume de respostas, aqui o cenário centralizado superou o cenário de referência em 28,5%. Esta diferença pode ser atribuída ao mecanismo de acesso ao LDP implementada pela plataforma *Sirius*, que realiza múltiplos acessos de forma diferente do mecanismo utilizado pela aplicação de teste do cenário de referência, conforme explicado no começo desta seção.

O cenário de particionamento de dados por entidade com dois servidores ficou com o terceiro melhor resultado. Para a taxa de 8 req/s este cenário alcançou vazão de 2,3 resp/s, o que significa uma superioridade de 228,5% em relação ao cenário de referência e de 155,5% em relação ao cenário centralizado. Já o cenário de particionamento por entidade com cinco servidores *Sirius* apresentou um ganho expressivo em relação aos três primeiros cenários, obtendo o seu melhor desempenho com a taxa de 8 req/s, quando atingiu uma vazão de 6,8 resp/s. Neste ponto, o desempenho em relação ao cenário de referência foi superior em 871,4% e em relação ao cenário com estrutura centralizada o desempenho foi superior em

655,6%. A vantagem do cenário com 5 servidores em relação ao cenário com 2 servidores foi de 195,6%.

Figura 29 :Vazão - Particionamento por Entidade. Consulta não geográfica com maior volume de resposta



	1	2	4	6	8	10	12	14	16
10 Servidores	1,0	2,0	4,0	6,0	8,0	10,0	12,0	12,5	11,4
5 Servidores	1,0	2,0	4,0	6,0	6,8	6,6	6,4	6,3	6,3
2 Servidores	1,0	2,0	2,1	2,2	2,3	2,2	2,3	2,3	2,2
Centralizado	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
Referencia	0,9	0,9	0,8	0,7	0,7	0,7	0,8	0,7	0,7

Taxa de Requisição (Requisições/segundo)

O melhor cenário analisado para a consulta não geográfica com particionamento por entidade com maior volume de resposta foi o cenário com 10 servidores *Sirius*. Este cenário alcançou a vazão de 12,5 respostas por segundo, e, neste ponto (com a taxa registrando 14 req/s), superou o cenário de referência em 1.685,7%. Em relação ao cenário centralizado, a diferença registrada foi de 1.288,8%.

Comparando os dois conjuntos de resultados (do particionamento de dados por entidade da consulta de menor volume - Seção 5.3.1 - e desta seção), os valores das linhas que representam a vazão dos cenários da consulta com maior volume de resposta (Figura 29) tendem a ser menores (menor vazão) que os valores obtidos nas consultas com menor volume de resposta. Considerando as especificidades deste experimento, como os atributos consultados e a distribuição de instâncias de entidades de forma uniforme entre servidores, os motivos que justificam este comportamento, para cada cenário de consulta com maior volume de resposta, são:

a) cada LDP processa maior volume de instâncias de entidade em sua resposta;

b) cada servidor *Sirius* realiza uma quantidade maior de requisições ao LDP;

c) maior volume de dados a ser trafegado pela rede, entre os servidores remotos, como também entre a aplicação e a plataforma *Sirius*;

d) maior volume de dados a ser integrado pela plataforma *Sirius*.

Embora em termos absolutos os cenários de consulta com menor volume de resposta (Seção 5.3.1) alcancem vazões mais expressivas, se for levado em consideração a diferença proporcional entre a vazão do melhor cenário e o cenário de referência, o cenário com maior volume de resposta apresenta vantagem. Enquanto na consulta com menor volume de dados e particionamento de dados por entidade a vantagem proporcional entre a vazão do cenário com 10 servidores (69,8 resp/s) e a vazão do cenário de referência (10,2 resp/s) é de 584,3%, no resultado da consulta que retorna maior volume de dados, considerando as vazões encontradas para o cenário com 10 servidores (12,5 resp/s) e do cenário de referência (0,7 resp/s), o ganho proporcional alcança 1.685,7%.

5.4.2. Particionamento de Dados por Atributo

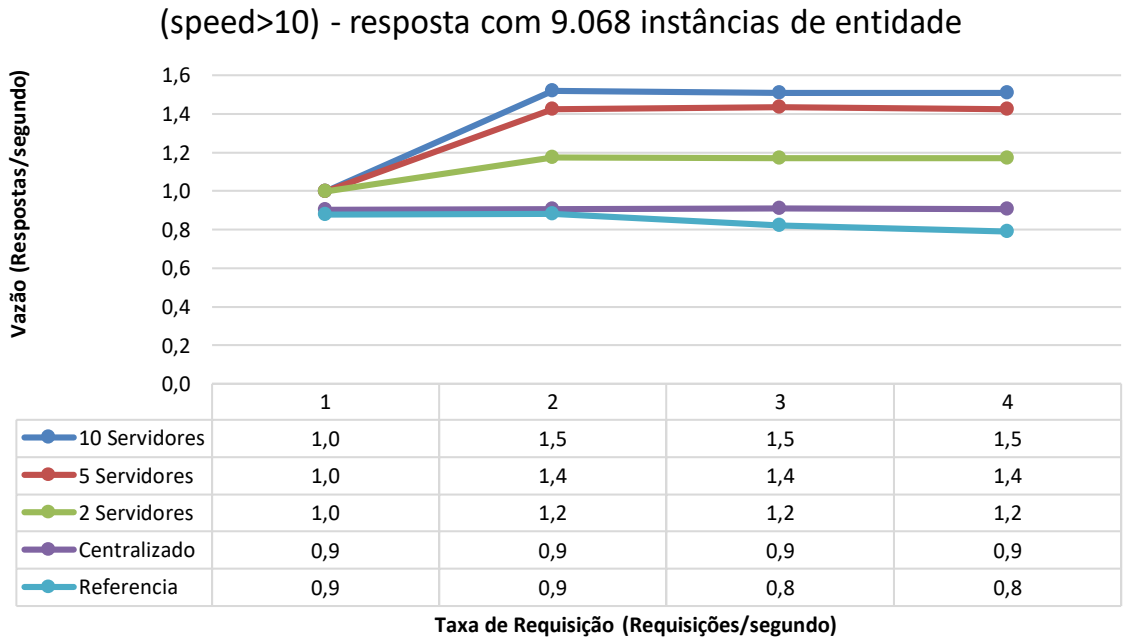
A Figura 30 apresenta os resultados dos cenários para a consulta não geográfica com maior volume de resposta em uma estrutura que utiliza particionamento de dados por atributo. O cenário de referência (OCB) ao registrar uma taxa de 3 req/s alcançou uma vazão de 0,8 resp/s, a melhor vazão deste cenário. O cenário centralizado apresentou desempenho ligeiramente melhor, mantendo-se estável durante os quatro testes realizados, com uma vazão de 0,9 resp/s. Descolado dos dois primeiros cenários, o cenário com dois servidores registrou 1,2 resp/s em sua vazão para as taxas de 2, 3 e 4 req/s. O cenário com 5 servidores apresenta um comportamento de destaque, uma vez que se distanciou do cenário anterior, e se aproximou bastante do cenário com 10 servidores, tendo registrado o melhor desempenho com uma vazão de 1,4 resp/s para as taxas de 2, 3 e 4 req/s. Ou seja, isto representa um desempenho superior em 75% ao cenário de referência e em relação ao cenário com estrutura centralizada o desempenho é de 55,5%, considerando a taxa de 3 req/s.

O melhor cenário analisado entre as estruturas de particionamento por atributo foi a que apresenta 10 servidores *Sirius*. Este cenário conseguiu atingir a vazão de 1,5 resp/s para as taxas de 2, 3, e 4 req/s. Considerando a taxa de 3 req/s, este cenário superou o cenário de referência em 87,5%, o cenário centralizado em 66,6%, o cenário com dois servidores em 25% e superou o cenário com 5 servidores em 7,1%.

