Os bancos de dados não relacionais, comumente chamados de NoSQL, oferecem soluções alternativas de armazenamento que podem ser vantajosas para o gerenciamento de dados geoespaciais. E o padrão SOA, utilizando microsserviços, facilita a inclusão de novos componentes de armazenamento e processamento de trajetórias.
Este trabalho apresenta um arcabouço estrutural para implantação de serviços de processamento de trajetórias em conjunto com o uso da persistência poliglota que busca aumentar a produtividade dos desenvolvedores de aplicações para trajetórias.

Orientador: Fabiano Baldo

Joinville, 2016

ANO
2016

GLAUCIO SCHEIBEL | UMA ARQUITETURA DE SOFTWARE PARA ARMAZENAMENTO DE TRAJETÓRIAS POR MEIO DA TÉCNICA DE PERSISTÊNCIA POLIGLOTA

UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
CURSO DE MESTRADO EM COMPUTAÇÃO APLICADA

DISSERTAÇÃO DE MESTRADO

UMA ARQUITETURA DE SOFTWARE PARA ARMAZENAMENTO DE TRAJETÓRIAS POR MEIO DA TÉCNICA DE PERSISTÊNCIA POLIGLOTA

GLAUCIO SCHEIBEL

JOINVILLE, 2016

GLAUCIO SCHEIBEL

UMA ARQUITETURA DE SOFTWARE PARA ARMAZENAMENTO DE TRAJETÓRIAS POR MEIO DA TÉCNICA DE PERSISTÊNCIA POLIGLOTA

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada da Universidade do Estado de Santa Catarina, como requisito parcial para obtenção do grau de Mestre em Computação Aplicada.

Orientador: Fabiano Baldo

JOINVILLE, SC
2016

FICHA CATALOGRÁFICA

S318u Scheibel, Glaucio
Uma arquitetura de software para armazenamento de trajetórias por meio da técnica de persistência poliglota / Glaucio Scheibel. – 2016.
148 p. : il. ; 21 cm

Orientador: Fabiano Baldo
Bibliografia: p. 117-128
Dissertação (mestrado) – Universidade do Estado Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada, 2016.

1. Engenharia de software. 2. Trajetórias. 3. NoSQL. 4. SOA.
I. Baldo, Fabiano. II. Universidade do Estado de Catarina. Programa de Pós-Graduação em Computação Aplicada. III. Título.

CDD 005.1 – 23.ed.

GLAUCIO SCHEIBEL

UMA ARQUITETURA DE SOFTWARE PARA ARMAZENAMENTO DE TRAJETÓRIAS POR MEIO DA TÉCNICA DE PERSISTÊNCIA POLIGLOTA

Dissertação apresentada ao Curso de Mestrado Acadêmico Computação Aplicada como requisito parcial para obtenção do título de Mestre em Computação Aplicada na área de concentração "Ciência da Computação".

Banca Examinadora

Orientador:

Prof. Dr. Fabiano Baldo
CCT/UDESC

Membros

Prof. Dr. Fernando José Braz
IFC/Araquari

Prof. Dr. Renato Fileto
INE/UFSC

Prof. Dr. Rui Jorge Tramontin Junior
CCT/UDESC

Joinville,SC, 26 de fevereiro de 2016.

Este trabalho é dedicado aos meus pais que,
com muito amor, sempre investiram na minha educação.

Agradecimentos

Ao meu orientador, professor Fabiano Baldo, que muito apoiou, cobrou e colaborou na construção deste trabalho.

Aos professores Avanilde Kemczinski, Carla D. M. Berkenbrock, Cristiano D. Vasconcellos, Isabela Gasparini, Marcelo da S. Hounsell e Omir C. Alves Junior que passaram não somente os seus conhecimentos, como também valiosos conselhos.

Aos novos amigos Alexandre A. de Melo, Mauro Hinz e Wilcilene Mª. K. Schratzenstaller por inúmeras horas de trabalho e divertidas discussões.

E aos velhos amigos Glauco V. Scheffel e Luciana R. Guedes que sempre incentivaram os estudos e o mestrado.

"Se eu vi mais longe,
foi por estar sobre ombros de gigantes."
(Isaac Newton)

RESUMO

SCHEIBEL, Glaucio. **Uma arquitetura de software para armazenamento de trajetórias por meio da técnica de persistência poliglota**. 2016. 148 p. Dissertação (Mestrado em Computação Aplicada - Área: Engenharia (*Engineering*)). Universidade do Estado de Santa Catarina. Programa de Pós-Graduação em Computação Aplicada. Joinville, 2016.

Com a popularização dos dispositivos eletrônicos móveis providos com GPS, o volume de dados de trajetórias de objetos móveis produzidos diariamente vem aumentando de forma exponencial. Tradicionalmente, os sistemas de gerenciamento de banco de dados relacionais vêm sendo usados para armazenar e gerenciar os dados de trajetórias. Entretanto, apesar da sua ampla utilização, eles não suportam adequadamente o gerenciamento de dados semi e não estruturados, assim como o armazenamento em *cluster* com escalabilidade horizontal, requisitos fundamentais para aplicações que manipulam *Big Data*. Os bancos de dados não relacionais, comumente chamados de NoSQL, oferecem soluções alternativas de armazenamento que buscam solucionar estes problemas e podem ser uma opção para o gerenciamento de dados geoespaciais, tais como as trajetórias. As aplicações que processam grandes volumes de dados podem se beneficiar das características providas pelos bancos NoSQL. Entretanto, estes benefícios ainda são pouco explorados no contexto de trajetórias. Isso pode ser justificado pela complexidade associada ao uso destes diferentes modelos, principalmente quando usados de forma conjunta na chamada persistência poliglota. O presente trabalho assume que prover um arcabouço estrutural para implantação de serviços de processamento de trajetórias pode abstrair parte da complexidade do uso da persistência poliglota e assim aumentar a produtividade dos desenvolvedores de aplicações que usem modelos NoSQL para este domínio. Portanto, o objetivo deste trabalho é propor uma arquitetura que sirva como base para a criação e implantação de serviços para aplicações que processem trajetórias

de objetos móveis. A arquitetura deve prover uma camada de persistência poliglota para facilitar o armazenamento de dados semi e não estruturados associados às trajetórias. Os resultados das experiências demonstram que o modelo NoSQL é mais adequado no armazenamento de dados de trajetórias em termos de performance e flexibilidade, além de reduzir a complexidade do código fonte. Os resultados também mostram que o padrão SOA com microsserviços facilita a inclusão de novos componentes de armazenamento e processamento de trajetórias.

Palavras-chaves: Trajetórias. NoSQL. SOA. Microsserviços.

ABSTRACT

With the popularity of mobile electronic devices fitted with GPS, the volume of mobile objects trajectory data produced daily is increasing exponentially. Traditionally, relational database management systems have been used to store and manage trajectory data. However, despite its widespread use, they do not adequately support the semi and unstructured data, as well as clustering with horizontal scalability, core requirement for applications that handle Big Data. Non-relational databases, commonly referred as NoSQL, offer alternative storage solutions that seek to address these problems and can be an option for the management of geospatial data, such as trajectory data. Applications that process trajectory data can benefit from the features provided by NoSQL databases. However, these benefits are still unexplored in trajectory domain. This can be justified by the complexity associated with the use of these different models, especially when they are used together as the polyglot persistence. This paper supposes that a structural framework for the implementation of trajectory processing services can abstract of the complexity of persistence polyglot and thus increase the productivity of application developers that use NoSQL models for this domain. Therefore, the aim of this study is to propose an architecture that serves as the basis for creation and deployment of services to applications for managing moving objects data. The architecture should provide a polyglot persistence layer to facilitate storing trajectories data with semi structured and unstructured data. The results of experiments show that the NoSQL model is best suited to the storage of data trajectories in terms of performance and flexibility, while reducing the complexity of the source code. The results also show that the standard SOA with microservices facilitates the addition of new storage and processing components.

Key-words: Trajectories. NoSQL. SOA. Microservices

Lista de ilustrações

Lista de quadros

Lista de tabelas

Lista de abreviaturas e siglas

ACID	Atomicidade, Consistência, Isolamento e Durabilidade (*Atomicity, Consistency, Isolation, Durability*)
ANSI	Instituto Nacional Americano de Padrões (*American National Standards Institute*)
API	Interface de Programação de Aplicação (*Application programming interface*)
BASE	Disponibilidade básica, Estado Relaxado, Consistência Eventual (*Basically Available, Soft-state, Eventual-consistency*)
BLOB	Grandes Objetos Binários (*Binary Large OBject*)
BSON	JSON binário (*Binary jSON*)
DBRMS	Sistema Gerenciador de Banco de Dados Relacional (*DataBase Relational Management System*)
CAP	Consistência, Disponibilidade, Tolerante ao Particionamento (*Consistency, Availability, Partition tolerance*)

CQL	Linguagem de Consulta do Cassandra (*Cassandra Query Language*)
CRUD	Criação, Leitura, Atualização e Remoção (*Create, Read, Update and Delete*)
DAO	Objeto de Acesso a Dados (*Data Access Object*)
DDL	Linguagem de Definição de Dados (*Data Definition Language*)
DML	Linguagem de Manipulação de Dados (*Data Manipulation Language*)
DNS	Serviço de Nomes de Domínio (*Domain Name System*)
EIP	Padrões de Integração Empresarial (*Enterprise Integration Patterns*)
ESB	Barramento de Serviços Empresariais (*Enterprise Service Bus*)
GIS	Sistema de Informação Geográfica (*Geographic Information System*)
GPS	Sistemas de Posicionamento Global (*Global Positioning System*)
HATEOAS	Hipertexto Como O Motor de Estado De uma Aplicação (*Hypertext As The Engine Of Application State*)
HDFS	Sistema de Arquivos Distribuído do Hadoop (*Hadoop Distributed FileSystem*)
HTML	Linguagem de Marcação de Hipertexto (*HyperText Markup Language*)
HTTP	Protocolo de Transferência de Hipertexto (*HyperText Transfer Protocol*)

IP	Protocolo de Internet (*Internet Protocol*)
JSON	Notação de Objetos JavaScript (*JavaScript Object Notation*)
KML	Linguagem de Marcação da *Keyhole, Inc.* (*Keyhole Markup Language*)
LDS	Estrutura Lógica de Dados (*Logical Data Structure*)
LBS	Serviços Baseados em Localização (*Location Based Services*)
LGC	Computação de Localização Geoespacial (*Localized Geospatial Computing*)
LoC	Linhas de Código (*Lines of Code*)
MCC	Complexidade Ciclomática de McCabe (*McCabe's Cyclomatic complexity*)
MOD	Banco de dados de Objetos Móveis (*Moving Object Database*)
OAI	Iniciativa Aberta para API's (*Open API Initiative*)
OGC	Consórcio Aberto Geo Espacial (*OpenGeospatial Consortium*)
OID	IDentificação de Objeto (*Object IDentification*)
OLAP	Processamento Analítico em Tempo Real (*On-Line Analytical Processing*)
OLTP	Processamento de Transações em Tempo Real (*On-Line Transaction Processing*)
ORM	Mapeamento Objeto-Relacional (*Object-Relational Mapping*)

PAML	Linguagem de Marcação para Agricultura de Precisão (*Precision Agriculture Markup Language*)
RAM	Memória de Acesso Aleatório (*Random Access Memory*)
REST	Transferência de Estado Representacional (*REpresentational State Transfer*)
RMM	Modelo de Maturidade de Richardson (*Richardson Maturity Model*)
SDI	Infraestrutura para Dados Espaciais (*Spatial Data Infrastructure*)
SOA	Arquitetura Orientada à Serviços (*Services Oriented Architecture*)
SQALE	Avaliação da Qualidade de Software com base em Expectativas de Ciclo de Vida (*Software Quality Assessment based on Lifecycle Expectations*)
SQL	Linguagem Estruturada de Consulta (*Structured Query Language*)
UML	Linguagem de Modelagem Unificada (*Unified Modeling Language*)
URI	Identificador Uniforme de Recursos (*Uniform Resource Identifier*)
WSDL	Linguagem de Descrição de Serviços Web (*Web Services Description Language*)
XML	Linguagem de Marcação Estendida (*eXtended Markup Language*)
YAML	YAML Não é Linguagem de Marcação (*YAML Ain't Markup Language*)

Lista de símbolos

δ	Variação

Sumário

1

INTRODUÇÃO

Com a popularização dos dispositivos eletrônicos móveis, a quantidade e diversidade de dados gerados e disponíveis para análise têm crescido vertiginosamente. Estes dispositivos, que têm como principal expoente os *smartphones*, podem coletar uma vasta quantidade de informações por meio de sensores embutidos. Dentre elas algumas das que mais se destacam são a geolocalização, a temperatura, a umidade, a velocidade, a direção de movimento e outros atributos que mudam ao longo do tempo (ANDRIENKO; ANDRIENKO, 2012). Segundo relatório da *United Nations International Telecommunication Union*, existe atualmente mais de 7 bilhões de linhas de celular no mundo (SANOU, 2015). Assumindo que apenas metade das linhas de celular (4,5 bilhões) são utilizadas com *smartphones* e que para armazenar a localização e *timestamp*[1] para cada ponto coletado são necessários no mínimo 16 bytes, para gravar um dia de deslocamento destes aparelhos, coletando um ponto por segundo, seriam necessários 4 petabytes de espaço de armazenamento.

A essa grande quantidade e variedade de dados que têm sido gerados, coletados e armazenados dá-se o nome de *Big Data*. De acordo com Zikopoulos et al (2012), o termo *Big Data* aplica-se à informação que não pode ser processada ou analisada uti-

[1] Marca temporal, indica o momento que um evento ocorreu

lizando processos ou ferramentas tradicionais. Recentemente, a empresa Cisco denominou o intervalo do ano 2014 ao ano de 2019 como *Zettabyte Era*, onde o tráfego de dados global na internet ultrapassará a quantidade de um zettabyte[2] e dois terços deste tráfego não será originado dos computadores pessoais, mas sim de dispositivos móveis (CISCO, 2015).

Atualmente, existe um vasta gama de cenários onde a quantidade e complexidade dos dados os classificam como um problema de *Big Data*. Dentre eles, o cenário de processamento e análise de dados de objetos móveis se destaca. Esta situação tem sido evidenciada por pesquisadores que destacam que trajetórias são estruturas espaço-temporais complexas (ANDRIENKO; ANDRIENKO, 2012) onde a sua análise é uma tarefa difícil, se não impossível, quando utilizados sistemas de banco de dados tradicionais centralizados (PELEKIS; THEODORIDIS, 2014). Portanto, para processá-los são necessárias abordagens alternativas que utilizem *software* de execução maciçamente paralela em vários servidores.

No contexto dos objetos móveis, destaca-se o processamento e análise de suas trajetórias. Uma trajetória representa a sequência de pontos registrados durante o caminho do objeto em movimento ao longo do espaço e tempo (ANDRIENKO et al., 2013; BRAZ; BOGORNY, 2012). O processamento destas trajetórias é comumente enquadrado como um tipo de aplicação geoespacial, e este tipo de aplicação é intrinsecamente complexa, pois envolve a coleta e processamento de grandes volumes de dados com frequência contínua (KARIMI, 2014).

Como o estado de um objeto móvel encontra-se em constante atualização, é possível considerar que aplicações de coleta de dados de trajetórias são aplicações de *streamming*. Neste contexto, os modelos tradicionais de bancos de dados não suportam adequadamente estas aplicações. Stonebraker e Cetintemel (2005) enfatizam que a ideia do *"one size fits all"*[3] do modelo

[2] Um zettabyte equivale à 10^{21} bytes.