Analisando os cenários com os dois tipos de particionamento para consultas não geográficas com maior volume de respostas, observa-se uma grande diferença entre as melhores vazões registradas. No cenário com particionamento por entidade foi alcançado 12,4 resp/s., enquanto que com particionamento por atributo a melhor vazão registrada foi de 1,5 resp/s. Apesar de registrar uma vazão tímida em comparação com o particionamento de dados

por entidade, a adoção de particionamento de dados por atributo pode ser uma restrição dos requisitos do sistema, onde diferentes variáveis podem nortear o emprego deste tipo de particionamento, como questões referentes à política da empresa ou limitações geográficas. O que o experimento demonstrou foi que o particionamento de dados por atributo proporcionou melhoria da vazão nos cenários com 2, 5 e 10 servidores.

Figura 30 : Vazão - Particionamento por Atributo. Consulta não geográfica com maior volume de resposta



	1	2	3	4
10 Servidores	1,0	1,5	1,5	1,5
5 Servidores	1,0	1,4	1,4	1,4
2 Servidores	1,0	1,2	1,2	1,2
Centralizado	0,9	0,9	0,9	0,9
Referencia	0,9	0,9	0,8	0,8

5.5. Considerações Finais

Este capítulo apresentou um estudo de caso para a validação da abordagem proposta. Para isto foram utilizadas diversas tecnologias que são comumente encontradas em outros projetos de plataformas para cidades inteligentes, como Python, Docker e *Orion Context Broker*. Para povoar o protótipo implementado foram utilizados dados semirreais de transportes públicos e foram organizados diferentes cenários para explorar diferentes características como tipo de particionamento, tipo de consulta e quantidade de servidores *Sirius*.

A análise dos resultados considerando os tipos de particionamento expôs algumas questões, tais como:

a) quanto maior o número de servidores na estrutura particionada, melhor a vazão. Ou seja, a plataforma *Sirius* permite configurações com diferentes quantidades de servidores, de

modo que o processamento e os dados sejam distribuídos, provendo assim uma solução escalável;

b) em relação aos dois tipos de particionamento (por entidade e por atributo), o particionamento por entidade mostrou alcançar melhor vazão;

c) os cenários particionados com 2, 5 e 10 servidores apresentaram desempenho superior ao cenário de referência e ao cenário centralizado.

Analisando o comportamento da vazão em relação aos dois tipos de consulta, um com maior volume de dados, e outro com menor volume de dados, foi possível observar que:

a) Nos cenários dos dois tipos de consulta pode-se observar um destaque no desempenho para os cenários com maior quantidade de servidores. Tanto o cenário com 5 servidores como com 10 servidores registraram as melhores vazões nos dois tipos de consulta;

c) A consulta com menor volume de resposta possui melhor desempenho para o cenário com 10 servidores, alcançando a vazão de 69,8 resp/s no particionamento por entidade e 17,3 resp/s no particionamento por atributo;

d) Por sua vez, a consulta com maior volume de dados alcançou uma vazão de 12,5 resp/s no particionamento por entidade e 1,5 resp/s no particionamento por atributo;

e) Em termos proporcionais, a consulta com menor volume de resposta e 10 servidores *Sirius* alcançou uma vantagem de 584,3% na comparação entre o cenário de referência no particionamento por entidade e 69,6% de vantagem no particionamento por atributo. Também analisando a vazão em termos proporcionais, a consulta com maior volume de resposta alcançou 1.685,7% de vantagem na comparação entre o cenário com 10 servidores e o cenário de referência no particionamento por entidade e 87,5% de vantagem no particionamento por atributo.

Como uma das limitações dos dados coletados do experimento, pode-se citar a limitada variedade de parâmetros coletados para análise. Outros parâmetros como memória, utilização da CPU e tempo de resposta poderiam contribuir para aprofundar os estudos realizados. No entanto, o tempo requerido para estudo, configuração e coleta de tais parâmetros foram fatores limitantes. Outra limitação do experimento diz respeito ao cenário de referência ser uma solução centralizada, e haver três cenários distribuídos. Embora citado entre os trabalhos relacionados plataformas distribuídas, o tempo exíguo do ambiente de testes e a pouca familiaridade com outras plataformas não permitiram configurar outro cenário de referência.

O estudo de caso pôde comprovar que a abordagem proposta alcançou o objetivo de acessar servidores distribuídos de forma transparente para a aplicação externa, através do

auxílio dos elementos do metamodelo DCDS, que minimiza trabalhos de configuração do sistema. Outro objetivo alcançado foi a viabilidade em realizar a integração de dados provenientes de diferentes servidores *Sirius* com a adoção de um modelo de dados unificado (DCDS), traduzindo um modelo de dados de um provedor de dados para o modelo unificado. Apesar do protótipo desenvolvido ser uma primeira versão da abordagem proposta, ele alcançou resultados interessantes, como por exemplo superar a vazão de um provedor de dados utilizado em vários projetos europeus, a partir da utilização de conceitos de particionamento e acesso concorrente.

6. Trabalhos Relacionados

O presente capítulo tem o objetivo de apresentar os trabalhos relacionados ao tema desta dissertação e compará-los com a abordagem proposta, analisando principalmente as características definidas para tratar problemas comumente enfrentados em processamento distribuído de consulta, bem como suporte a particionamento de dados. É apresentado um resumo das principais características destes trabalhos, e, ao final, é apresentada uma discussão comparando estes trabalhos e a abordagem proposta.

6.1. NEXUS

NEXUS (DÜRR et al., 2004) é um sistema que provê informações de contexto com características espaciais e móveis. Utiliza servidores de dados para modelar diferentes tipos de entidades de contexto como ruas, quartos, veículos, postes, etc. Cada servidor do Nexus pode acessar e disponibilizar seu conteúdo de forma federada, criando uma rede de dados para aplicações de contexto. Trabalha com objetos de diferentes localizações geográficas e possui um sistema de banco de dados que armazena serviços de acordo com os tipos de objetos. Nexus prevê que outros sistemas possam ser acoplados e que haja interoperabilidade, precisando para isto que outros sistemas adotem a mesma interface de comunicação. Desta forma, os servidores que implementam esta interface têm compatibilidade para prover informações à plataforma (DÜRR et al., 2004; NICKLAS et al., 2001).

6.2. *Orion Context Broker*

O *Orion Context Broker* (FIWARE, 2019b) faz parte da plataforma de Internet do Futuro - FIWARE, uma iniciativa da União Europeia destinada a fomentar o desenvolvimento de aplicações para cidades inteligentes. Dentro deste arcabouço, o Orion foi desenvolvido para ser o componente responsável, entre outras funções, pelo gerenciamento de dados de contexto. Utiliza padrões de interface e modelo de dados da iniciativa *Open Mobile Alliance*, através das especificações NGSI9 e NGSI10 (OMA, 2012). O acesso é realizado através de uma API REST e o formato dos dados é o padrão JSON (SOTIRIADIS; BESSIS, 2016). Para a representação de entidades, utiliza estruturas de dados genéricas denominadas de *Context Elements*. O Orion implementa os métodos de acesso aos dados de consulta direta e de subscrição (conforme foi conceituado na Seção 2.1.1), como também possui um recurso para criar comunicação entre servidores Orion, denominada Federação (este recurso possibilita o acesso à entidades armazenadas em outros servidores Orion).