[3] Expressão comumente utilizada para dizer que um produto serve para todas as situações.

relacional já não é mais aplicável, principalmente devido à geração contínua de dados pelas aplicações. Segundo Braz e Bogorny (2012), sistemas gerenciadores de bancos de dados convencionais e sistemas de informações tradicionais, em geral, não oferecem suporte para a análise e a mineração de dados de trajetórias. Além disso, o modelo relacional foi criado visando a garantia de consistência dos dados em aplicações de processamento de transações em tempo real (OLTP). Entretanto, para aplicações que não são sensíveis a consistência estas propriedades hoje estão sendo negligenciadas a fim de prover maior disponibilidade aos dados por meio da replicação.

Dentre as instituições e organizações que fomentam o uso de dados geoespaciais, o consórcio industrial internacional *Open Geospatial Consortium* (OGC) é uma das que se preocupa com a enorme quantidade de dados de localização que estão sendo coletados todos os dias. Um aspecto chave que a OGC tem relacionado com o *Big Data* é a chamada "fusão de dados". Ela define fusão de dados como sendo "o processo de combinar dados ou informações sobre uma ou mais entidades no intuito de melhorar a capacidade de caracterização delas" (REED, 2014). No domínio das trajetórias de objetos móveis, a fusão de dados pode ser usada, por exemplo, para geração de mapas rodoviários combinando dados de trajetórias de diversos veículos motorizados (COSTA, 2014) ou para a identificação do perfil de direção por meio da análise de dados de aceleração e direção de movimento de vários motoristas (CARBONI, 2014), entre outros.

As tecnologias propostas para resolver os problemas de *Big Data* são enquadradas no âmbito do paradigma denominado NoSQL (VIEIRA et al., 2012; MONIRUZZAMAN; HOSSAIN, 2013). Neste paradigma o armazenamento de dados é menos rígido do que nos bancos de dados relacionais, o que possibilita armazenar dados semi e não estruturados sem a necessidade de modificações no esquema de dados. Além disso, as soluções NoSQL são concebidas para o armazenamento distribuído de dados em *clusters* de centenas ou milhares de servidores, possibilitando assim o armazenamento de zettabytes de dados. Por fim,

estas tecnologias possibilitam a replicação dos dados armazenados o que permite aumentar a sua disponibilidade, facilitando, por consequência, sua acessibilidade (SADALAGE; FOWLER, 2012).

As aplicações que processam dados de trajetórias podem se beneficiar das características providas pelos bancos NoSQL. Dentre as vantagens em utilizá-los, destacam-se a facilidade em adicionar novos dados a medida que novos sensores forem surgindo e a possibilidade de escalar horizontalmente a capacidade de armazenamento aumentando a quantidade de servidores a medida que o volume de dados aumente. Entretanto, estas não são as únicas vantagens no uso dos bancos de dados NoSQL. Dada a variedade de modelos que eles implementam surge a possibilidade de armazenar os dados no formato mais adequado para o seu consumo. Por exemplo, dados coletados por sensores podem ser armazenados como documento, dados usados por ferramentas OLAP para tomada de decisão podem ser armazenados em famílias de colunas e etc. Porém, a utilização de diferentes bancos de dados por uma única aplicação não é considerada uma tarefa trivial. Portanto, para que aplicações possam utilizar bancos de dados de modelos diferentes simultaneamente foi proposto por Leberknight (2008) a técnica chamada de Persistência Poliglota. O autor, inspirado pelo conceito de programação poliglota de Ford (2006), previu que os múltiplos modelos de armazenamento de dados seriam utilizados em conjunto devido aos diferentes problemas a serem resolvidos. Na persistência poliglota, escolhe-se o modelo de dados mais adequado para cada problema a ser resolvido em uma aplicação. Sadalage e Fowler (2012) definem a persistência poliglota como sendo uma abordagem híbrida de persistência de dados.

Um termo frequentemente mencionado no âmbito do processamento de dados espaciais em geral é o *Spatial Data Infrastructure* (SDI). SDI é o conceito usado para designar a coleção de tecnologias, políticas e arranjos institucionais que facilitam a disponibilização e compartilhamento de dados espaciais. Um SDI fornece os mecanismos básicos para a descoberta, avaliação

e aplicação de dados espaciais para usuários e provedores em todos os níveis dos setores governamental, comercial, acadêmico e de cidadãos em geral.

Na construção de uma infraestrutura SDI um conjunto de serviços passíveis de execução coordenada são disponibilizados. Para tal construção, o modelo arquitetural orientado a serviços apresenta-se como a mais adequada e aderente. Marino e Rowley (2009) descrevem a *Service Oriented Architecture* (SOA) como uma arquitetura composta por serviços web altamente interoperáveis e autônomos. Para Erl (2008), SOA representa uma arquitetura aberta, facilmente extensível, federada, combinável e composta de serviços autômatos, interoperáveis, detectáveis, reutilizáveis implementados por meio de *web services*. Erl (2008) também menciona que os principais benefícios desta arquitetura estão associados com a padronização, consistência, confiabilidade e escalabilidade estabelecidas por meio da aplicação de princípios de projetos orientados a serviços.

Recentemente uma nova tendência tem-se apresentado na área de sistemas que utilizam a arquitetura SOA, a de microsserviços. Nesta nova abordagem são utilizados um conjunto de pequenos serviços, cada um executando em seu próprio processo e que se comunicam por meio de mecanismos mais leves (LEWIS; FOWLER, 2014). Como cada microsserviço executa em seu próprio processo, a distribuição destes pelos diversos nós de uma rede, chamada de escalabilidade horizontal, torna-se bem mais simples. É relevante observar que alguns autores já vem considerando microsserviços um retorno às origens da arquitetura SOA, sem o *overhead* gerado pelos fornecedores de plataformas SOA (NEWMAN, 2015; LEWIS; FOWLER, 2014). Grandes companhias da internet como o Netflix tem-se utilizado deste modelo de forma a escalar e evoluir a sua plataforma de *streaming* (MAURO, 2015).

No âmbito da análise de trajetórias de objetos móveis, a quantidade de processos utilizados é elevada e grande parte deles se repetem em diferentes aplicações. Por exemplo, são frequentes as aplicações de processos de limpeza e compactação dentro da

fase de pré-processamento realizada antes da aplicação dos processos de análise propriamente ditos. Além disso, também são realizadas em vários tipos de aplicações processos de clusterização e identificação do meio de transporte. Por fim, não são poucos os casos onde são necessárias implementações de APIs (*Application programming interfaces*) para realizar a persistência das trajetórias brutas, assim como dos resultados intermediários e final dos processos de análise de trajetórias. Como pode ser observado, na análise de trajetórias há uma frequente reutilização de processos e algoritmos que precisa ser facilitada e onde uma arquiteturas SDI orientada a serviços pode auxiliar na reutilização.

Como pode-se perceber são vários os benefícios que podem ser obtidos pelo uso de modelos NoSQL e arquiteturas SDI no âmbito da análise de trajetórias de objetos móveis. Esses benefícios são aumentados quando da utilização de múltiplos modelos, na chamada persistência poliglota. Entretanto, estes benefícios ainda são pouco explorados no armazenamento e processamento de trajetórias. Parte desta ausência de trabalhos é explicada pela complexidade associada à quebra de paradigma que é utilizar estes diferentes modelos. Esta complexidade é ainda maior quando da utilização mútua destes modelos pela mesma aplicação. Portanto, o presente trabalho assume como hipótese que prover um arcabouço estrutural na forma de um SDI para implantação de serviços de processamento de trajetórias pode abstrair parte da complexidade do uso da persistência poliglota e assim aumentar a produtividade no desenvolvimento de aplicações que usem modelos NoSQL para este domínio. Com a proposição deste SDI espera-se que diferentes tipos de dados sobre trajetórias possam ser armazenados e acessados em larga escala e sem a necessidade de mapeamentos entre modelos.

1.1 Objetivos

Nas subseções a seguir são apresentados o objetivo geral e os objetivos específicos deste trabalho.

1.1.1 Geral

Este trabalho tem por objetivo prover uma arquitetura SDI orientada a serviços que sirva como base para a criação e implantação de serviços para aplicações que processem trajetórias de objetos móveis. Ela deve facilitar a reutilização e escalabilidade dos processos e prover uma camada de persistência poliglota que possibilite o armazenamento de dados semi e não estruturados associados às trajetórias.

1.1.2 Específicos

Com base no objetivo geral, tem-se os seguintes objetivos específicos:

- Projetar um repositório de persistência poliglota que armazene as trajetórias dos objetos móveis, primando pela alta disponibilidade dos dados armazenados.

- Criar um mecanismo para automatizar a persistência poliglota, sendo que ele deve prover transparência quanto aos modelos de dados utilizados pela arquitetura.

- Criar uma arquitetura SDI orientada a serviços buscando padrões que facilitem o reuso e a escalabilidade/elasticidade dela.

1.2 Resultados Esperados

Como resultado espera-se obter um arcabouço computacional que suporte a implantação de serviços de processamento de trajetórias e que facilite o reuso dos serviços existentes. Além disso, este arcabouço deve suportar o armazenamento de trajetórias com dados heterogêneos e o processamento de grandes volumes de dados de trajetórias.

1.3 Metodologia

A seção 1.3.1 apresenta a caracterização metodológica deste trabalho. Já a seção 1.3.2 apresenta a metodologia de pesquisa utilizada.

1.3.1 Caracterização metodológica

A caracterização metodológica de um trabalho científico é importante para enquadrar o objetivo e prover ao pesquisador instrumentos científicos conhecidos que o ajudem a alcançar tal objetivo. Neste sentido, a seguir é apresentada a caracterização do presente trabalho quanto a diversas classificações encontradas na literatura.

Em relação ao tipo de ciência (WAZLAWICK, 2014; HEDGES, 1987), o presente trabalho pode ser classificado como ciência exata e *soft*. Exata pois não há variação nos resultados quando utilizado como entrada o mesmo conjunto de dados, independentemente da quantidade de execuções. *Soft* pois usa evidências de execuções de estudos de caso sem a utilização de formalismos matemáticos.

Em relação ao paradigma científico dominante (EDEN, 2007), o presente trabalho pode ser classificado dominantemente dentro do paradigma tecnocrático, pois a eficiência dos sistemas e algoritmos desenvolvidos só é conhecida após a realização de testes.

Em relação ao raciocínio lógico (LAKATOS; MARCONI, 2010), o presente trabalho utiliza o método indutivo, pois o desenvolvimento do arcabouço é baseado na condução de testes que visam evidenciar se a arquitetura atende aos requisitos levantados, e induz-se que resultados com a mesma qualidade podem ser obtidos em cenários semelhantes.

Finalmente, em relação ao nível de maturidade da pesquisa (WAZLAWICK, 2014), o presente trabalho pode ser classificado no estilo “apresentação de algo diferente”, pois utiliza uma combinação de técnicas e abordagens existentes a fim de apresentar uma contribuição nova à área de estudos baseado em

argumentações teóricas e pesquisas bibliográficas.

1.3.2 Metodologia de pesquisa

Conforme definido na literatura, uma pesquisa científica deve ser baseada em um conjunto de procedimentos executados rigorosamente de forma a produzir algum conhecimento novo (FILIPPO; PIMENTEL; WAINER, 2011). Portanto, para que o avanço ocorra na pesquisa é necessário que o pesquisador planeje e execute determinados passos.

Neste trabalho, o procedimento metodológico adotado abrange o seguinte conjunto de passos. Primeiramente foi feita uma extensa revisão da literatura para identificar os principais requisitos para os sistemas de armazenamento e processamento de grandes volumes de dados de trajetórias de objetos móveis. Após a identificação dos requisitos, iniciou-se o projeto e desenvolvimento da solução.

Paralelamente ao projeto, foram estudados os principais conceitos utilizados no desenvolvimento do trabalho. Também foram realizados testes de sistemas gerenciadores de bancos de dados e ferramentas de desenvolvimento e implantação de microsserviços com o intuito de definir quais melhor atenderiam aos requisitos levantados anteriormente. A realização destes testes possibilitou a proposição de uma solução para a questão de pesquisa e a identificação dos gerenciadores de bancos de dados possibilitou o início do desenvolvimento de um protótipo para validar a solução.

Além da análise das ferramentas, buscou-se *datasets* públicos de trajetórias que pudessem ser utilizados na análise do esquema de dados proposto. Foram selecionados preferencialmente *datasets* usados em pesquisa científica a fim de avaliar a aderência do esquema de dados proposto com as demais pesquisas que são realizadas pela comunidade científica.

Para o desenvolvimento da solução foi necessário:

1. Projetar o esquema conceitual de objetos para os dados de

trajetórias;

2. Projetar o esquema de dados para cada um dos modelos NoSQL por meio do mapeamento do esquema conceitual concebido no passo 1;

3. Projetar a arquitetura da solução em UML;

4. Criar os esquemas de dados nos bancos de dados selecionados;

5. Implementar os serviços de suporte à arquitetura projetados no passo 3;

6. Identificar processos e implementar os algoritmos de processamento de dados de trajetória utilizados na avaliação da arquitetura;

7. Importar os dados dos *datasets* para cada um dos bancos de dados selecionados;

8. Executar experimentos com os componentes de processamento implementados no passo 6 a fim de avaliar a solução.

Para avaliar a solução foram realizados experimentos sobre a arquitetura desenvolvida e sobre os esquemas de dados projetados a fim de avaliar se tanto a arquitetura quanto os esquemas de dados suportavam o armazenamento de dados de trajetórias com dados heterogêneos e o processamento de grandes volumes de dados de trajetórias.

1.4 Estrutura do Trabalho

O restante do texto está organizado da seguinte forma. O Capítulo 2 descreve os conceitos relacionados ao trabalho onde os mais relevantes são banco de dados NoSQL, *Big Data*, arquitetura orientada a serviços e microsserviços. O Capítulo 3 detalha e compara os principais trabalhos relacionados. O Capítulo 4

descreve o esquema de dados e a arquitetura propostos para armazenamento e processamento de trajetórias e suas informações relacionadas. O Capítulo 5 apresenta os experimentos realizados e a análise dos resultados. Por fim, o Capítulo 6 apresenta as conclusões, contribuições e trabalhos futuros.

2

CONCEITOS

Este capítulo apresenta os principais aspectos relacionados aos dois conceitos abordados por este trabalho, sendo eles NoSQL e arquiteturas orientadas a serviços. A seção 2.1 a seguir, apresenta os temos relacionados a trajetórias, a seção 2.2 apresenta os temas relacionados a NoSQL, a seção 2.3 apresenta os temas relacionados às arquiteturas orientadas a serviços e a seção 2.4 apresenta as consideração.

2.1 Trajetórias de Objetos Móveis

Segundo Fileto et al (2015), uma trajetória é uma sequência temporal ordenada de posições espaço-temporais ocupados por um objeto em movimento. Segundo Braz e Bogorny (2012), trajetória é a sequência de pontos registrados por um objeto móvel. Os autores também definem que objeto móvel é qualquer objeto que carregue um dispositivo que armazene os dados de coordenadas geográficas num instante de tempo.

Já Boulmakoul, Karim e Lbath (2012), definem os estados da trajetória, onde uma trajetória bruta é a gravação das posições de um objeto em um espaço-tempo específico; uma trajetória estruturada é uma trajetória bruta separada em segmentos que correspondem em passos significativos no traço da trajetória;

e uma trajetória semântica expressa o significado desta usando quatro componentes (Parar, avançar, começar e terminar).

De acordo com Parent et al (2013), uma trajetória bruta é minimamente definida pela seguinte tupla:

(IDTraj, IDObj, rastro: Lista De posição(instante, ponto, δ)),

onde IDTraj é a identificação da trajetória, IDObj é a identificação do objeto móvel e δ denota uma possível lista de dados brutos adicionais como, por exemplo, velocidade e direção.

2.2 NoSQL

O termo NoSQL está relacionado ao conjunto de tecnologias que surgiram predominantemente em respostas aos problemas advindos do aumento exponencial da geração de dados presenciada nos últimos anos, denominada como *Big Data*. Esta seção apresenta os principais temas relacionados ao termo NoSQL, assim como suas principais características e modelos de dados.

2.2.1 *Big Data*

Big Data é um termo que descreve o armazenamento e análise de grandes e/ou complexos conjuntos de dados utilizando uma série de técnicas que incluem, mas não estão limitadas ao, uso de NoSQL, *Map-Reduce* e aprendizado de máquina (WARD; BARKER, 2013).

A *International Telecommunication Union*, agência das Nações Unidas, define *Big Data* como sendo um paradigma para habilitar a coleta, armazenamento, gerenciamento, análise e visualização, potencialmente sob restrições de tempo real, de extensos conjuntos de dados com características heterogêneas (ITU, 2015).

De acordo com Hurwitz et al (2013), para ser considerado um problema de *Big Data*, a fonte de dados tem que apresentar, pelo menos, as seguintes características:

- Grande **Volume** de dados. Para ser considerado um grande volume de dados a fonte deve produzir dados que superem a ordem dos *petabytes*[1].

- Grande **Velocidade** dos dados. A velocidade está associada à produção contínua de dados na forma de *streaming*. Esta característica é comumente encontrada em sensores embutidos em equipamentos eletrônicos.

- Ampla **Variedade** de dados. A variedade diz respeito ao armazenamento de dados semi e não estruturados de forma natural, assim como já é realizada com dados estruturados. Como exemplo de dados não estruturados tem-se imagens, áudios, vídeos, entre outros.

2.2.2 Principais características do NoSQL

O NoSQL está relacionado à geração de bancos de dados que abordam pontos tais como: (i) modelos não-relacional, (ii) arquitetura distribuída, (iii) *open-source* e (iv) escalabilidade horizontal (EDLICH, 2015).

Segundo Cattell (2011), para ser considerado NoSQL os bancos de dados devem possuir as seguintes características:

1. Capacidade de escalar horizontalmente uma simples operação em muitos servidores;

2. Capacidade de replicar e distribuir os dados ao longo de muitos servidores (partições);

3. Ter um protocolo ou interface simples de comunicação, em contraste ao fornecido pela SQL;

4. Um modelo de concorrência entre transações mais fraca do que o estabelecido pela ACID;

[1] Um petabyte equivale a 10^{15} bytes.

5. Utilização eficiente dos índices e memória RAM para o armazenamento de dados distribuídos;

6. Capacidade de adicionar novos atributos dinamicamente nos registros de dados.

O quadro 1 apresenta os quatro principais fatores de distinção entre os sistemas de gerenciamento de banco de dados relacionais tradicionais e os NoSQL.

Quadro 1 – Fatores de distinção entre Relacional e NoSQL

Fator	**Relacional**	**NoSQL**
Variedade	um tipo	quatro tipos
Estrutura	pré-definida	dinâmica
Escalabilidade	vertical	horizontal
Foco	integridade de dados	performance e disponibilidade

Fonte: Hoberman, 2014.

2.2.3 Teorema CAP

O teorema CAP proposto por Brewer estabelece que qualquer sistema de compartilhamento de dados em rede pode ter no máximo duas das seguintes três propriedades: consistência, disponibilidade e tolerância à particionamento. A alta disponibilidade é alcançada com a replicação e distribuição de dados, entretanto a consistência é afetada pelo particionamento de rede (BREWER, 2012; BREWER, 2000).

Gilbert e Lynch (2002), comprovaram o teorema de Brewer e afirmaram que é impossível no modelo assíncrono de rede implementar uma leitura e gravação de objetos de dados que garanta as propriedades de disponibilidade e consistência atômica em todas as execuções justas.

Desde que o princípio de transação foi definido por Gray (1981), os bancos de dados relacionais tradicionais buscam fornecer as propriedades ACID, que garantem principalmente a con-

sistência dos dados. Elmasri e Navathe (2011) as conceituam da seguinte forma:

Atomicidade
Uma transação é uma unidade de processamento atômica que deve ser realizada na sua totalidade ou não ser executada de forma alguma.

Consistência
Uma transação deve preservar a consistência levando o banco de dados de um estado consistente para outro estado também consistente.

Isolamento
A execução de uma transação não deve ser interferida por quaisquer outras transações que aconteçam simultaneamente.

Durabilidade
As mudanças aplicadas ao banco de dados pela transação confirmada (*committed*) precisam persistir no banco de dados e não devem ser perdidas em caso de alguma falha.

Por natureza, o teorema CAP não atende as restrições impostas pelas propriedades ACID. Por este motivo, bancos de dados que priorizam a disponibilidade por meio da distribuição e replicação, como é o caso dos NoSQL, seguem outro conjunto de propriedades chamado BASE. Como princípio, o BASE aceita que o estado dos dados esteja inconsistente por um período de tempo, ampliando a disponibilidade deles para as aplicações. Esta característica é chamada de consistência eventual (PRITCHETT, 2008). As propriedades BASE estabelecem as seguintes características:

Disponibilidade básica
Haverá uma resposta a qualquer requisição. O armazenamento parece funcionar na maior parte do tempo.

Estado relaxado
Armazenamentos não têm que ter uma escrita consistente, nem as réplicas têm de ser coerentes entre si o tempo todo.

Consistência eventual
O sistema, eventualmente, ficará consistente após parar de receber as entradas de dados.

2.2.4 *Schemaless*

Grande parte dos bancos de dados NoSQL proveem modelos de dados que não necessitam do projeto prévio de um esquema de dados, aumentando assim a flexibilidade no armazenamento e a agilidade no desenvolvimento e manutenção de aplicações.

McCreary e Kelly (2013) definem agilidade como sendo a capacidade do *software* se adaptar rapidamente às mudanças de requisitos de negócios, e a flexibilidade do esquema de dados como uma das características que contribuem consideravelmente no atendimento deste requisito.

Por natureza, aplicações que seguem princípios básicos de engenharia de *software* em sua implementação apresentam o projeto de uma camada de dados. As aplicações que utilizam um banco de dados *schemaless* podem utilizar o projeto desta camada de dados da aplicação como um esquema implícito para o banco de dados, não necessitando assim duplicar regras entre aplicação e banco de dados (SADALAGE; FOWLER, 2012).

2.2.5 Modelos de Dados NoSQL

Os bancos de dados NoSQL são basicamente classificados de acordo com o modelo de dados utilizados. Os principais modelos são: chave-valor, documento, família de colunas e grafos. A seguir, cada um destes modelos são detalhados.

Chave-Valor
O modelo de dados chave-valor é uma simples tabela *hash*, onde todo o acesso ao valor é feito por meio da chave, pois o valor é considerado opaco, ou seja, não é conhecido pelo banco de dados (SADALAGE; FOWLER, 2012).

Documento
Os bancos de dados que utilizam este modelo armazenam e

recuperam documentos que podem estar em formato XML, JSON, BSON, entre outros. Os dados no modelo documentos são auto-descritivos e estruturados de forma hierárquica (SADALAGE; FOWLER, 2012).

Família de Colunas

Este modelo armazena dados com chaves mapeadas para valores e os valores agrupados em várias famílias de colunas, sendo cada família de coluna um mapa de dados. Famílias de colunas são grupos de dados relacionados que são frequentemente acessados juntos (SADALAGE; FOWLER, 2012).

Grafos

O modelo grafo é formado por um conjunto de vértices e arestas que ligam os vértices entre si, representando seus relacionamentos. Tanto os vértices quanto as arestas podem ter um tipo e propriedades na forma de pares chave-valor (ROBINSON; WEBBER; EIFREM, 2015).

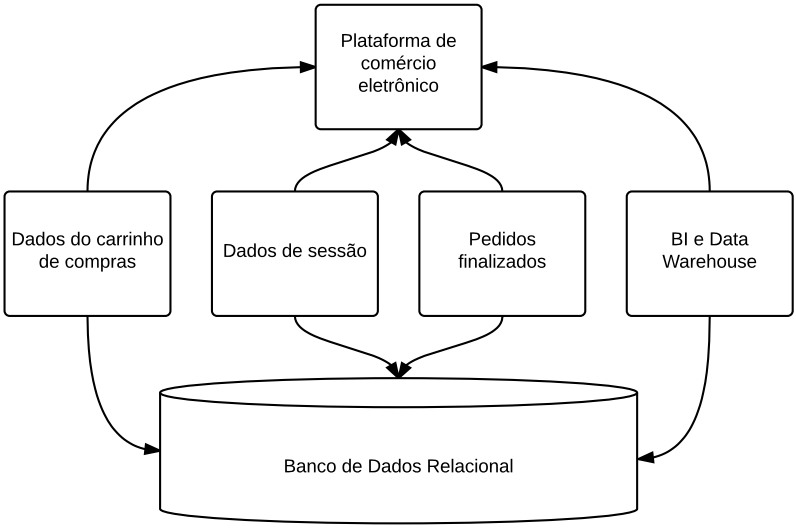
2.2.6 Persistência Poliglota

Persistência Poliglota é o termo criado por Leberknight (2008) que significa escolher o modelo correto de persistência para cada tarefa a ser efetuada. Considerando que diferentes modelos de dados são projetados para resolver diferentes problemas, a persistência poliglota significa usar diferentes modelos de dados em diferentes circunstâncias, não sendo o modelo relacional a única opção (SADALAGE; FOWLER, 2012).

Na figura 1 é possível observar o exemplo de uma plataforma de comércio eletrônico usando a estratégia convencional de armazenamento onde o modelo relacional é o único utilizado. Em contraste, na figura 2, onde a persistência poliglota é utilizada, cada uma das funcionalidades da plataforma utiliza o modelo mais adequado para persistir seus dados. Um dos exemplos apresentados na figura 2 é a persistência dos pedidos completos (realizados), por ser um tipo de informação que não sofre atualização e que comumente precisa ser enviada para órgãos de controle

fiscal e tributário, ela é melhor armazenada como um arquivo em formato XML ou JSON em um banco de dados orientado a documentos.

Figura 1 – Modelo de persistência convencional



Fonte: Sadalage e Fowler, 2012.

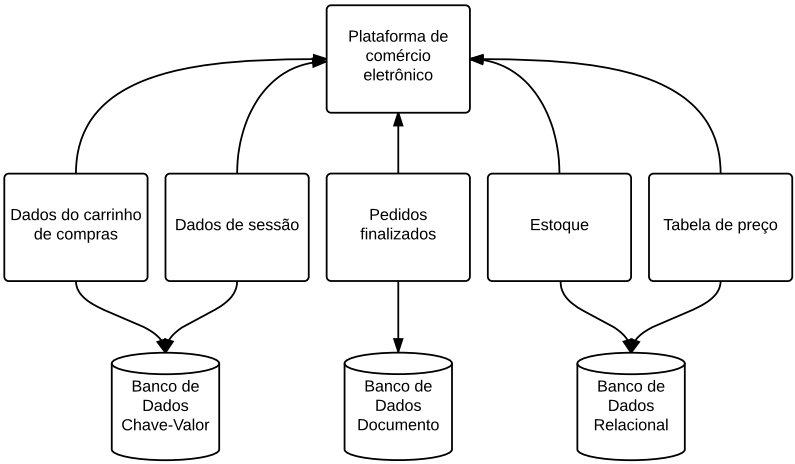
2.3 Arquitetura orientada a serviços

SOA é a denominação dada para representar uma arquitetura aberta, facilmente extensível, federada, combinável e composta de serviços autômatos, interoperáveis, detectáveis, reutilizáveis implementados por meio de serviços web (ERL, 2008). O *World Wide Web Consortium* (W3C) descreve serviço web como sendo um componente de *software* projetado para suportar interações máquina-a-máquina interoperáveis em uma rede (W3C, 2004).

A interação entre os serviços web pode ser coordenada de duas formas, sendo elas a orquestração e a coreografia. O termo orquestração se refere à coordenação de vários serviços por meio

de um mediador centralizado, como um consumidor de serviços ou um *hub* de integração (RICHARDS, 2015).

Figura 2 – Modelo de persistência poliglota



Fonte: Sadalage e Fowler, 2012.

A coreografia diz respeito à coordenação de múltiplas chamadas de serviços sem um mediador central. O termo comunicação inter-serviços é por vezes utilizado em conjunto com coreografia de serviços. Com o uso de coreografia um serviço chama outro serviço, sendo que esse também pode chamar outro serviço e assim por diante, realizando o que também é conhecido como encadeamento de serviços (RICHARDS, 2015).

2.3.1 REST

Conforme definido por Fielding (2000), REST é um estilo arquitetural onde a transferência de estados representacionais ocorre sobre o protocolo HTTP. Nele a arquitetura SOA é utilizada de uma maneira mais simples, restringindo-se aos métodos existentes no protocolo HTTP, como GET e POST, e é utilizada a estrutura dos endereços URL para o acesso aos serviços (BURKE, 2009).

De acordo com Erl et al (2012) o estilo REST possui um contrato uniforme composto por três elementos fundamentais:

1. Sintaxe de identificação de recurso
Identificadores de recursos representam os recursos reais que um serviço expõe. Os recursos podem ser dados, lógica de processamento, arquivos ou qualquer outra coisa que um serviço pode ter acesso.
esquema://autoridade/caminho?pesquisa#fragmento
ex: http://pedido.site.com/pedidos?menor-que=100#pag2

2. Métodos
Um método é um tipo de função que é fornecida por um contrato uniforme para processar dados e identificadores de recursos.

3. Tipos de mídia
Ao se definir os métodos para um serviço REST, pode-se especificar os tipos de dados que um determinado método pode processar.

2.3.1.1 Modelo de Maturidade de Richardson (RMM)

Richardson (2008) propôs uma classificação para os serviços web que executam com chamadas REST. Esta classificação possui três níveis de maturidade dependendo das características suportadas pelo serviço, mais um quarto nível caso o serviço não possua suporte algum (WEBBER; PARASTATIDIS; ROBINSON, 2010; FOWLER, 2010). São eles:

Nível 0 O nível mais básico de maturidade caracteriza aqueles serviços que têm uma única URI e que usam apenas um método HTTP, sendo este normalmente o POST.

Nível 1 Este nível é aplicado aos serviços que utilizam várias URI's, mas somente um único método HTTP, sendo este normalmente o GET.

Nível 2 Este nível é aplicado aos serviços que utilizam várias URI's e vários métodos e códigos de retorno HTTP.

Nível 3 Este nível é aplicado aos serviços que, além de abranger o nível 2, fornecem suporte a HATEOAS, onde a aplicação utiliza *hypermedia* para representação do seu estado.