6.3. COALITION

O COALITITON (CHEN, 2015) é um *middleware* para sistemas sensíveis ao contexto e apresenta processamento distribuído para a realização de consultas. Seu foco é voltado a dispositivos móveis e para a representação de entidades e ambientes, a partir do conceito de *Mobile Physical Space Gateway* (MPSG), no qual cada MPSG possui seus modelos e métodos específicos. Para este sistema, os autores desenvolveram uma linguagem baseada em SQL para utilizar os recursos provenientes do sistema, e um mecanismo de consulta para resolver o problema de sobrecarga computacional. A partir de uma servidor do COALITION é possível acessar os dados armazenados em outros servidores, de forma transparente, a partir da consulta ao componente central MPSG.

6.4. KNot

O KNot (DE SOUZA, 2017) é uma meta-plataforma que foi desenvolvida por pesquisadores do CESAR (CESAR, 2019) com o intuito de integrar hardware e software de plataformas IoT. O diferencial desta proposta é permitir a comunicação com diferentes tipos de dispositivos, que utilizam diferentes protocolos de comunicação, incluindo dispositivos que não possuem comunicação nativa com a Internet. Esta característica o difere de outras plataformas que exigem comunicação web entre plataforma e dispositivo.

Para a coleta de dados que abrange desde os sensores dos dispositivos até uma aplicação cliente, a plataforma dispõe de três componentes principais: a) dispositivos – chamado também de "*Thing*", é constituído por sensores e atuadores que se conectam ao Gateway por um transmissor de rádio para troca de dados; b) Gateway - é o elemento central que possui conexão Ethernet e rádio, sendo responsável pelo armazenamento dos dados provenientes dos dispositivos para serem acessados por aplicações ou pela Cloud Knot; Cloud - responsável por agregar dados de vários Gateways, tornando-se um ponto de acesso remoto para as aplicações, além de ser responsável pelo roteamento de mensagens para aplicativos e dispositivos (BATISTA, 2018).

6.5. InterSCity

InterSCity (DEL ESPOSTE, 2017) é uma plataforma para cidades inteligentes fundamentada em microsserviços que oferecem as seguintes funcionalidades: a) integração com dispositivos IoT; b) gerenciamento de recurso e dados; c) descoberta de recursos; d) visualização de recursos. Estes microsserviços estão disponíveis através de API REST para mensagens HTTP síncronas, mas também possuem comunicação assíncrona através de outros

protocolos. Esta plataforma suporta a descentralização de gerenciamento dos recursos uma vez que cada microsserviço possui suas respectivas funcionalidades e responsabilidades. Desta forma, cada microsserviço pode evoluir independentemente, como aumentar sua capacidade de processamento com mecanismos de escalabilidade.

6.6. Discussão sobre os Trabalhos Relacionados

A Tabela 5 apresenta características dos trabalhos relacionados já introduzidos. Nota-se que são utilizados pontos marcadores para representar o nível de cada proposta com a respectiva característica, variando de zero a três pontos marcadores. A avaliação zero será atribuída para trabalhos que não abordaram a característica no trabalho consultado ou não possui a característica. A avaliação com um ponto marcador é atribuída para características que envolvem muito esforço do desenvolvedor para configurar a plataforma, ou que exige retrabalho em cada servidor da instância para tal configuração, ou ainda para trabalhos que tenham mencionado a característica sem explicar maiores detalhes. A avaliação com dois marcadores é atribuída para características que estão presentas na plataforma, porém possui uma abordagem arquitetural que limita a escalabilidade ou a interoperabilidade, como por exemplo possuir serviços centralizados. Por último, é atribuído três pontos marcadores para as características dos trabalhos que explicaram o mecanismo, minimizam o esforço do desenvolvedor para a configuração da característica e não possuem fatores arquiteturais que limitam a escalabilidade ou interoperabilidade. As características observadas são:

a) Consulta a bases de dados distribuídas – indica se a proposta possui capacidade para consultar fontes de dados distribuídos como provedores de dados.

b) Processamento distribuído de consulta – indica se a plataforma realiza o processamento de consultas de forma distribuída pelos diversos provedores de dados ou fontes de dados.

c) Interoperabilidade entre diferentes provedores de dados – indica se o sistema possui mecanismos que possibilitem a utilização de provedores de dados heterogêneos, como, por exemplo, a utilização de adaptadores para interfaces.

d) Particionamento por entidade – indica a presença de recursos para realizar consultas a dados organizados conforme a estrutura de particionamento por entidade.

e) Particionamento por atributo – indica a existência de recursos para realizar consultas a dados organizados conforme a estrutura de particionamento por atributo.

Tabela 6: Comparação entre os trabalhos relacionados e o trabalho proposto.

	Consulta a bases de dados distribuídas	Processamento distribuído de consulta	Interoperabilidade entre provedores hete-rogêneos	Particionamento por entidade	Particionamento por atributo
NEXUS	•••	••	••	•••	-
Orion	•	-	-	•	•
COALITION	•••	••	-	•••	-
KNote	-	-	-	-	-
Intescity	•	•	-	•	-
Sirius	•••	•••	•••	•••	•••

A plataforma NEXUS possui as características: a) consulta a bases distribuídas – permite a formação de um tipo de federação de provedores de dados distribuídos; b) processamento distribuído de consulta – cada provedor de dados processa a sua consulta de forma distribuída, porém um dos mecanismos é centralizado (existe apenas tipo de registro de serviços global); c) interoperabilidade entre provedores heterogêneos – prevê que qualquer provedor de dados que implemente uma interface compatível possa ser acoplado à rede federada, porém não detalha os mecanismos necessários; d) particionamento por entidade – possibilita que diferentes entidades sejam representadas em cada provedor, o que caracteriza este particionamento. Desta forma, a plataforma Nexus possui recursos bem elaborados para a integração e interoperabilidade de uma rede de provedores, no entanto não abordou com profundidade os mecanismos de manipulação de provedores heterogêneos nem o particionamento por atributo.

O Orion *Context Broker* possui as características: a) consulta a base de dados distribuídas – as funções majoritárias do Orion são para acesso ao próprio provedor de dados, porém incorpora algumas funcionalidades ainda iniciais de federação de provedores, que exige muito esforço do desenvolvedor e retrabalho para a configuração das localizações para cada provedor participar da consulta distribuída; d) particionamento por entidade – pode realizar a consulta a este tipo de estrutura de particionamento, porém exige retrabalho em cada provedor que for configurado, de modo que quebra a característica de transparência de localização; e) particionamento por atributo – da mesma forma que ocorre com o particionamento por entidade, exige muito trabalho para a configuração. Assim, o Orion *Context Broker* exige que várias configurações sejam feitas até permitir uma consulta a bases

distribuídas. Porém, este recurso ainda é limitado, como por exemplo não possui o processamento distribuído (não processa uma restrição no provedor de origem) e também não possui a capacidade de interoperabilidade entre provedores heterogêneos. Por outro lado, este provedor utiliza um modelo de dados flexível e fácil de ser utilizado.

O *middleware* COALITION apresenta as características: a) consulta a bases distribuídas – este trabalho foca a consulta a diversos provedores de dados; b) processamento distribuído de consulta – ele realiza os procedimento de divisão de consulta e de processamento em cada provedor, suportando assim esta característica, no entanto, utiliza um recurso centralizado para armazenar as definições e localizações dos provedores (por isto não foi atribuído pontuação máxima); d) particionamento por entidade: a forma de organizar os dados pelos provedores é com a representação completa de cada entidade. O COALITION oferece recursos interessantes para a transparência de localização dos provedores de dados, como também recursos em relação ao processamento de consulta. No entanto, não previu recursos para a adoção de provedores de dados heterogêneos, nem suporte ao particionamento de atributos.