2.3.2 Microsserviços

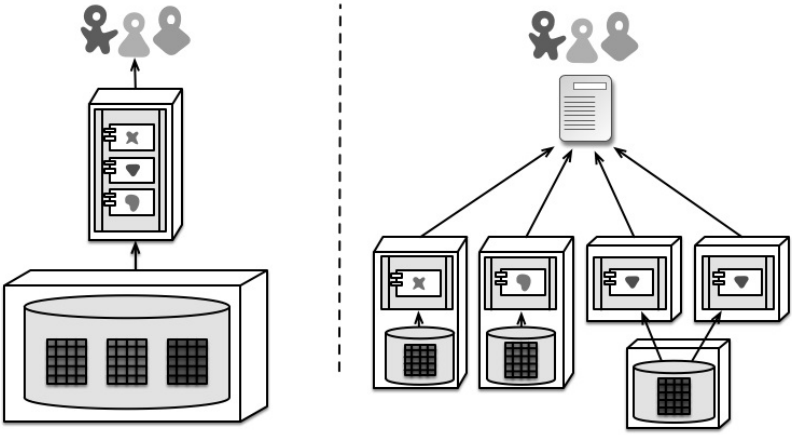
De acordo com Newman (2015), microsserviços são pequenos serviços autônomos que trabalham em conjunto e que possuem os seguintes princípios:

- Modelagem em torno da divisão de responsabilidades. Cada serviço é responsável por implementar um artefato completo e autocontido da regra de negócios da aplicação.
- Adoção da cultura de automação. Os processos de teste e implantação de cada serviço devem ser facilmente automatizados.
- Esconder os detalhes internos de implementação. Apesar de esconder a implementação, eles devem fornecer uma interface de acesso aos métodos por eles disponibilizados.
- Descentralização. Cada serviço pode ser desenvolvido por uma equipe diferente e distinta.
- Implantação independente. Cada serviço possui a habilidade de ser implantado e evoluído independentemente dos demais serviços.
- Isolamento de falhas. Com a independência dos serviços, a resiliência geral do projeto aumenta consideravelmente, pois se um serviço falha os demais não devem ser afetados.
- Altamente observável. Cada serviço fornece informações sobre a sua saúde de execução, permitindo o monitoramento do sistema.

É possível observar que os princípios expostos acima são basicamente o retorno aos conceitos originais da arquitetura SOA. Esta nova proposta procura remover as camadas e *overheads* introduzidas durantes os anos pelos vendedores comerciais de soluções SOA, retornando ao princípio da responsabilidade única de cada serviço independente (NEWMAN, 2015; LEWIS; FOWLER, 2014; LEWIS, 2012; MARTIN, 2003).

A Figura 3 resume a diferença entre o modelo SOA considerado monolítico e o modelo de microsserviços onde ao invés de haver um grande contêiner que mantém os diversos serviços, estes são executados como processos que podem ser distribuídos em diferentes dispositivos. Lewis e Fowler (2014) destacam ainda que o uso de microsserviços reduz a complexidade da utilização de persistência poliglota, pois permite que cada serviço administre o seu próprio banco de dados.

Figura 3 – Diferença entre os modelos monolítico e de microsserviços em SOA



Fonte: Lewis e Fowler, 2014.

2.3.3 Descoberta e monitoramento de serviços

Quando se fala de serviços disponíveis para o consumo de sistemas clientes dois problemas que precisam ser tratados são a

localização e a disponibilidade do serviço.

Sobre a localização, a internet possui servidores de DNS que são incumbidos de traduzirem nomes de domínio para endereços IP dos respectivos *sites* que estão sendo acessados. No contexto dos serviços web, infelizmente o serviço DNS resolve apenas parte dos problemas, pois, apesar de efetuar a tradução, não informa se o *site* ou serviço em questão está acessível ou não. Descobrir esta informação acaba se tornando uma responsabilidade da aplicação cliente.

Para resolver completamente o problema de descoberta e monitoramento, faz-se necessário um serviço que forneça uma espécie de catálogo de serviços disponíveis com seus respectivos endereços de rede e que avalie periodicamente a disponibilidade destes serviços.

De acordo com Richardson (2015), existem basicamente dois padrões de projeto que abordam a descoberta de serviços:

Client-side service discovery
: Ao fazer um pedido para um serviço, o cliente obtém a localização de uma instância de serviço consultando um *service registry*, que conhece a localização de todas as instâncias de serviços.

Server-side service discovery
: Ao fazer um pedido para um serviço, o cliente faz um pedido por meio de um roteador ou balanceador de carga que é executado em um local bem conhecido. O roteador consulta um *service registry*, que pode ser embutido nele, e encaminha a solicitação para uma instância disponível do serviço.

2.4 Considerações sobre os conceitos revisados

Pela revisão dos conceitos relacionados ao trabalho é possível observar a relevância do tema NoSQL no armazenamento de dados estruturados e não estruturados e no processamento de grandes massas de dados. Essas características atendem aos

requisitos oriundos do problema de *Big Data* que não eram suportados apenas pelo modelo relacional tradicional.

Também é possível observar a revitalização dos conceitos associados a SOA com a proposição do padrão de microsserviços. Ele auxilia na flexibilidade e suporte a múltiplos modelos de dados por meio do desacoplamento de um grande gerenciador monolítico.

Estes conceitos formam a base da arquitetura proposta no capítulo 4 na página 65.

3

TRABALHOS RELACIONADOS

A literatura apresenta trabalhos que abordam o armazenamento e manipulação de trajetórias e outros que proveem infraestruturas orientada a serviços para o processamento de dados georeferenciados. Dentre eles, os mais relevantes em relação ao referente trabalho são revisados a seguir.

3.1 *An infrastructure for the development of distributed service-oriented information systems for precision agriculture*

Murakami et al. (2007) apresentam um *framework* baseado em serviços web que provê processos aplicados ao contexto da agricultura de precisão. Para a concepção deste *framework* os autores realizaram um abrangente levantamento de requisitos, cobrindo desde as necessidades específicas dos usuários (agricultores) até a usabilidade da interface gráfica. Como resultado, os autores apresentam uma arquitetura de referência para execução de serviços com comunicação via *Enterprise Service Bus* (ESB), e um conjunto de serviços web focados no processamento de dados agrícolas. Todas as mensagens trocadas entre os processos foram padronizadas utilizando a linguagem de marcação para agricultura de precisão chamada PAML.

Sobre o trabalho de Murakami et al. (2007) vale destacar a forma como eles propuseram o *framework*. Apesar de ser aparentemente uma solução tecnológica, dada a profundidade e a abrangência da solução apresentada, o *framework* por eles proposto pode ser considerado como uma contribuição técnico/científica, servindo portanto como base para o desenvolvimento de trabalhos científicos de natureza semelhante.

3.2 *A distributed geospatial data storage and processing framework for large-scale WebGIS*

Zhong et al. (2012) apresentam um *framework* GIS voltado à web chamado WebGIS que tem por objetivo prover armazenamento de dados geoespaciais de forma distribuída. Na proposta, os autores optaram por utilizar um banco de dados colunar e um sistema de arquivos distribuído em lugar de um centralizado devido aos conhecidos problemas de escalabilidade encontrados no segundo.

Como resultado, os autores conceberam um novo banco de dados colunar chamado de VegaCI que tem por intuito melhorar o desempenho do processamento de dados espaciais. A implementação da solução foi efetuada sobre a ferramenta Apache Hadoop[1], que proveu o sistema de arquivos distribuído (HDFS) utilizado.

Para avaliar a proposta foram efetuados testes comparativos entre *clusters* do protótipo desenvolvido (VegaCI) e de soluções similares com modelo relacional, tais como: MySQL, PostGIS e Oracle Spatial. De acordo com os autores, a melhora no tempo de resposta foi de 69,1% a 83,7% em relação aos SGBDs tradicional e a eficiência geral medida melhorou de 57% a 153%.

O trabalho de Zhong et al. (2012) destaca a eficiência do modelo NoSQL sobre o modelo relacional no contexto do processamento de grande quantidade de dados e usuários por um sistema GIS.

[1] <http://hadoop.apache.org>

3.3 *Implementing geospatial web services using service oriented architecture and NoSQL solutions*

Amirian et al. (2013) apresentam o projeto de uma arquitetura orientada a serviços com armazenamento NoSQL voltada para aplicações geoespaciais. Segundo os autores, a escolha de uma arquitetura SOA visa permitir uma maior interoperabilidade e o uso de modelos NoSQL facilita a manipulação de dados semi e não estruturados.

A arquitetura proposta contém vários componentes utilizados para analisar as requisições dos usuários, prover acesso aos dados espaciais e a geração de mapas. Estes componentes são acessados por meio de uma camada de fachada que recebe as requisições e encaminha ao componente responsável pelo processamento correspondente.

Assim como em trabalhos anteriores dos mesmos autores, este estudo foca na utilização de dados em XML armazenados nativamente em bancos de dados que suportam este formato (AMIRIAN; ALESHEIKH, 2008a; AMIRIAN; ALESHEIKH, 2008b). Apesar das importantes contribuições, este trabalho restringe-se ao uso de dados em formato XML, o que se apresenta como uma limitação nos dias atuais.

3.4 Um modelo de dados para trajetórias de objetos móveis com suporte a agregação de movimentos

Almeida et al. (2012) apresentam a proposta de um esquema conceitual de *Data Warehouse* para armazenar trajetórias agregando as semânticas associadas às informações geográficas. Levando em consideração as dimensões geométrica e semântica dos dados é possível obter uma trajetória espaço-temporal semântica para descobrir as origens e destinos mais comuns, as rotas mais utilizadas, as velocidades médias em cada trecho, entre outras. Portanto, o esquema proposto foi concebido visando principalmente as pesquisas voltadas ao estudo de movimentação urbana.

O trabalho de Almeida et al. (2012), apesar de padronizar o armazenamento das trajetórias, propôs um esquema que enfatiza os movimentos e paradas ao invés da trajetória propriamente dita. Este foco difere do presente trabalho que apresenta um esquema voltado ao armazenamento das trajetórias e sua semântica de forma genérica e preservando o histórico de processamento da trajetória por meio do armazenamento de seus vários estados.

3.5 *BigGIS: how big data can shape next-generation GIS*

Yue e Jiang (2014) evidenciam o surgimento de uma nova geração de sistemas GIS por eles denominada de *BigGIS*. De acordo com os autores, *BigGIS* é o conceito que abrange o desenvolvimento de sistemas GIS na era do *Big Data* voltado para detectar, armazenar, integrar, processar e visualizar grandes quantidades de dados geoespaciais. O *BigGIS* engloba questões conceituais, metodológicas, técnicas e gerenciais no intuito de resolver os desafios oriundos do *Big Data*.

Os autores apresentam cinco características chaves que os sistemas precisam atender para suportarem o *BigGIS*: (i) observação coordenada de sensores heterogêneos, (ii) esquema universal de dados geoespaciais, (iii) armazenamento de dados de forma distribuída, (iv) indexação espaço-temporal e (v) um *framework* para computação paralela.

Observando o trabalho de Yue e Jiang (2014) é possível perceber que, além de enquadrarem os GIS como um problema de *Big Data*, os autores identificaram as necessidades a serem atendidas no projeto de sistemas GIS dado o crescente volume de dados georeferenciados gerados.

3.6 *A spatial data model for moving object databases*

Hajari e Hakimpour (2014) definiram um modelo de dados para a criação de *Moving Object Databases* (MOD). Os au-

tores procuraram preencher as lacunas dos sistemas disponíveis para aplicações geoespaciais que utilizam preferencialmente modelos geométricos e com expressões algébricas.

Os principais pontos que os autores consideraram como deficientes nas implementações tradicionais são o livre movimento e a durabilidade do movimento. Os autores consideraram que objetos móveis estão restritos numa rede de transporte ao invés de possuir livre movimento e consideraram que o movimento dos objetos pode ser durável e contínuo.

Como solução, os autores apresentam dois diagramas UML, um com as classes das entidades da rede de transportes e outro diagrama com as classes dos objetos móveis. Os autores também apresentam os comandos de definição das tabelas (DDL) no banco de dados relacional.

Assim como outros trabalhos já revisados, o trabalho de Hajari e Hakimpour (2014) identifica a necessidade de um esquema de dados para o armazenamento de objetos móveis. Entretanto, ele se difere do presente trabalho na flexibilização dos dados adicionais. Enquanto eles propõem atributos como velocidade, aceleração e lado da via, o presente trabalho se utiliza de entidades extras com o objetivo de não limitar os dados adicionais.

3.7 *Towards scalable distributed framework for urban congestion traffic patterns warehousing*

Boulmakoul et al. (2015) apresentam um modelo conceitual que armazena juntamente ao dados oriundos de GPS sua semântica associada ao contexto de computação urbana. De acordo com os autores, essa fusão de dados de geolocalização e semântica urbana pode levar a previsão de prováveis congestionamentos por meio da análise de dados históricos. Os autores optaram pela utilização de armazenamento em banco de dados NoSQL usando modelo documento pelas vantagens da ausência de esquemas e do armazenamento de dados em formato JSON.

3.8 Considerações sobre os trabalhos relacionados

Pela análise da literatura resumida neste capítulo, pode-se observar que a maioria dos trabalhos abordam apenas parte do problema atacado pelo trabalho em tela. O Quadro 2 apresenta quais assuntos são abordados em cada trabalho, e nele é possível observar que os que mais se assemelham ao conduzido aqui são os produzidos por Amirian et al. (2008a, 2008b, 2013), pois envolvem tanto processamento quanto persistência utilizando uma arquitetura de serviços. Entretanto, mesmo assim, eles não fazem uso das tecnologias mais recentes de armazenamento NoSQL. Portanto, para poder analisar com mais profundidade os trabalhos relacionados é necessário classificá-los de acordo com os seguintes aspectos: (i) Aqueles que tratam de armazenamento de trajetórias em bancos de dados NoSQL; (ii) Aqueles que tratam do uso de arquiteturas orientadas a serviço para o processamento de dados geoespaciais.

Sobre os trabalhos que tratam do armazenamento em NoSQL, é consenso entre os autores dos trabalhos revisados que os bancos de dados centralizados não conseguem acompanhar o crescente aumento no volume e na diversidade de dados coletados. Portanto, abordagens distribuídas e de estrutura flexível são a solução (ALMEIDA; PIRES; SCHIEL, 2012; YUE; JIANG, 2014). A utilização de soluções distribuídas que suportam modelos NoSQL permitem a evolução contínua dos esquemas de dados e o processamento paralelo (*cluster*) de forma massiva e escalável.

É importante destacar também que alguns trabalhos enfatizam a fusão de dados e metadados (semântica) das trajetórias. Esta fusão permite que dados que enriqueçam o significado da geolocalização do objetivo móvel sejam armazenados para posterior análise. Entretanto, para que isso seja possível são necessários modelos de dados semiestruturados que possibilitem a constante evolução dos esquemas para acomodar novos metadados a medida que novas características do objeto forem obtidas (BOULMAKOUL et al., 2015).

Quadro 2 – Trabalhos relacionados

Trabalho	**Arquitetura**		**Big Data**	**NoSQL**	
	Arcabouço	**SOA**		**Modelo**	**NoSQL**
Murakami et al., 2007	X	**X**			
Zhong et al., 2012	X		**X**		**X**
Amirian et al., 2013	X	**X**	**X**		**X**
Almeida et al., 2012				**X**	
Yue e Jiang, 2014			**X**		
Hajari e Hakimpour, 2014				**X**	
Boulmakoul et al., 2015				**X**	

Fonte: produção do próprio autor, 2016.

Sobre os trabalhos que abordam arquiteturas orientadas a serviço, de forma geral, todos destacam a importância da uti-

lização de uma camada de serviços no suporte ao processamento das grandes massas de dados. Como exemplo, Murakami et al. (2007) enfatizam a melhora da eficiência no processamento de dados de agricultura de precisão por meio da construção de uma arquitetura orientada a serviço. Mesmo não estando relacionado ao contexto de trajetórias de objetos móveis, por propor uma arquitetura para processar dados georeferenciados, o trabalho de Murakami et al. (2007) corrobora com a aplicabilidade de SOA no contexto de análise de grandes volumes de dados geoespaciais.