A meta-plataforma KNot possui módulos que ainda estão em desenvolvimento e, embora pretenda trabalhar com o conceito de Fog e *Cloud*, isto não se concretizou, sendo uma promessa para trabalhos futuros (DE SOUZA, 2017). Por enquanto, esta pesquisa oferece diferencial em relação à comunicação com dispositivos heterogêneos e sem conexão ethernet nativos. Não foram encontrados maiores detalhes sobre a comunicação de gateways e servidores *Cloud* do KNot, nem o mecanismo necessário para esta interação. Por estes motivos o KNot não pontuou na tabela apresentada. Sobre a característica de processamento distribuído, este também não ocorre, porque apenas poderia ser considerada em relação à comunicação entre servidores *Cloud* e *Gateway*, em que diversos Gateways aliviariam a carga no serviço *Cloud* centralizado.

A plataforma InterSCity possui as características: a) consulta a bases distribuídas – permite que vários servidores locais do módulo de gerenciamento sejam instanciadas e assim distribuir entre elas o processamento, que é descentralizado porém não é distribuído (por este motivo foi atribuído apenas um ponto); b) processamento distribuído de consulta – diferentes microsserviços realizam o processamento de consultas, embora localmente; d) particionamento por entidade – uma vez que a plataforma consegue dividir o gerenciamento e a consulta em diferentes microsserviços, entende-se que está provendo o particionamento por entidade. A partir destas características, conclui-se que a plataforma InterSCity procura prover recursos de escalabilidade em âmbito local, e o processamento descentralizado que realiza

também acontece em âmbito local. Outra característica que está em estudo é a possibilidade de se utilizar diferentes sistemas de banco de dados, estando limitado a um específico (DEL ESPOSTE, 2017).

A plataforma *Sirius* possui as características: a) consulta a bases distribuídas – um servidor *Sirius* pode consultar vários provedores independentes e distribuídos; b) processamento distribuído de consulta – o processamento é distribuído em cada servidor consultado, aliviando a carga no servidor que distribui e integra o resultado das sub-respostas; c) interoperabilidade entre provedores heterogêneos – com a utilização de adaptadores e um modelo de dados unificado, a plataforma provê interoperabilidade; d) particionamento por entidade – através da definição de localização dos provedores no esquema DCDS (LOPES; SILVA; ELIAS, 2019), é possível consultar os dados que estejam em particionamento por entidade; e) particionamento por atributo – a plataforma também provê capacidade de consultar bases de dados que adotam estrutura de particionamento por atributo. A plataforma *Sirius* provê todos as características da tabela apresentada, permitindo a comunicação entre provedores de dados heterogêneos, geograficamente distribuídos e que os dados estejam em diferentes estratégias de particionamento.

6.7. Considerações Finais

Os trabalhos relacionados foram apresentados e analisados neste capítulo. Primeiramente NEXUS, um projeto financiado pela União Europeia que apresentou recursos de processamento distribuído. Este trabalho abordou apenas um tipo de particionamento, o particionamento por atributo e apenas acena a possibilidade de interoperabilidade entre diferentes provedores de dados (com a implementação de interface compatível), sem aprofundar outros detalhes necessários. Outro fator que limita esta proposta para o volume de processamento que aplicações para cidades inteligentes exigem é forma que o projeto arquitetou o componente registro de serviço geral, um componente de abordagem centralizada, que está vulnerável a sobrecarga, por exemplo. Desta forma, o *Sirius* se diferencia por permitir que informações sobre a localização dos dados distribuídos sejam armazenados em diferentes servidores da plataforma, além de permitir a utilização e adotar mecanismos de suporte a provedores de dados heterogêneos.

Alguns trabalhos apresentam arquitetura centralizada, como o Orion *Context Broker*, KNote e InterSCity. Esses trabalhos propõem provedores de dados que manipulam e armazenam dados de dispositivos IoT. O KNote propõe uma estrutura que interliga vários Gateways a um servidor Cloud, porém não apresentou os detalhes e algumas publicações

sinalizam que esta etapa ainda está em desenvolvimento. O InterSCity foca o desenvolvimento baseado em microsserviços para prover escalabilidade, mas não prevê a distribuição geográfica destes módulos. Enquanto o Orion, majoritariamente possui funções centralizadas, apresenta alguns recursos no intuito de constituir uma federação de servidores da plataforma. Através destes recursos, o Orion pode suportar as duas estruturas de particionamento citadas. No entanto, o ponto que foi levantado sobre este recurso do Orion é que ele exige muito esforço de configuração, repetido para cada entidade e cada servidor que deseje "enxergar" a federação de servidores. O *Sirius* se diferencia por permitir o processamento distribuído e a consulta a dados distribuídos geograficamente de forma transparente, a partir de qualquer servidor da plataforma, sem a necessidade de configurações individuais que demandem esforço, mas simplesmente a partir da utilização do esquema DCDS.

De todos os trabalhos relacionados elencados, o que apresentou recursos que mais se aproximaram da abordagem proposta foi o COALITION. No entanto, não apresentou recursos de interoperabilidade entre provedores heterogêneos e a estratégia de consulta previa apenas o particionamento por entidade. Assim como o Nexus, adota uma estrutura centralizada para o componente que armazena a estrutura de localização dos recursos.

Em relação aos trabalhos apresentados, a plataforma *Sirius* se diferencia por permitir que consultas sejam realizadas de forma transparente a dados correlacionados, que se encontram em provedores de dados heterogêneos e distribuídos geograficamente. A plataforma *Sirius* está alicerçada no metamodelo DCDS que deriva tanto a especificação de entidades de contexto, que define uma entidade e a localização de seus dados, quanto a instanciação de entidades, que permite a utilização de um modelo de dados unificado para a plataforma distribuída prover interoperabilidade entre diferentes modelos de dados heterogêneos. Este trabalho forma uma infraestrutura como um conjunto inter-relacionado de provedores de dados que podem estar organizados com particionamento por entidade e por atributo.

7. Conclusão

Neste trabalho foi apresentada a abordagem de consulta distribuída e particionada para plataformas de *middleware* para cidades inteligentes, que tem por objetivo prover, de forma transparente, a consulta a conjuntos correlacionados de dados que encontram-se distribuídos por provedores de dados heterogêneos e geograficamente dispersos. Para isso, a abordagem utiliza um metamodelo chamado DCDS, do qual é possível derivar dois elementos: um esquema (modelo) de dados de entidades de contexto e uma representação de dados das instâncias destas entidades de contexto.

Várias plataformas de *middleware* têm sido propostos com o objetivo de prover uma visão integrada e holística dos objetos de uma cidade, entretanto, uma das principais limitações das propostas atuais é que elas são projetadas para dar suporte a apenas um tipo de provedor ou a apenas uma instância de provedor, sem considerar estratégias de particionamento, por exemplo. Essa restrição limita o escopo destas plataformas, no sentido de que elas não podem facilmente escalar, ou interoperar com novos provedores, que representam dois requisitos importantes para um sistema para cidades inteligentes.

A abordagem proposta, por sua vez, oferece suporte para consultar diferentes estruturas de particionamento, tornando-a apta a atender uma ampla variedade de cenários de divisão e organização dos dados, sendo assim adaptável e flexível a diferentes necessidades (geográficas, financeiras, organizacionais, etc.). Outra característica presente na abordagem é que permite a interoperabilidade com diferentes tipos de provedores de dados através do uso de mecanismos como adaptadores, desta forma, favorecendo o reuso dos dados. A vantagem disto é permitir a plataforma poder se comunicar tanto com sistemas legados ou outros que ainda estão por surgir.

Com a finalidade de ilustrar os benefícios da abordagem proposta em um ambiente distribuído, um experimento foi implementado utilizando dez servidores *Sirius* para consultar uma base de 10.000 entidades. Durante o experimento foram realizados avaliações de desempenho em diferentes cenários, explorando diferentes estruturas de particionamento, quantidade de servidores *Sirius* e tipos de consultas. Durante estes experimentos ficou evidente a facilidade de consulta a diversos servidores da plataforma e a melhora do desempenho de consultas com a utilização de particionamento.

7.1. Contribuições

As principais contribuições do trabalho foram:

a) a concepção da abordagem de consulta distribuída e particionada para plataformas de *middleware*, em que é possível consultar de forma integrada conjuntos de dados correlacionados dispersos por provedores de dados;

b) a concepção e especificação do metamodelo DCDS como vértice de sustentação para vários mecanismos de transparência e integração de dados;

c) o desenvolvimento de um mecanismo que explorou o conceito de esquema de entidade do metamodelo DCDS para permitir, de forma automatizada e transparente a usuários externos realizar consultas a dados de servidores distribuídos, com diferentes estruturas de particionamento;

d) a obtenção de um mecanismo para desacoplar a representação dos dados armazenados em provedores de dados locais de diferentes formatos, através da utilização de adaptadores para traduzir cada tipo de representação para o modelo de dados DCDS;

e) a adaptação da linguagem de consulta de um grande projeto da comunidade europeia para cidades inteligentes, permitido que de forma simples, seja representado consultas a instâncias de entidades definidas em esquemas de entidades, na qual a localização e forma de armazenamento ficam transparente para quem realiza consultas utilizando sua sintaxe;

f) a obtenção dos resultados de experimentos com diversos cenários de particionamento, tipos de consulta e volume de respostas, podendo analisar vantagens e desvantagens de cada configuração testada.

g) a identificação da vantagem da estratégia de particionamento de dados por entidade em relação a estratégia de particionamento de dados por atributo para os cenários avaliados.

7.2. Limitações e Trabalhos Futuros

Algumas questões não foram exploradas no escopo deste trabalho, constituindo trabalhos a serem desenvolvidos no futuro, dentre elas:

- **Controle de acesso e privacidade:** estes requisitos são cruciais para arquiteturas distribuídas para prover segurança em um ambiente de integração de dados com várias organizações e atores presentes.
- **Resposta parcial de uma consulta:** permitir que a plataforma retorne respostas parciais a partir de prévia configuração, que se aplicaria para os casos em que algum servidor remoto esteja incomunicável.

- **Desenvolver novos adaptadores**: para ampliar a validação da abordagem na direção de permitir que a plataforma manipule diferentes formatos de dados;
- **Aperfeiçoamento da otimização de consultas**: A abordagem proposta demostrou uma série de vantagens frente a outros trabalhos, no entanto, o desenvolvimento de estratégias de otimização de consultas, assim como mecanismos para explorar a topologia e fluxo de execução de uma consulta são sugestões para trabalhos futuros.
- **Evoluir a Gramática EBNF para contemplar consultas relacionadas a metadados de atributos**: Além de definir uma consulta por características de atributos, é desejável que seja levado em consideração características presentes nos metadados dos atributos das entidades de contexto.
- **Mecanismo de consulta *publish-subscribe***: implementar o mecanismo e incorporar uma linguagem para acessar os recursos deste mecanismo é importante para um sistema para cidades inteligentes.

8. Referências

ABELLÁ-GARCÍA, A.; ORTIZ-DE-URBINA-CRIADO, M.; DE-PABLOS-HEREDERO, C. **The ecosystem of services around smart cities**: An exploratory analysis. Procedia Computer Science, v. 64, p. 1075-1080. 2015.

ABOWD, G. D. et al. Towards a better understanding of context and context-awareness. *In*: **International Symposium on Handheld and Ubiquitous Computing**. Springer, Berlin, Heidelberg, 1999. p. 304-307.

AGRAWAL, S.; NARASAYYA, V.; YANG, B.. Integrating vertical and horizontal partitioning into automated physical database design. *In*: **Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data**. ACM, 2004. p. 359-370.

APACHE. Apache Jmeter. Disponível em: <https://jmeter.apache.org/> Acessado em 20 Jul. 2019.

AYRES, F. **QEEF-Uma Máquina De Execução de Consultas Extensível. Tese de Doutorado**. Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática. 2003.

BATISTA, C. P. et al. Integrando as Plataformas FIWARE e KNoT para o Desenvolvimento de Aplicações de Internet das Coisas. *In*: **10º Simpósio Brasileiro de Computação Ubíqua e Pervasiva (SBCUP 2018)**. SBC. 2018.

BELLAVISTA, P. et al. A survey of context data distribution for mobile ubiquitous systems. **ACM computing surveys (CSUR)**, v. 44, n. 4, p. 24, 2012.

BOLCHINI, C. et al. A data-oriented survey of context models. **ACM Sigmod Record**, v. 36, n. 4, p. 19-26, 2007.

BOUSSARD, M. et al. A process for generating concrete architectures. *In*: ***Enabling Things to Talk***. Springer, Berlin, Heidelberg, 2013. p. 45-111.

CASANOVA, M. A. **Princípios de sistemas de gerência de bancos de dados distribuídos**. Campus, 1999.

CESAR. **CESAR - Centro de Estudos e Sistemas Avançados do Recife**. Disponível em: <https://www.cesar.org.br/>. Acessado em 01 Jul. 2018.

CHEN, P. **A Distributed Approach To Context Data Processing**. 2015. Tese de Doutorado. Department Of Electrical & Computer Engineering. National University Of Singapore. 2015.

CHEN, P. et al. A SQL-based Context Query Language for Context-aware Systems. *In*: **IMMM 2014: The Fourth International Conference on Advances in Information Mining and Management**. no. c, p. 96-102, 2014.

CODATO, A. C. de O. C. **Processamento de consultas em bancos de dados de diferentes tecnologias**. Dissertação de Mestrado. Departamento de Informática e Estatística. UFSC. 2001.

da SILVA, W. M. et al. Smart cities software architectures: a survey. *In*: **Proceedings of the 28th Annual ACM Symposium on Applied Computing**. ACM, p. 1722-1727. 2013.

DASZCZUK, W. B. Communication and Resource Deadlock Analysis Using IMDS Formalism and Model Checking. **The Computer Journal**, v. 60, n. 5, 2016.

de SOUZA, D. A. M. **Estudo de Lacunas da Meta-Plataforma KNoT para IoT**. Projeto de Graduação. Centro de Informática, UFPE, Brasil, 2017

DEL ESPOSTE, A. M. et al. InterSCity: A Scalable Microservice-based Open Source Platform for Smart Cities. *In*: **SMARTGREENS**, p. 35-46, 2017.

DOCKER. Enterprise Container Platform. Disponível em: <https://www.docker.com/>. Acessado em 08 Jul. 2019.

DÜRR, F. et al. Nexus–a platform for context-aware applications. **Roth, Jörg, editor**, v. 1, p. 15-18, 2004.

ELLER, N. A. **Estudo e implementação de um sistema de banco de dados distribuido**. Departamento de Informática e Estatística. UFSC. Dissertação de Mestrado. 1997.

FAZIO, M., et al. Heterogeneous sensors become homogeneous things in smart cities. *In*: **2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing**. IEEE, p. 775-780. 2012.