Apesar de proporem o armazenamento flexível e o processamento distribuído de grandes massas de dados, os trabalhos relacionados identificados na literatura não se preocuparam no armazenamento de dados em múltiplos modelos. Este tipo de estratégia propicia que os dados mantenham suas características originais, pois não precisam sofrer transformações para serem persistidos. A este tipo de técnica dá-se o nome de persistência poliglota. Portanto, uma das principais contribuições do trabalho em tela é explorar o armazenamento de dados de trajetória utilizando uma arquitetura orientada a serviços e que proveja persistência poliglota.

4

DESENVOLVIMENTO DA SOLUÇÃO

Com a ampliação constante da quantidade, variedade e velocidade dos dados de trajetórias coletados por aparelhos móveis, a possibilidade de análise e geração de conhecimento sobre estes dados ganha cada dia mais importância. Entretanto, analisar de forma extensiva e incremental massivas quantidades de trajetórias demanda de um *framework* metodológico e computacional adequado. Conforme os trabalhos relacionados apresentados no capítulo 3, é possível perceber que as contribuições na direção de prover um *framework* neste sentido são estritamente pontuais. Dentre as demandas pertinentes necessárias para este *framework*, destacam-se a necessidade de um modelo de dados que represente de forma adaptativa a grande diversidade dos dados contidos em trajetórias, assim como a necessidade de uma infraestrutura computacional que permita análises múltiplas e incrementais sobre as trajetórias, de forma integrada e transparente.

Com base nestas necessidades, este trabalho propõe a utilização da persistência poliglota, para o armazenamento dos dados em diferentes modelos, e do padrão arquitetural de orientação a serviços, para o processamento e análise das trajetórias de objetos móveis. Com a união destas duas práticas, torna-se possível realizar o armazenamento das diferentes informações contidas

nas trajetórias, assim como a análise delas para os mais diversos propósitos.

A abordagem utilizada para o desenvolvimento da solução se baseia no ciclo de vida de *software*. Portanto, as próximas seções estão estruturadas da seguinte forma. A seção 4.1 apresenta os requisitos que devem ser atendidos para que a arquitetura possa armazenar e processar dados de trajetórias. A seção 4.2 apresenta o projeto do modelo de dados para o uso da persistência poliglota. A seção 4.3 apresenta o projeto da arquitetura orientada a serviços. A seção 4.4 apresenta a implementação da arquitetura proposta.

4.1 Requisitos

Os requisitos para o desenvolvimento da solução foram organizados em duas categorias, requisitos de dados e de aplicação. Os primeiros estão relacionados aos requisitos que devem ser atendidos pelos esquemas de armazenamento de dados e pela camada de persistência, já os segundos estão relacionados aos requisitos que devem ser satisfeitos pela arquitetura que dá suporte à análise das trajetórias.

4.1.1 Requisitos de dados

Para os esquemas e estruturas de dados a serem gerenciados e armazenados pela arquitetura foram elicitados os seguintes requisitos.

4.1.1.1 Suportar dados estruturados e não estruturados de uma trajetória

A definição formal de trajetória bruta fornecida por Parent et al (2013) descrita na subseção 2.1 na página 43, caracteriza o requisito mínimo que pode ser considerado como dado estruturado de uma trajetória acrescida de informações não necessariamente obrigatórias. Como as trajetórias de objetos móveis podem ser enriquecidas com informações de outras fontes de dados, tais

como sensores ou mesmo anotações manuais ou automáticas, os autores também levam em consideração estas informações extras e opcionais por meio do símbolo δ. Os dados δ tornam a trajetória um dado semi-estruturado ou, até mesmo, não-estruturado.

Assumindo que novos dispositivos são criados regularmente e que estes comumente fornecem cada vez mais dados que pode vir a ser relevantes nas minerações e análises comportamentais dos objetos móveis, o armazenamento conjunto dos dados estruturados e não estruturados das trajetórias leva à necessidade de modelos de dados que acomode estas particularidades.

4.1.1.2 Suportar dados de trajetórias em diferentes estados

Os dados de uma trajetória podem estar em diferentes estados. Após a sua carga inicial, ela é considerada uma trajetória de dados brutos, ou seja, ainda sem nenhum processamento ou limpeza. Após os primeiros processos executados, diferentes estados da mesma trajetória são observados, como por exemplo, bruta, compactada, limpa, entre outras. Muitas vezes, durante experiências efetuadas com trajetórias, são feitos comparativos entre algoritmos que modificam a trajetória. Com o armazenamento de múltiplos estados dos dados das trajetórias, o resultado de diferentes processos e algoritmos são facilmente comparados, pois todos estarão disponíveis no armazenamento.

4.1.1.3 Suportar *Big Data*

O problema do *Big Data* é fortemente perceptível no contexto de trajetórias de objetos móveis, principalmente no que se refere ao volume dos dados históricos. Um exemplo de quantidade de dados gerados e coletados por sensores GPS é a página de estatísticas do site do projeto *OpenStreetMap* que apresenta na data de 22 de Janeiro de 2016 a quantidade de mais de 5 bilhões de pontos GPS coletados (OPENSTREETMAP, 2016). Outro exemplo são as trajetórias de táxis disponibilizados pelo projeto T-Drive, onde em apenas uma semana de rastreamento efetuada na cidade de Pequim, foram coletados 15 milhões de

pontos GPS das trajetórias efetuadas pelo táxis participantes (T-DRIVE, 2010).

No Brasil, por meio da iniciativa Dados Abertos da prefeitura da cidade do Rio de Janeiro e mantido pela Federação das Empresas de Transporte de Passageiros do Estado do Rio de Janeiro (FETRANSPOR), é possível acessar os dados de localização dos ônibus circulares. Após serem efetuadas consultas, observou-se que são coletados em média 750 pontos de localização por dia para cada um dos 7.678 ônibus rastreados[1] (DATA.RIO, 2014).

Um outro dado interessante vem do relatório da *United Nations International Telecommunication Union*, que mostra que existem mais de 7 bilhões de linhas de celular no mundo, sendo praticamente igual à quantidade de habitantes e cobertura de rede 3G atende à 69% da população (SANOU, 2015). Este dado apresenta um grande potencial de geração de dados de trajetórias devido ao sensor GPS instalado na maioria destes aparelhos.

Presumindo que com apenas os dados mínimos necessários para armazenar um ponto seja latitude, longitude e o momento temporal, que seriam dois números de ponto flutuante de 32 bits e um número inteiro de 64 bits, cada ponto irá ocupar pelo menos 16 bytes. Ao aplicar o mínimo de 16 bytes ao dados, por exemplo, do *OpenStreeMap* chega-se a quantidade de 820 gigabytes de dados de armazenamento. Já para os dados mínimos do projeto T-Drive, para apenas uma semana, seriam de 229 megabytes.

Além do armazenamento, o próprio processamento também pode ser um problema de *Big Data*. Dependendo do problema a ser analisado, a quantidade de trajetórias está diretamente relacionada à qualidade do resultado obtido. Ou seja, quanto mais dados forem analisados melhor será o resultado. Portanto, muitas vezes a quantidade de trajetórias analisadas para se obter um resultado satisfatório leva o problema ao âmbito do *Big Data*.

[1] Dados observados durante o mês de Novembro/2015.

4.1.2 Requisitos de aplicação

Visando facilitar a interoperabilidade e reutilização dos processos utilizados em vários tipos de análises de trajetórias, a arquitetura proposta neste trabalho deve satisfazer os seguintes requisitos de aplicação.

4.1.2.1 Interoperabilidade com novos serviços

Por ser uma área onde as pesquisas estão em franca expansão, a análise de trajetórias de objetos móveis necessita de uma arquitetura que suporte novos serviços (processos) da maneira mais transparente possível. Esta arquitetura precisa facilitar a utilização de serviços existentes assim como a implantação de novos serviços criados para resolver problemas identificados em novas pesquisas. Esta arquitetura deve dar suporte à interoperação entre serviços, independentemente da linguagem de implementação, por meio de um protocolo simples de comunicação.

4.1.2.2 Descoberta de serviços

Para aumentar a reutilização dos serviços é necessário facilitar a sua localização. Para que isso seja possível é necessário que a arquitetura forneça um mecanismo que possibilite o cadastro e a pesquisa pelos serviços por ela disponibilizados. Como resultado, este mecanismo deve fornecer o endereço de acesso e a interface dos serviços pesquisados. Além de possibilitar a descoberta, este mecanismo também deve ser capaz de monitorar a "saúde" dos serviços nele cadastrados. Assim, é possível identificar quando um serviço, mesmo ainda cadastrado, está sobrecarregado ou mesmo indisponível.

4.1.2.3 Escalabilidade horizontal e computação elástica

Uma das características comum a aplicações que atendem problemas de *Big Data* é a possibilidade de escalar horizontalmente seu poder de armazenamento e processamento. Sob o ponto de vista de armazenamento, a solução proposta neste tra-

balho atende esta necessidade com o uso de tecnologias NoSQL. Sob o ponto de vista de aplicação, a arquitetura deve possibilitar o balanço de carga e a instanciação de novas cópias de serviços que estejam sobrecarregados, assim como o desligamento de serviços que não estão sendo utilizados. Esta abordagem racionaliza o uso de recursos por meio da distribuição do processamento em um *cluster* de computadores *commodities*[2]. Dorband et al. (2003) afirma que com o uso de computadores *commodities*, o custo de infraestrutura de processamento é reduzido consideravelmente.

4.2 Projeto do modelo de dados

Baseado nos requisitos de dados levantados na seção 4.1.1 é possível realizar o projeto do modelo de dados da arquitetura. Seguindo as fases de projeto de banco de dados, primeiramente é apresentado o projeto conceitual do esquema na seção 4.2.1, e depois os mapeamentos para cada um dos modelos de dados utilizados pela arquitetura, sendo eles: Documento (seção 4.2.2.1), Chave-Valor (seção 4.2.2.2), Família de Coluna (seção 4.2.2.3) e Relacional (seção 4.2.2.4).

4.2.1 Projeto conceitual do esquema de dados de trajetórias

Conforme Hoberman (2014), projeto de dados é o processo de aprendizagem sobre os dados e o esquema de dados é o resultado final do processo de modelagem. Hoberman (2014) também enfatiza que o projeto dos dados com NoSQL pode ser mais importante do que quando usado bancos de dados relacionais, pois estes possibilitam um nível maior de estruturação da informação. Portanto, bancos de dados com capacidade de esquema flexível exigem mais disciplina no seu desenvolvimento e utilização.

Neste trabalho o formalismo escolhido para projetar o esquema de dados da arquitetura foi a modelagem orientada a objetos, onde, particularmente, foi utilizado o diagrama de classes

[2] *Commodities* são produtos de qualidade e características uniformes.

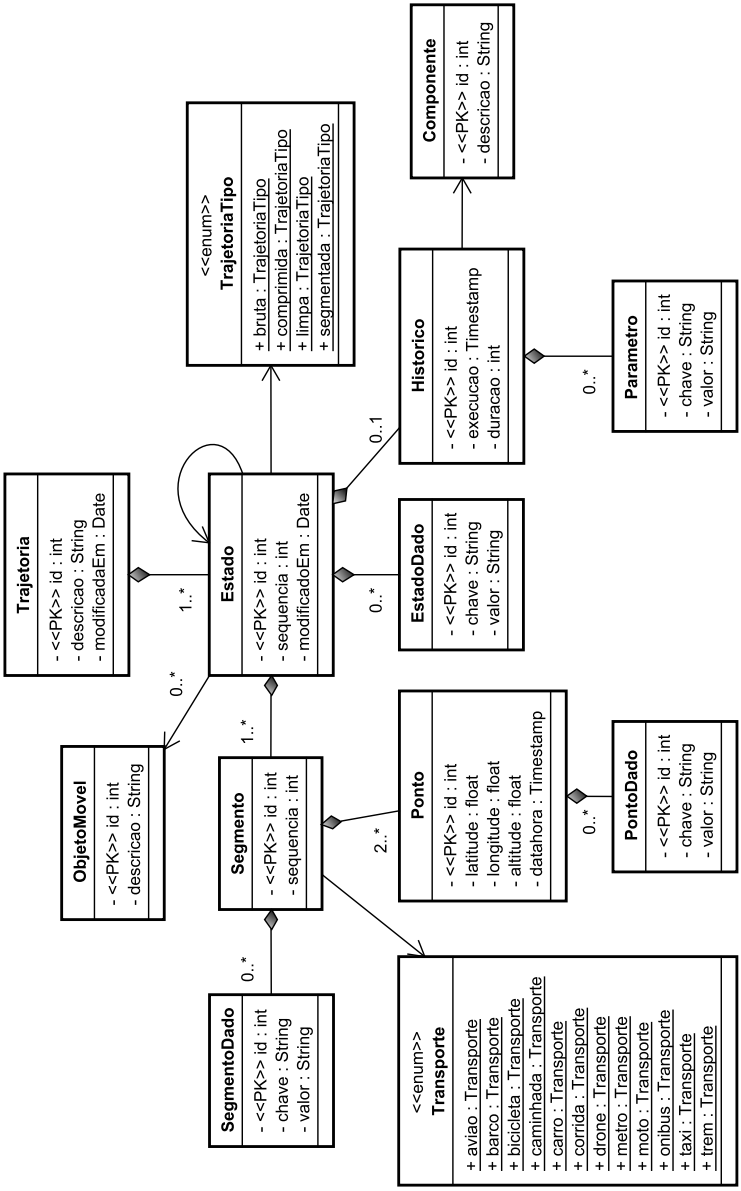
para modelar o esquema conceitual de dados. Entretanto, mesmo utilizando um formalismo mais robusto que o modelo Entidade-Relacionamento, o mais comumente utilizado para este propósito, algumas dificuldades foram enfrentadas, tais como:

1. Nem o modelo orientado a objetos nem o modelo relacional suportam atributos dinâmicos. Apesar dos modelos NoSQL suportarem a inclusão dinâmica de dados, o que atende o requisito de inclusão de informações oriundas de novos sensores e de resultados de cálculos, os modelos tradicionais não suportam, exigindo a criação de entidades específicas para os dados adicionais.

2. Necessidade de campo de identificação nas relações. Devido à necessidade de utilização de chaves estrangeiras no modelo relacional, faz-se necessária a utilização de atributos de identificação nas diversas entidades criadas, mesmo que elas não sejam necessárias para os modelos de dados agregados, como é o caso do chave-valor, documento e família de colunas.

O projeto conceitual das entidades de dados que representam uma trajetória é apresentado no diagrama de classes UML da Figura 4 na próxima página.

Figura 4 – Diagrama de classes UML do modelo de dados da arquitetura



Fonte: produção do próprio autor, 2016.

4.2.1.1 Descrição das entidades do esquema

No quadro 3 são descritas as entidades (classes) do modelo de dados representado pela Figura 4 na página anterior em ordem alfabética:

Quadro 3 – Descrição das entidades no modelo de dados

Entidade	**Descrição**
Componente	Componente de processamento da camada de serviços
Estado	Um estado dos dados da trajetória
EstadoDado	Dados adicionais do estado dos dados da trajetória
Historico	Histórico de processamento que gerou o estado dos dados da trajetória
ObjetoMovel	Descrição do objeto móvel que originou os dados da trajetória
Parametro	Parâmetros passados para o componente que gerou o estado
Ponto	Um ponto x, y do segmento da trajetória no momento t
PontoDado	Dados adicionais sobre o ponto do segmento da trajetória
Segmento	Segmento de trajetória
SegmentoDado	Dados adicionais sobre o segmento de trajetória
Trajetoria	A trajetória em si
TrajetoriaTipo	Enumeração com os tipos de dados de trajetórias
Transporte	Enumeração com os tipos de meio de transporte

Fonte: produção do próprio autor, 2016.