FIWARE. Fiware-NGSI v2 Specification. FIWARE FOUNDATION. Disponível em <http://telefonicaid.github.io/fiware-orio/api/v2/stable/>. Acessado em 20 Jul. 2019.

FIWARE. Fiware-Orion Documentation. FIWARE FOUNDATION. Disponível em <http://fiware-orion.readthedocs.io>. Acessado em 20 Jul. 2019.

FIWARE. Knowage and NGSI. FIWARE FOUNDATION. Disponível em <https://knowage.readthedocs.io/en/6.1.1/user/NGSI/README/index.html. Acessado em 20/07/2019>. Acessado em 01 Jul. 2018.

FIWARE. What is Fiware. FIWARE FOUNDATION. Disponível em <https://www.fiware.org/developers/. Acessado em 20/07/2019>. Acessado em 01 Jul. 2018.

GOOGLE. Google Maps. Disponível em <https://www.google.com.br/maps>. Acessado em 20 Jul. 2019.

GRINBERG, M. **Flask web development: developing web applications with python**. " O'Reilly Media, Inc.". 2018.

GUNICORN. Gunicorn - Python WSGI HTTP Server for UNIX. Disponível em <https://gunicorn.org/>. Acessado em 20 Jul. 2019.

HALEVY, A. Y. Data integration: A status report. *In*: **Proceedings of 10th Conference on Database Systems for Business Technology and the Web (BTW 2003)**, Germany, 2003.

HASSANI, A. et al. CDQL: A Generic Context Representation and Querying Approach for Internet of Things Applications. *In*: **Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media**. ACM, p. 79-88. 2016.

GAMMA, E.. **Design patterns: elements of reusable object-oriented software**. Pearson Education India, 1995.

HERNÁNDEZ-MUÑOZ, J. M. et al. Smart cities at the forefront of the future internet. *In*: **The future internet assembly**. Springer, Berlin, Heidelberg, p. 447-462. 2011.

HURSON, A. R.; JIAO, Y. Database system architecture–A walk through time: From centralized platform to mobile computing–keynote address. *In*: **International Symposium and School on Advanced Distributed Systems**. Springer, Berlin, Heidelberg, p. 1-9. 2005.

FIELDING, R. et al. Hypertext Transfer Protocol–HTTP/1.1, January 1997. Internet Proposed Standard RFC, v. 2068. 1997.

JALALI, R.; EL-KHATIB, K.; MCGREGOR, C. Smart city architecture for community level services through the internet of things. *In*: **2015 18th International Conference on Intelligence in Next Generation Networks**. IEEE, p. 108-113. 2015.

JARA, A. J. et al. Semantic web of things: an analysis of the application semantics for the iot moving towards the iot convergence. Inte**rnational Journal of Web and Grid Services**, v. 10, n. 2-3, p. 244-272. 2014.

JSON. JavaScript Object Notation, Disponível em: <http://www.json.org>. Acessado em 05 de Abr. 2019.

KHAN, R. et al. Future internet: the internet of things architecture, possible applications and key challenges. *In*: **Frontiers of Information Technology (FIT), 2012 10th International Conference on frontiers of information technology**. IEEE, 2012. p. 257-260.

KON, F.; SANTANA, E. F. Z.. Cidades inteligentes: conceitos, plataformas e desafios. **Jornadas de Atualização em Informática**, 2016.

LOPES, J.; SILVA, L.; ELIAS, G. A Context Data Metamodel for Distributed Middleware Platforms in Smart Cities. *In*: **11th International Conference on Advances in Databases, Knowledge, and Data Applications**. DBKDA, 2019. p. 39-45

MARTINE, G. et al. State of world population 2007: unleashing the potential of urban growth. *In*: **State of world population 2007: unleashing the potential of urban growth**. UNFPA, 2007.

MASSE, M. **REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces**. " O'Reilly Media, Inc.", 2011.

MCGEE, W. C. On user criteria for data model evaluation. **ACM Transactions on Database Systems (TODS)**, v. 1, n. 4, p. 370-387, 1976.

NADAI, F. P. **Acesso e integração de dados para sistemas context-aware**. Projeto de Graduação. Departamento de Informática, UFES, Brasil, 2006.

NAPHADE, M. et al. Smarter cities and their innovation challenges. **Computer**, v. 44, n. 6, p. 32-39, 2011.

NICKLAS, D. et al. A model-based, open architecture for mobile, spatially aware applications. *In*: **International Symposium on Spatial and Temporal Databases**. Springer, Berlin, Heidelberg, 2001. p. 117-135.

OLIVEIRA, N. Q. de. **Uma arquitetura para acesso e integração de dados em sistemas sensíveis ao contexto**. Dissertação de Mestrado. Centro de Tecnológico. Universidade Federal do Espírito Santo. 2007.

OMA. NGSI Context Management. Technical Report V1. Open Mobile Alliance. 2012. Disponível em <http://www.openmobilealliance.org/release/NGSI/V1_0-20120529-A/OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf>. Acessado em 19 Jul. 2018.

ÖZSU, M. T.; VALDURIEZ, P. **Principles of Distributed Database Systems**. Springer Science & Business Media, 2011.

QUANTUMLEAP. QUANTUMLEAP. Disponível em: < https://quantumleap.readthedocs.io>. Acessado em 20 jul. 2019.

PCAST. Technology and the future of cities, report to the president. **Technical report, Executive Office of the President**, United States. 2016. Disponível em: https://www.whitehouse.gov/sites/whitehouse.gov/files/images/Blog/PCAST%20Cities%20Report%20_%20FINAL.pdf. Acessado em 08 Jul. 2018.

PERERA, C. et al. Context aware computing for the internet of things: A survey. **IEEE communications surveys & tutorials**, v. 16, n. 1, p. 414-454, 2013.

PERTTUNEN, M.; RIEKKI, J.; LASSILA, O. Context representation and reasoning in pervasive computing: a review. **International Journal of Multimedia and Ubiquitous Engineering**, v. 4, n. 4, 2009.

REICHLE, R. et al. A context query language for pervasive computing environments. *In*: **2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)**. IEEE, 2008. p. 434-440.

SILVA, I. D. P. **IoT Standards for Smart Cities**. Dissertação de Mestrado. Departamento de Eletrôonica, Telecomunicações e Informática. Universidade de Aveiro. 2016

SOTIRIADIS, S.; BESSIS, N. An inter-cloud bridge system for heterogeneous cloud platforms. **Future Generation Computer Systems**, v. 54, p. 180-194, 2016.

SPTRANS. Olho Vivo – Sistema de Moniotramento do Transporte. Disponível em: < http://olhovivo.sptrans.com.br/>. Acessado em 20 Jul. 2019

UNITED NATIONS. World Urbanization Prospects the 2011 Revision - Highlights. New York: **United Nations**, 2012. p. 33.

VILLANUEVA, F. J. et al. Civitas: The smart city middleware, from sensors to big data. *In*: **2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing**. IEEE, 2013. p. 445-450.

van KRANENBURG, R. **The Internet of Things: A critique of ambient technology and the all-seeing network of RFID**. Institute of Network Cultures, 2008.

WHITE, R.; TANTSURA, J. E. **Navigating Network Complexity: Next-generation routing with SDN, service virtualization, and service chaining**. Addison-Wesley Professional, 2015.

WIEDERHOLD, G. Mediators in the Architecture of Future Information Systems. Computer, v. 25, n. 3, p. 38-49, 1992.

ZUNKE, S.; D'SOUZA, V. JSON vs XML: A Comparative Performance Analysis of Data Exchange Formats. **IJCSN International Journal of Computer Science and Network**, v. 3, n. 4, 2014.

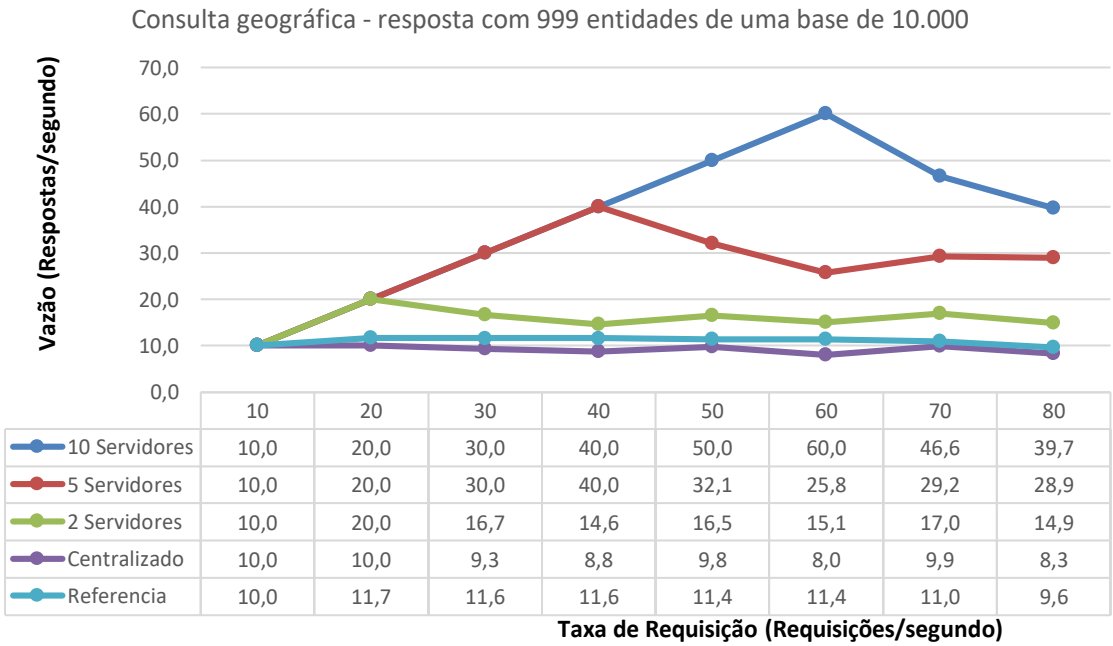
APÊNDICE A - Discussão dos Resultados da Consultas Geográfica com Menor Volume de Resposta

Este apêndice apresenta a discussão sobre os resultados dos experimentos da consulta geográfica com menor volume de dados. Estas consultas foram apresentadas no Capítulo 5, porém foram posicionadas no Apêndice A em virtude de que apresentaram comportamentos semelhantes à consulta não geográfica com menor volume de resposta, tendo sido avaliado pertinente limitar as discussões realizadas no Capítulo 5 apenas às consultas classificadas como não geográficas. O conteúdo está organizado pela sequência dos resultados dos cenários com particionamento de dados por entidade e por atributo.

Este conjunto de análises abordou uma consulta geográfica que retorna todos os ônibus da cidade de São Paulo que se encontram a 5.400 metros de distância da coordenada geográfica apresentada na Seção 5.1.2, considerando os dados reais capturados do sistema de transporte público da Prefeitura de São Paulo. Esta distância foi escolhida para que a quantidade de instâncias de entidade presentes no resultado fosse o mais próximo da consulta analisada pela consulta não geográfica com menor volume de resposta (993 instâncias de entidade). A atual consulta retorna 999 instâncias de entidade.

A Figura 31 apresenta os resultados da análise de desempenho para uma consulta geográfica com menor volume de resposta no cenário com particionamento de dados por entidade. No gráfico é possível perceber que maiores registros da vazão do sistema para os cenários que são formados por maior quantidade de servidores *Sirius*. O melhor resultado registrado foi para o cenário com 10 servidores *Sirius*, registrando uma vazão de 60 resp/s. Para a mesma taxa de requisição, o cenário de referência possui uma vazão de 11,4 resq/s, proporcionalmente o primeiro cenário tem vantagem de 426,3%.

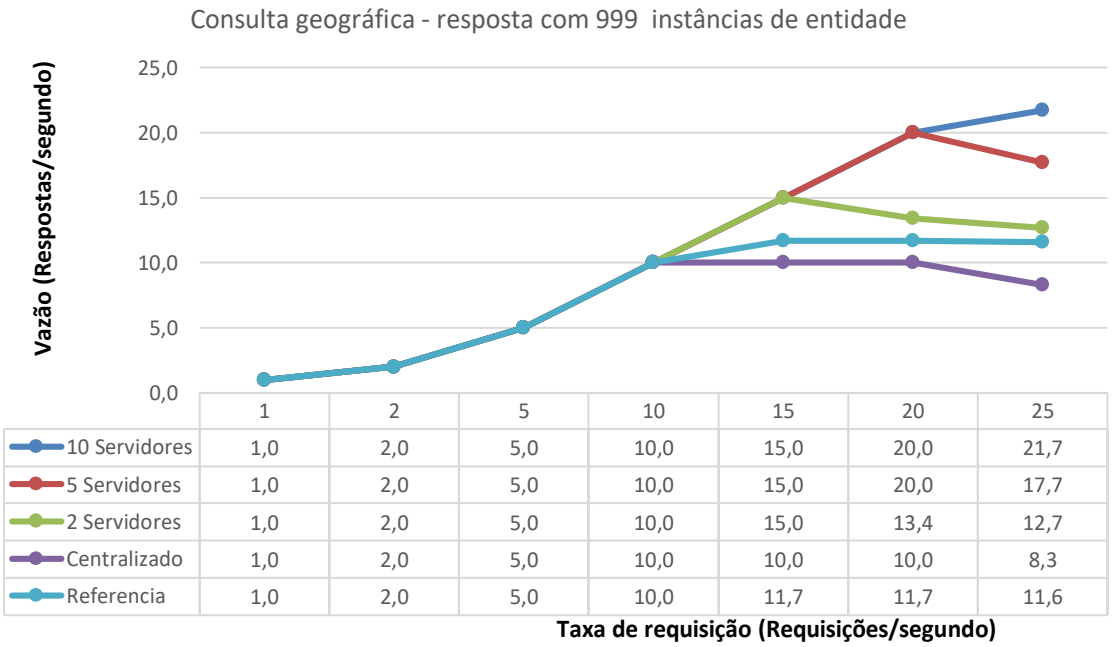
Figura 31 : Particionamento por Entidade. Consulta geográfica com menor volume de resposta



	10	20	30	40	50	60	70	80
10 Servidores	10,0	20,0	30,0	40,0	50,0	60,0	46,6	39,7
5 Servidores	10,0	20,0	30,0	40,0	32,1	25,8	29,2	28,9
2 Servidores	10,0	20,0	16,7	14,6	16,5	15,1	17,0	14,9
Centralizado	10,0	10,0	9,3	8,8	9,8	8,0	9,9	8,3
Referencia	10,0	11,7	11,6	11,6	11,4	11,4	11,0	9,6

Por sua vez, a Figura 32 apresenta os resultados da análise de desempenho para uma consulta geográfica com menor volume de resposta no cenário com particionamento por atributo. No gráfico é possível perceber que o melhor resultado registrado foi para o cenário com 10 servidores *Sirius*, registrando uma vazão de 21,7 resp/s. Para a mesma taxa de requisição, o cenário de referência possui uma vazão de 11,6 resp/s, proporcionalmente o primeiro cenário tem vantagem de 87%.