O histórico de processamento e estados da trajetória são mantidos referenciados pelo componente de catálogo. Conforme Figura 4 na página oposta, o modelo de dados das trajetórias

é composto pela estado e seu tipo, pelos pontos da trajetória e seus dados, e pelo histórico de processos executados para gerar tal estado. Um histórico de processos pode ser exemplificado da seguinte forma: Carga inicial, limpeza com o algoritmo de clusterização DBScan (RINZIVILLO et al., 2008) e compactação com o algoritmo Doulgas-Peucker (DOUGLAS; PEUCKER, 1973). O modelo de dados deve viabilizar o armazenamento de diversos estados de uma mesma trajetória, cada uma com um histórico de processamento diferente.

4.2.2 Mapeamento do modelo conceitual para lógico dependente de implementação

Apesar dos modelos NoSQL não exigirem um esquema rígido de dados devido à propriedade *schemaless*, neste trabalho optou-se por apresentar o mapeamento do modelo conceitual em cada um dos modelos NoSQL a fim de esboçar como eles devem ser implementados. Portanto, as próximas subseções apresentam o mapeamento do modelo definido na Figura 4 na página 72 para cada um dos modelos utilizados pela arquitetura proposta.

4.2.2.1 Documento

O mapeamento para o modelo documento tomou como base a criação de um elemento no documento para cada entidade do modelo conceitual da Figura 4 na página 72. Cada relacionamento 1 para 1 presente no projeto conceito foi mapeado como uma entidade agregada ao elemento já definido, assim como cada relacionamento 1 para N foi mapeado como uma lista de entidades associada à entidade do lado 1 na forma de uma hierarquia. A representação hierárquica agrega os dados da trajetória fazendo com que eles possam ser encapsulados em um documento.

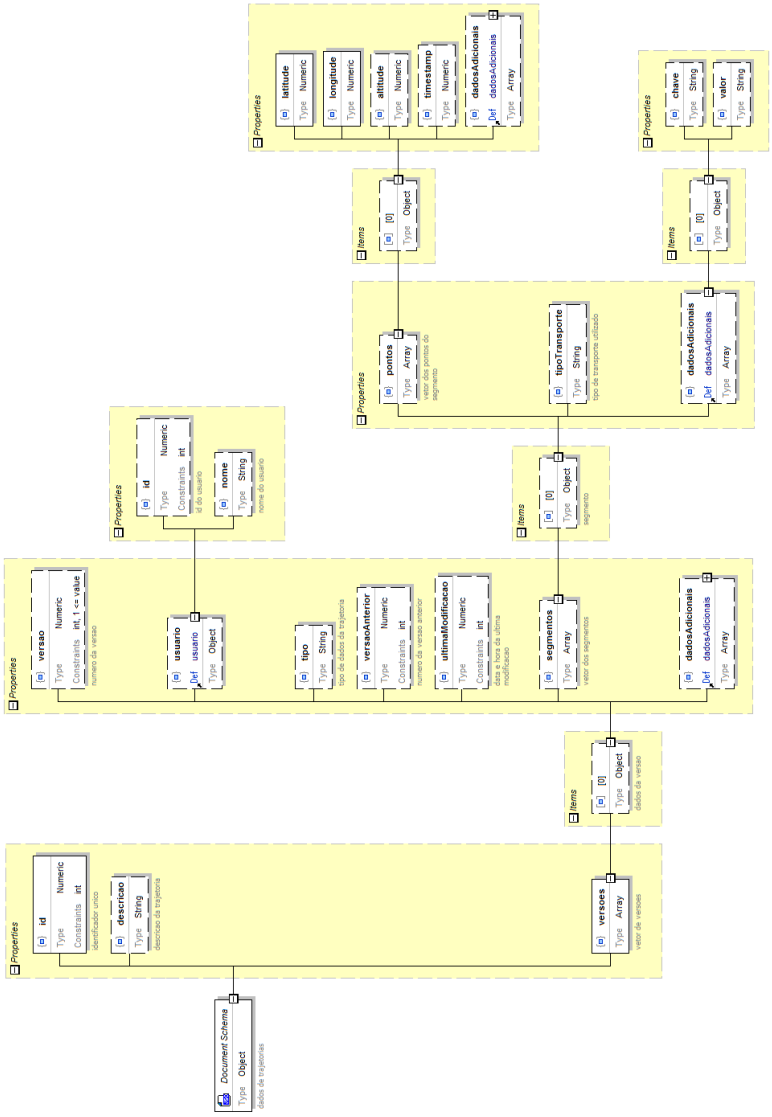
O armazenamento no formato documento representa um modelo em árvore. Ao observar o diagrama de classes na Figura 4 na página 72 é possível perceber que a modelagem das trajetórias é representada por uma árvore. Muitas vezes um modelo orientado a objetos é estruturado como um grafo, mas os dados de

trajetória refletem uma árvore. Por este motivo, o modelo documento possui uma menor diferença de impedância[3] com relação ao modelo proposto da modelagem das trajetórias.

Para representar a estrutura de armazenamento da trajetória no modelo documento foi definido um esquema JSON para a Figura 5 na página seguinte. O fonte deste esquema pode ser encontrado no Apêndice A na página 129.

[3] Expressão criada por Ambler(2003)

Figura 5 – Esquema JSON das trajetórias



Fonte: produção do próprio autor, 2016.

4.2.2.2 Chave-valor

O armazenamento de trajetórias neste modelo pode ser realizado de diversas maneiras, tais como: (i) Objeto na sua forma binária ou em formato texto JSON, (ii) uma chave para cada atributo, (iii) uma chave para cada entidade, entre outras.

O armazenamento no formato chave-valor é o mais simples dos modelos NoSQL. Nele, o valor pode ser qualquer dado textual ou binário sendo este identificado pela sua chave. Devido a esta característica, antes de se definir a forma de armazenamento é especialmente necessário refletir *a priori* sobre como a informação será consumida. Esta questão é particularmente importante quando o dado é armazenando em formato binário, pois o dado se torna dependente da tecnologia em uso. Por exemplo, um objeto Java armazenado em binário só poderá ser lido por esta mesma tecnologia, dificultando assim o uso dele. As opções que utilizam uma chave para cada atributo (opção ii) ou entidade (opção iii), por requerer uma série de buscas no banco de dados de forma a reconstruir a trajetória, foram descartadas.

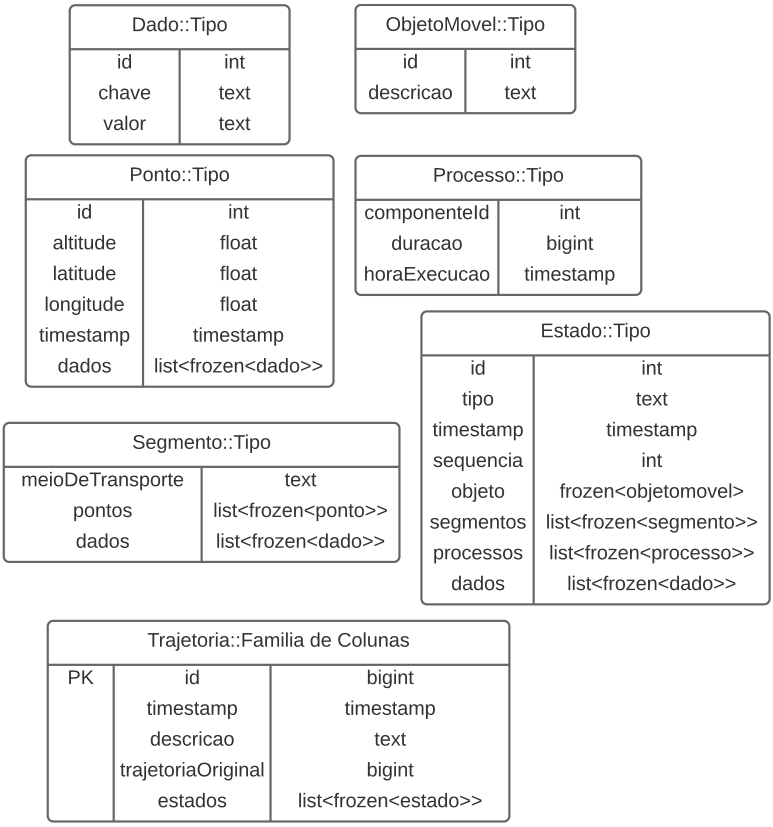
Portanto, neste trabalho o armazenamento escolhido é na forma de uma chave por trajetória (opção i) com valor em formato texto JSON, usando o esquema já apresentado na Figura 5 na página oposta.

4.2.2.3 Família de colunas

O modelo família de colunas também segue a forma tabular do modelo relacional, mas sem as restrições de integridade impostas por este último. Além disso, o modelo família de colunas possibilita a criação de atributos compostos e multivalorados, características não suportada pelo relacional tradicional, apenas pela sua extensão objeto-relacional. No mapeamento para este modelo foram criados tipos para cada uma das entidades do modelo conceitual e uma tabela que representa a família de colunas. Como não há uma representação formal gráfica deste modelo, foi utilizado um diagrama *Logical Data Structure* (LDS), similar ao que é utilizado para o modelo relacional.

O esquema apresentado na Figura 6 contém apenas uma família de colunas, que é a trajetória, e seis tipos definidos pelo usuário que são utilizados para definir os atributos da família de colunas.

Figura 6 – Modelo família de colunas



Fonte: produção do próprio autor, 2016.

4.2.2.4 Relacional

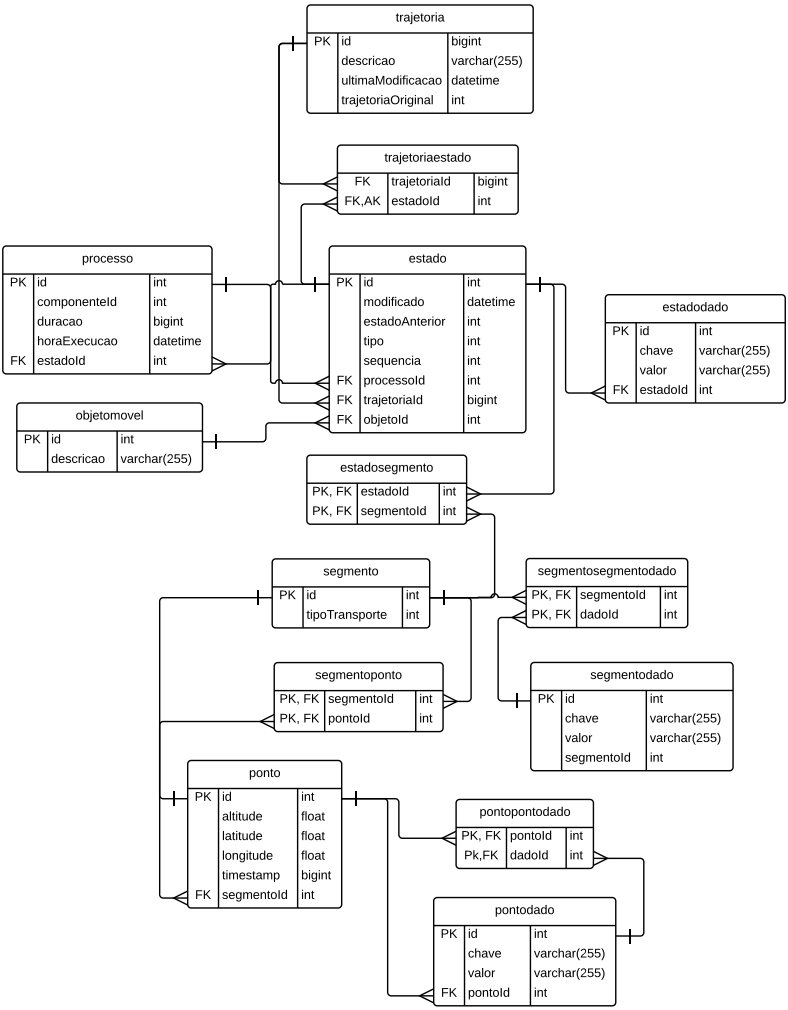
O mapeamento para o modelo relacional tomou como base a criação de uma relação para cada entidade do modelo conceitual da Figura 4 na página 72. Além disso, cada relaci-

onamento presente no modelo conceitual implicou a criação de uma nova relação para representá-lo. Esta abordagem, comum nos mapeamentos objeto-relacionais, faz com que seja necessária a definição de um número elevado de relações que têm como propósito exclusivo a representação dos relacionamentos. Isso faz com que a reconstrução completa do dado, por exemplo, da trajetória, precise envolver a consulta a várias relações por meio da aplicações de junções, operações estas que têm um elevado custo computacional.

Como o paradigma orientado a objetos é baseado em princípios de estruturas complexas (os objetos) que se relacionam por meio de referências a objetos (os OIDs), e o paradigma relacional é baseado em princípios de estruturas simples bidimensionais (as tabelas) que se relacionam por referências explícitas a chaves estrangeiras, esses dois paradigmas não podem ser mapeados diretamente sem a necessidade de transformações nos dados. Este problema é conhecido como diferença de impedância objeto-relacional (AMBLER, 2003). Devido a esta diferença, o armazenamento no modelo relacional exige um processo de mapeamento objeto-relacional (ORM), o que amplia consideravelmente a complexidade de armazenamento, sendo inclusive necessária a criação de relações associativas devido às navegações bidirecionais definidas no modelo conceitual orientado a objetos.

O modelo entidade relacionamento está apresentado na Figura 7 na próxima página.

Figura 7 – Modelo entidade relacionamento



Fonte: produção do próprio autor, 2016.

4.3 Projeto da Arquitetura

Baseado nos requisitos de sistema apresentados na seção 4.1 na página 66, a Figura 8 na página 83 apresenta o projeto da arquitetura construída para satisfazê-los. Esta arquitetura foi concebida em camadas, onde as camadas projetadas são a de serviços de trajetória e a de serviços de persistência

A construção de projetos em camadas é um padrão clássico de desenvolvimento de *software*, onde as três camadas fundamentais são: (i) apresentação, (ii) domínio de negócio e (iii) fonte de dados (FOWLER, 2002; ECKERSON, 1995)

De acordo com Fowler (2002), com a utilização deste padrão de projeto é possível obter as seguintes vantagens:

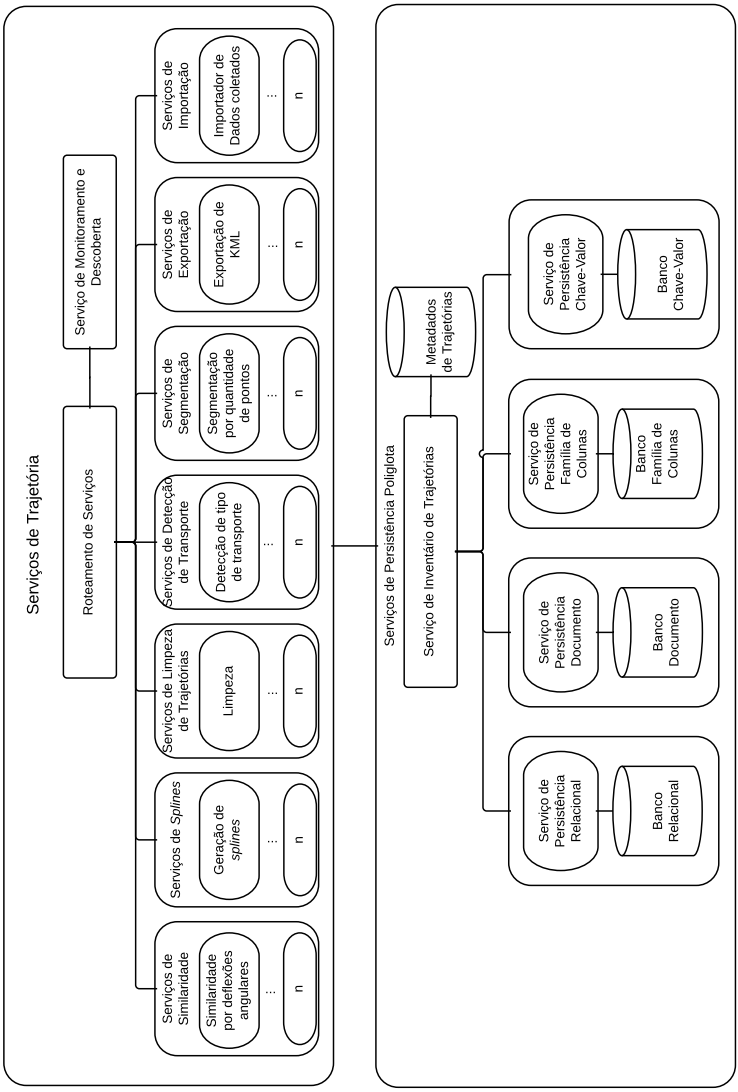
- Facilitar a compreensão de uma camada como sendo um todo coerente sem o conhecimento do funcionamento das demais camadas.
- Ter a possibilidade de substituir camadas por implementações diferentes dos mesmos serviços.
- Reduzir as dependências entre as partes.
- Criar pontos de localização para a padronização do desenvolvimento.