Figura 32 : Particionamento por Atributo. Consulta geográfica com menor volume de resposta



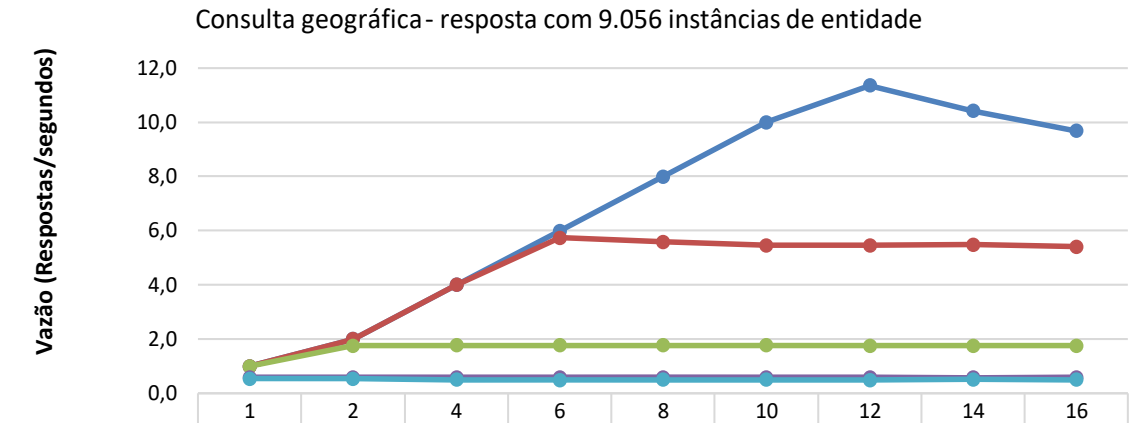
	1	2	5	10	15	20	25
10 Servidores	1,0	2,0	5,0	10,0	15,0	20,0	21,7
5 Servidores	1,0	2,0	5,0	10,0	15,0	20,0	17,7
2 Servidores	1,0	2,0	5,0	10,0	15,0	13,4	12,7
Centralizado	1,0	2,0	5,0	10,0	10,0	10,0	8,3
Referencia	1,0	2,0	5,0	10,0	11,7	11,7	11,6

APÊNDICE B - Discussão dos Resultados da Consultas Geográfica com Maior Volume de Resposta

Este apêndice apresenta a discussão sobre os resultados dos experimentos da consulta geográfica com maior volume de dados. A descrição das consultas podem ser conferidas no Capítulo 5. O conteúdo está organizado pela sequência dos resultados dos cenários com particionamento por entidade e o particionamento por atributo.

Neste último conjunto de testes foram analisados os desempenhos dos cenários para uma consulta geográfica que retorna 9.056 entidades. Conforme o gráfico da Figura 33, é possível observar que as linhas da vazão dos cenários de referência e centralizado ficam praticamente empatadas (0,5 resp/s), seguidas pelo cenário com dois servidores (1,8 resp/s). Depois, com uma diferença significativa encontra-se o cenário com cinco servidores (5,5 resp/s) e, por último, com a melhor vazão, o cenário com 10 servidores (11,4 resp/s). Ao registrar a taxa de 12 req/s no cenário com 10 servidores, ou seja, a melhor vazão deste cenário, sua vantagem em relação ao cenário de referência é de 2.180%.

Figura 33 : Particionamento por Entidade. Consulta geográfica com maior volume de resposta



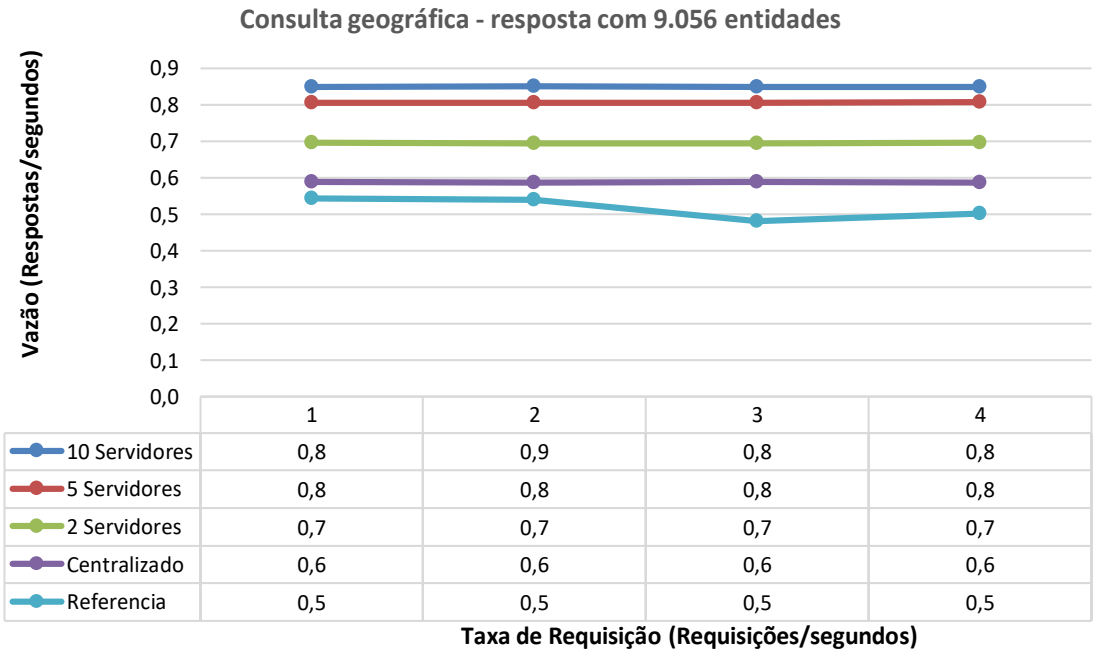
	1	2	4	6	8	10	12	14	16
10 Servidores	1,0	2,0	4,0	6,0	8,0	10,0	11,4	10,4	9,7
5 Servidores	1,0	2,0	4,0	5,7	5,6	5,5	5,5	5,5	5,4
2 Servidores	1,0	1,8	1,8	1,8	1,8	1,8	1,8	1,8	1,8
Centralizado	0,6	0,6	0,6	0,6	0,6	0,6	0,6	0,6	0,6
Referencia	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5

A Figura 34 apresenta o gráfico para o particionamento de dados por atributo, este gráfico apresenta-se diferente de todos os últimos (os gráficos do Apêndice A e do Apêndice B), pois a vazão dos diferentes cenários mantiveram-se praticamente estáveis a medida que a

taxa de requisição aumentou. Analisando melhor e com a experiência ao decorrer dos experimentos, supõe-se que as taxas de requisição utilizadas foram superiores ao ponto de saturação. Ou seja, não é possível ver neste gráfico a evolução acompanhada nos demais gráficos.

O melhor caso foi registrado para o cenário com 10 servidores e registrou vazão de 0,9 resp/s, seguido do cenário com cinco servidores, com vazão de 0,8 resp/s. Na sequência, o cenário com dois servidores, com vazão de 0,7 resp/s, seguido do cenário centralizado, com 0,6 resp/s e, na última posição, o cenário de referência, que obteve 0,5 resp/s de vazão.

Figura 34 : Particionamento por Atributo. Consulta geográfica com maior volume de resposta



	1	2	3	4
10 Servidores	0,8	0,9	0,8	0,8
5 Servidores	0,8	0,8	0,8	0,8
2 Servidores	0,7	0,7	0,7	0,7
Centralizado	0,6	0,6	0,6	0,6
Referencia	0,5	0,5	0,5	0,5