De forma a prover maior flexibilidade e extensibilidade a arquitetura, optou-se por desenvolver seus componentes utilizando a abordagem de microsserviços. Desta maneira, cada retângulo arredondado representado na Figura 8 na página 83 representa um microsserviço coeso e com baixo acoplamento.

O uso de microsserviços é vantajoso neste caso devido ao suporte à diversidade tecnológica (FOWLER, 2015), pois permite que os componentes sejam desenvolvidos em diferentes linguagens o que atende ao requisito de interoperabilidade com novos serviços independente da linguagem de implementação. Outro requisito atendido com os microsserviços é o da escalabilidade horizontal, pois estes podem ser replicados independentemente em diversos servidores.

As duas camadas providas pela arquitetura foram concebidas para possibilitar a adição tanto de serviço de processamento de trajetórias quanto de persistência. Estas características são suportadas pelos microsserviços de roteamento, e de monitoramento e descoberta, para os serviços de processamento de trajetórias, e pelo microsserviço de inventário de trajetórias, para o serviços de persistência. As próximas subseções apresentam detalhes sobre cada um destes serviços fornecidos pela arquitetura.

Figura 8 – Diagrama de arquitetura



Fonte: produção do próprio autor, 2016.

4.3.1 Camada de serviços de trajetória

O propósito da camada de serviços de trajetória é prover às aplicações cliente não apenas serviços de processamento de trajetórias mas também o suporte necessário para abstrair deles a descoberta dos serviços (catálogo de componentes). Esta camada também tem por intuito resolver as requisições de entrada e saída de dados dos serviços que a compõem. Utilizando o catálogo de serviços, ela pode identificar onde está armazenada a trajetória requisitada ou direcionar a saída do componente para o repositório correto.

Cada microsserviço contido nela fornece uma interface de comunicação com as aplicações cliente usando o modelo de *web-services*. Esta interface é implementada utilizando-se o padrão REST que executa sobre o protocolo HTTP e os dados são transferidos utilizando o formato JSON. Conforme Webber, Parastatidis e Robinson (2010), a *Web* passa a ser uma plataforma para a construção de serviços distribuídos, e as URI's são os endereços dos serviços disponibilizados seguindo o padrão REST.

A escolha do padrão REST dá-se pelas vantagens que ele tem apresentado sobre o padrão SOAP. Conforme Muehlen, Nickerson e Swenson (2005) o padrão REST oferece baixo acoplamento entre as partes, o que não ocorre com o padrão SOAP que necessita do conhecimento prévio do cliente sobre as operações existentes em um serviço, descrito no formato WSDL.

Além das vantagens advindas pelo uso do padrão REST, outros benefícios podem ser associados ao formato JSON. Ele é menos verboso que o XML (padrão adotado pela W3C), sendo assim ele necessita de menos recursos computacionais que o formato XML para seu processamento e transferência (NURSEITOV et al., 2009).

Para uma aplicação cliente usar os serviços é necessário que ela envie uma requisição para o serviço de roteamento passando a identificação do serviço desejado e o *payload*[4] com os dados da trajetória que deseja processar e/ou armazenar. Como

[4] *Payload* é o nome dado a parte dos dados de uma transmissão.

a arquitetura utiliza microsserviços, é possível, caso a aplicação cliente possuir tal informação, enviar a requisição diretamente para o serviço desejado. Essa característica é interessante nas situações onde o cliente já conhece a localização do serviço que deseja utilizar, diminuindo assim a quantidade de mensagens trocadas e, por consequência, o tempo de processamento. A seção 4.3.1.1 apresenta mais detalhes sobre a localização dos serviços fornecida pelo microsserviço de roteamento.

4.3.1.1 Roteamento de serviços

O roteamento de serviços segue o padrão de integração empresarial (EIP) *Dynamic Router* que é composto por um *Message Router* e uma base de regras para determinar o destino da mensagem (HOHPE; WOOLF, 2003). Esse padrão de integração é particularmente interessante em situações onde o destino da mensagem muda frequentemente com o acréscimo de novos componentes, que no caso deste trabalho são os microsserviços.

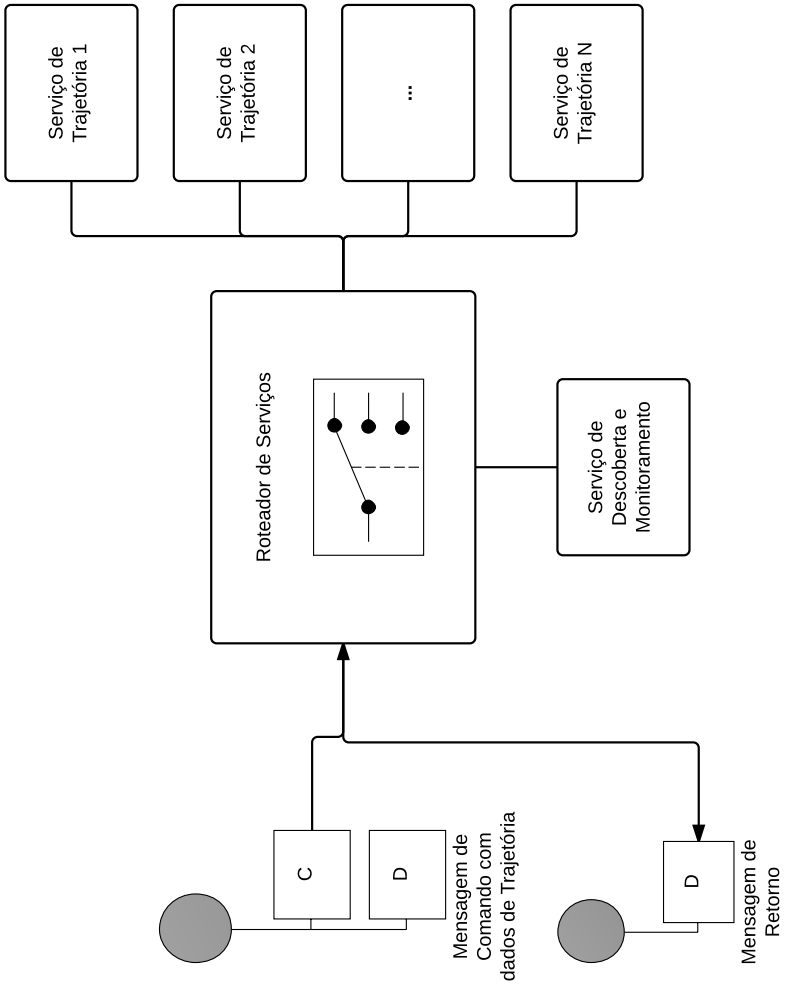
Para a base de regras, foi utilizado um serviço de descoberta e monitoramento de serviços que está descrito na seção 4.3.1.2 na página 87.

A Figura 9 na página seguinte apresenta uma mensagem com um comando de processamento (C)[5] e uma trajetória de *payload* (D) sendo direcionada pelo roteador de serviços para o componente responsável pela execução do comando (C).

A Figura 10 na página 87 apresenta o processo de roteamento de serviços passo a passo por meio de um diagrama UML de atividades. Neste diagrama é possível observar o papel desempenhado por cada microsserviço no processo de armazenamento de uma trajetória.

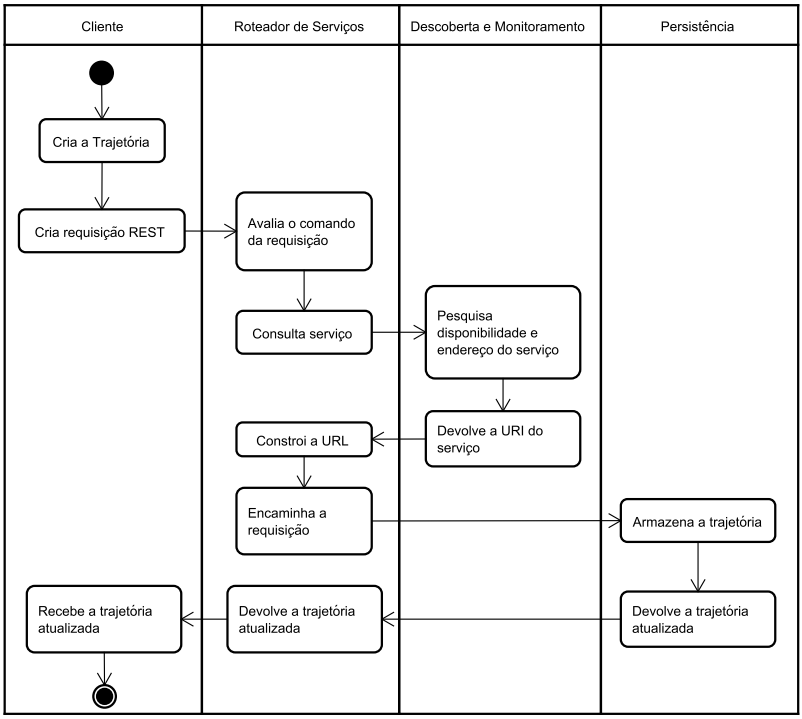
[5] No padrão EIP, a letra "C" na mensagem representa um Comando e a letra "D" representa um Documento.

Figura 9 – Roteamento de serviços



Fonte: produção do próprio autor, 2016.

Figura 10 – Atividades do roteamento de serviços



Fonte: produção do próprio autor, 2016.

4.3.1.2 Descoberta e monitoramento de serviços

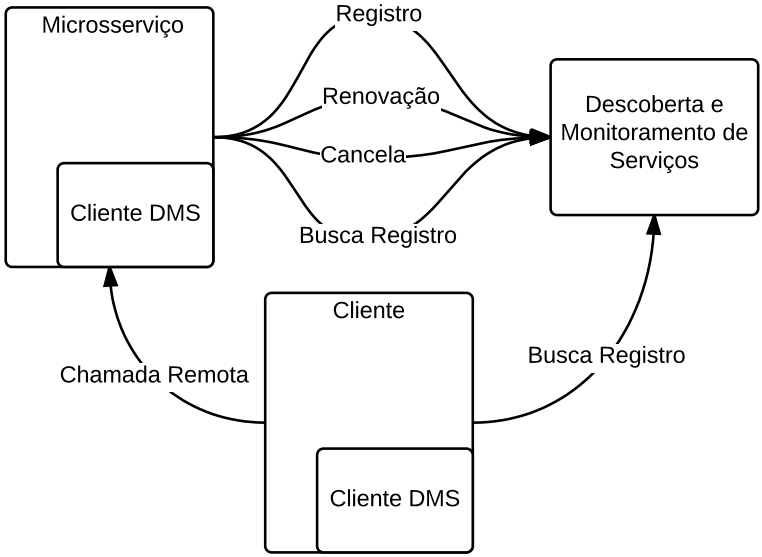
O componente de descoberta e monitoramento de serviços (DMS) é uma implementação do padrão de projeto *Service Registry* para microsserviços utilizando a estratégia *Self Registration* (RICHARDSON, 2015). Neste modelo, quando um serviço inicia, ele deve enviar uma requisição de registro para o DMS e uma requisição de cancelamento ao sair do ar. Para o monitoramento do estado de execução do serviço, é necessário que ele envie um *heartbeat*[6] regularmente para que o DMS o considere

[6] *Heartbeat* é um sinal periódico gerado por *hardware* ou *software* para indicar que ele ainda está em execução.

apto para continuar a receber requisições.

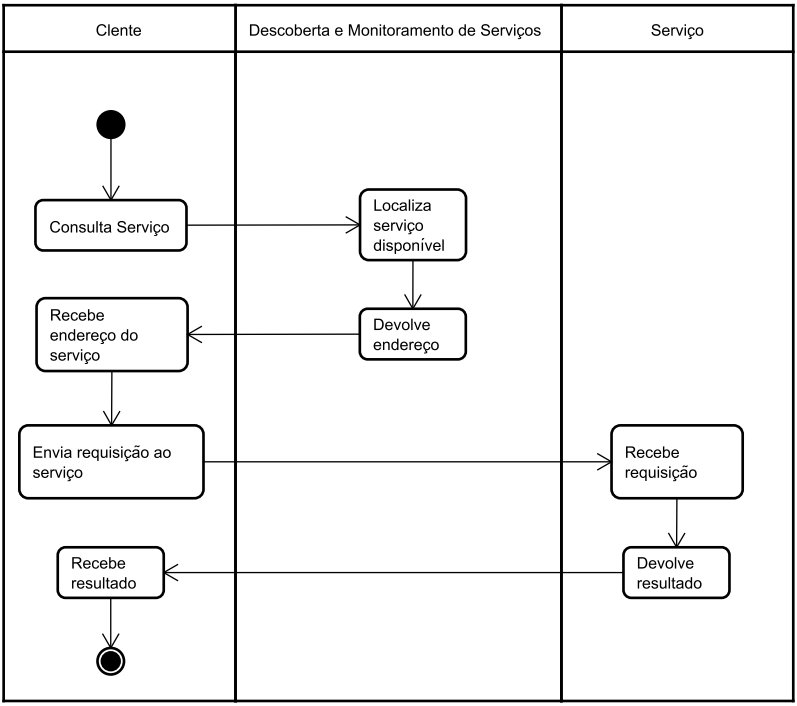
Sempre que um processo cliente necessitar a URI de um serviço para o envio de requisições, basta que faça uma consulta ao DMS para saber quais serviços existem e o seus endereços. A Figura 11 apresenta a arquitetura de alto nível do relacionamento entre os serviços, o cliente e o componente de descoberta e monitoramento de serviços (NETFLIX, 2014). A Figura 12 na próxima página apresenta a sequência de atividades para a descoberta do endereço de um serviço e o subsequente envio da requisição para este a partir de um processo cliente, que representa tanto uma aplicação externa que deseja localizar um serviço especificamente quanto o roteador do *framework* proposto.

Figura 11 – Descoberta e monitoramento de serviços



Fonte: Netflix, 2014.

Figura 12 – Pesquisa de serviços no DMS



Fonte: produção do próprio autor, 2016.

4.3.1.3 Componentes de negócio

A camada de serviços de trajetória contém os microsserviços que implementam os processos diretamente relacionados ao domínio da aplicação, que no âmbito deste trabalho estão relacionados ao processamento de trajetórias de objetos móveis. Dado que estes componentes são implementados como serviços, eles podem ser combinados em cadeia a fim de implementar um processo de negócio completo e assim gerar o resultado esperado da aplicação cliente.

Cada microsserviço define a sua especificação contratual, também conhecida como a interface do componente. Os componentes de aplicação são divididos nas seguintes categorias: im-

portação, processamento e exportação.

Componentes de importação
Seu propósito é servir de ponte para a entrada da dados de aplicações externas que coletam dados de trajetórias. Esses dados podem ser brutos ou já processados por outra aplicação. Um exemplo seria o envio de dados GPS coletados por telefones celulares.

Componentes de processamento
Os componentes de processamento comumente agem sobre trajetórias já armazenadas e têm por propósito analisar o conjunto de dados a fim de gerar novos dados e informações. Os componentes desta camada são categorizados de acordo com o seu objetivo de processamento, podendo ser classificados como: de compactação, de limpeza, de classificação (tipo de trajetória, tipo de transporte, etc.), de segmentação entre outros. Estes componentes, após a sua execução, podem fornecer um novo conjunto de dados que é então armazenado como sendo um novo estado da trajetória, para posterior recuperação por um outro componente de processamento ou um componente de exportação.

Componentes de exportação
Os componentes de exportação têm por finalidade formatar os dados armazenados de uma trajetória de acordo com o formato requisitado pelo cliente. Exemplos de formatos são a imagem oriunda da geração de um mapa ou formatos de intercâmbio de dados geoespaciais como o KML e o GeoJSON. Assim como os outros componentes da arquitetura, eles podem receber o identificador da trajetória ou a trajetória completa, e retornam a informação requisitada acerca da trajetória recebida.

4.3.2 Camada de persistência poliglota

A camada de persistência poliglota segue o padrão de projeto *Data Access Object* (DAO) definido por Alur et al (2003),

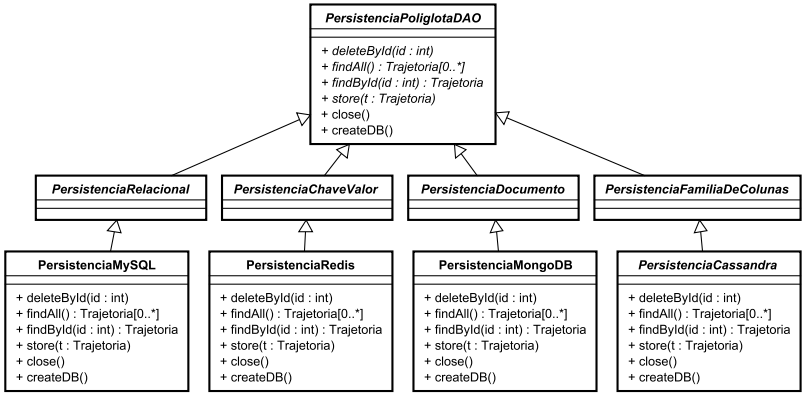
onde é definida uma interface com as operações *Create, Read, Update and Delete* (CRUD), e estas funções são implementadas de maneira específica para cada um dos modelos e gerenciadores de banco de dados utilizados. Como ainda não há um padrão de acesso a bancos de dados NoSQL, como o caso do ANSI SQL para o modelo relacional, o padrão de projeto DAO é particularmente necessário neste caso para evitar o problema chamado de *vendor lock-in*[7] (MCCREARY; KELLY, 2013; TEE, 2013).

A vantagem de utilizar o padrão DAO está no desacoplamento entre as regras de negócios e o modelo de dados da aplicação. Além disso, ela abstrai aspectos específicos do gerenciador de banco de dados, encapsulando a execução de comandos na base de dados. A camada de persistência é retratada na Figura 13 na página seguinte.

Devido à flexibilidade fornecida pela técnica da persistência poliglota e a habilidade dos modelos NoSQL de gerenciar grandes volumes de dados, o armazenamento dos diversos estados dos dados de uma mesma trajetória torna-se viável. As seções 4.3.2.2 e 4.3.2.3 apresentam, respectivamente, como é realizado o armazenamento e a recuperação de trajetórias utilizando a abordagem implementada neste trabalho.

[7] *Vendor lock-in* ocorre quando um produto fica preso tecnologicamente a um componente de um único fornecedor.

Figura 13 – Classes de persistência poliglota



Fonte: produção do próprio autor, 2016.

4.3.2.1 Catálogo de trajetórias

Devido à característica de poder armazenar as trajetórias em diferentes bancos de dados, é necessário manter um catálogo para que seja possível fazer o rastreamento das trajetórias armazenadas.

Para este componente, é feita é uma cópia de informações de metadados da trajetória e armazenadas em um repositório de dados específico para esta funcionalidade.

Os metadados selecionados para o armazenamento no catálogo são os seguintes:

- Identificação da trajetória;
- Identificação do estado da trajetória;
- Data de armazenamento;
- Usuário que a criou;
- Componente utilizado;
- Estado da trajetória;

- Banco de dados onde foi armazenada.

Por meio deste serviço é possível pesquisar no repositório se existe, por exemplo, uma trajetória segmentada, ou um trajetória compactada, ou quais as trajetórias armazenadas de um determinado usuário.

4.3.2.2 Armazenamento de trajetórias

A Figura 14 na próxima página apresenta o fluxo da mensagem que contém a trajetória a ser armazenada.

O cliente faz uma requisição REST cujo *payload* são os dados da trajetória em formato JSON. O roteamento de persistência então encaminha a mensagem de armazenamento para o serviço de persistência correspondente que faz a desserialização[8] do formato JSON para objetos em memória utilizando as classes da Figura 4 na página 72.

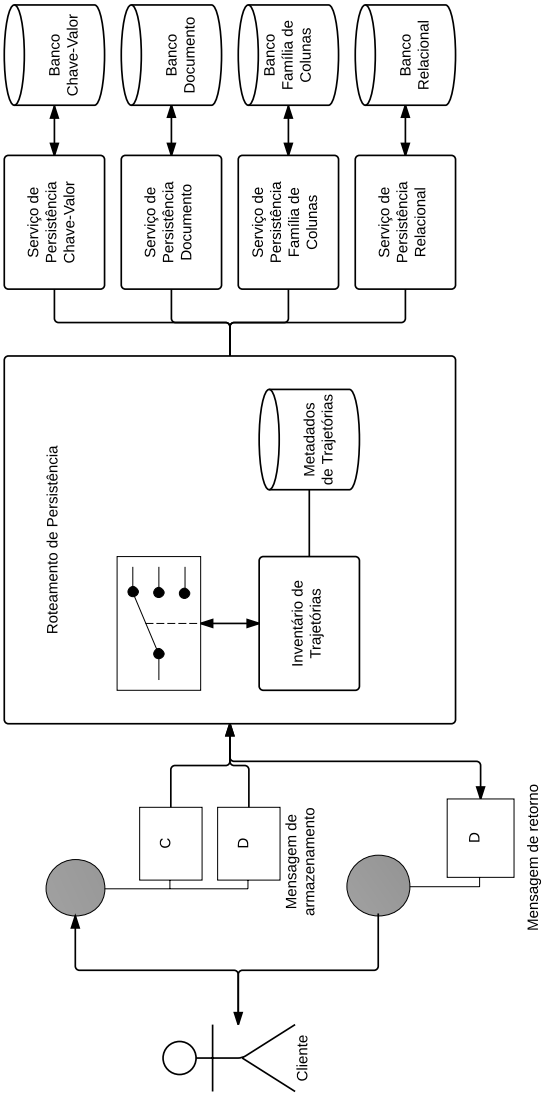
Após a instanciação destes objetos de trajetória em memória, esses dados são passados para o DAO especializado no modelo de dados utilizado que executa a transformação destes objetos em comandos específicos do SGBD, ex: SQL para o modelo relacional, comando *set* no modelo chave-valor, CQL para o modelo família de colunas ou comando *insert* para o modelo documento.

4.3.2.3 Recuperação de trajetórias

Para o processo de recuperação de trajetórias, cujo processo é basicamente o inverso ao apresentado na subseção 4.3.2.2, o cliente envia uma requisição REST com a identificação da trajetória (C) que deseja recuperar.

[8] Serialização é o processo de converter um objeto em um fluxo de *bytes* ou caracteres, a fim de que ele persista na memória, num banco de dados, ou num arquivo. O processo inverso é chamado desserialização.

Figura 14 – Armazenamento de uma trajetória



Fonte: produção do próprio autor, 2016.

O roteador de mensagens então faz uma pesquisa no inventário de trajetórias armazenadas afim de identificar em qual

modelo de dados a trajetória desejada foi armazenada. Após a identificação e confirmação que a trajetória está armazenada, a mensagem de recuperação é encaminhada ao serviço de persistência correspondente.

O serviço de persistência especializado no modelo envia os comandos necessários para a recuperação da trajetória e instancia os objetos na memória. Após isto, estes objetos são serializados no formato JSON (D) e retornado ao cliente. Este processo está representado na Figura 15 na página seguinte.

4.4 Implementação

Para a implementação da arquitetura foi utilizada a linguagem Java devido à sua popularidade (TIOBE, 2016), sua larga utilização em gerenciadores de bancos de dados NoSQL (EDLICH, 2015) e porque ela foi a linguagem utilizada na proposta inicial dos microsserviços (LEWIS, 2012). Foram selecionados alguns componentes prontos de modo a acelerar o processo de desenvolvimento.

Para a serialização e deserialização de objetos para o formato JSON, foi utilizado o Jackson JSON Processor (JACKSON, 2016). Para a camada de persistência no modelo relacional foi selecionada a ferramenta de ORM Hibernate (HIBERNATE, 2016) para as funções de criação de esquema (DDL) e de manipulações dos dados (DML). Para os demais modelos utilizados não foram localizados ou não formam necessários *frameworks* de mapeamento objeto-modelo utilizado.

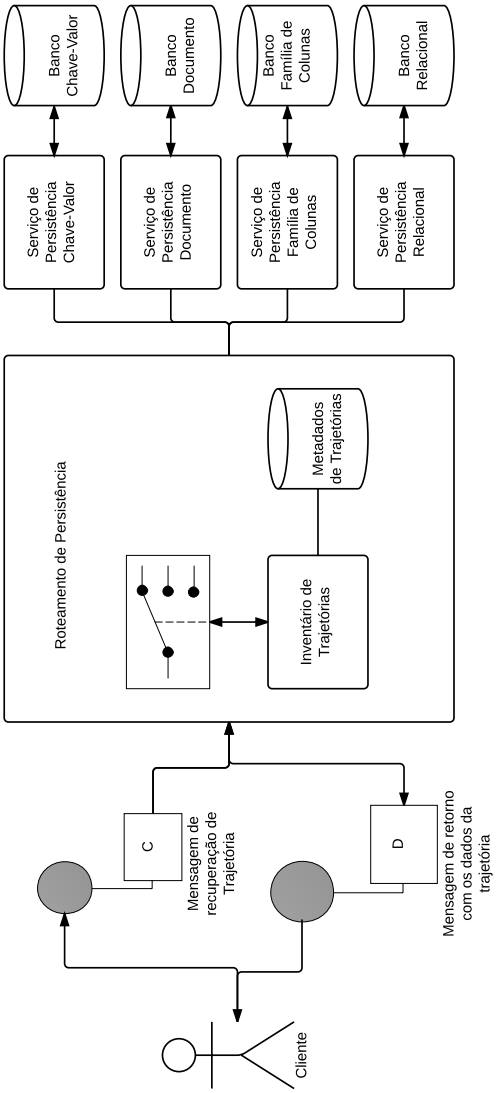
Para adequar-se ao nível de maturidade 2 do modelo de Richardson (2008), os seguintes verbos HTTP foram selecionados para as operações de persistência:

GET : Utilizado para requerer uma ou mais trajetórias

POST : Utilizado para armazenar ou modificar uma trajetória

DELETE : Utilizado para excluir uma trajetória

Figura 15 – Recuperação de uma trajetória



Fonte: produção do próprio autor, 2016.

Para a construção dos microsserviços foi selecionado o produto Dropwizard que gera um empacotamento das biblio-

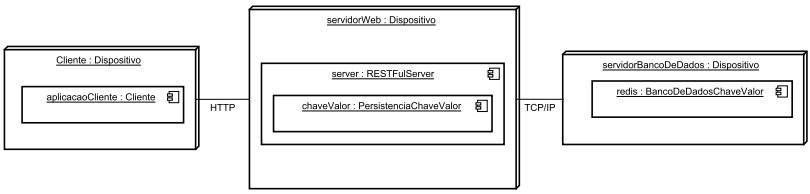
tecas necessárias para o gerenciamento do ciclo-de-vida de um serviço (DROPWIZARD, 2016). O Dropwizard embarca o Jetty (JETTY, 2016) como servidor HTTP e o Jersey (JERSEY, 2016) para a publicação e controle de serviços REST.

No apêndice F na página 147 estão descritas as bibliotecas e ferramentas selecionadas para a construção do protótipo.

A Figura 16 apresenta como exemplo o diagrama UML de implantação de um microsserviço de persistência. O ambiente possui três nós, onde o primeiro nó representa uma aplicação cliente que comunica-se via HTTP com o segundo nó onde está implantado o microsserviço de persistência em banco de dados chave-valor. O segundo nó comunica-se via TCP/IP com o terceiro nó onde encontra-se implantado o gerenciador de banco de dados. A comuncação entre o segundo e terceiro nó utiliza TCP/IP devido aos protocolos proprietários dos *drivers* de conexão com o banco de dados.

A descrição do serviço REST foi criado usando a especificação da *Open API Iniciative* (2016) . Esta especificação busca descrever os serviços REST de uma maneira agnóstica à tecnologias e linguagens e, para tal, sua descrição é feita com o uso de YAML, uma linguagem "*human-friendly*"[9] que, no lugar de marcações, utiliza indentação para a representação de valores, listas e mapas (EVANS, 2009). A descrição textual desta especificação encontra-se no apêndice C na página 135.

Figura 16 – Implantação de microsserviço único

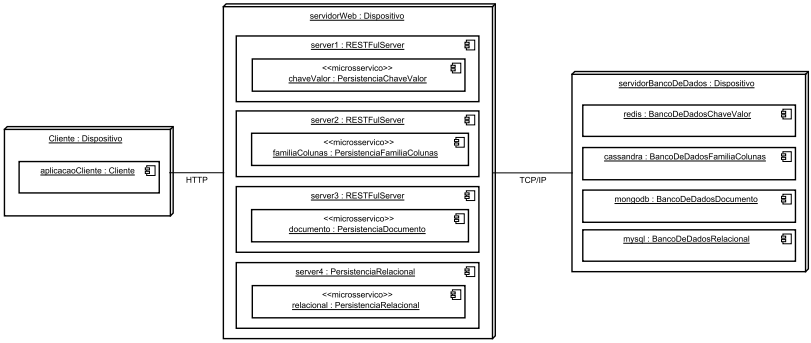


Fonte: produção do próprio autor, 2016.

[9] Expressão que indica que algo é facilmente compreensível por humanos, e não somente máquinas.

Com a utilização de microsserviços é possível também a implantação de vários serviços num único nó (servidor). A Figura 17 apresenta um diagrama UML de implantação onde vários serviços estão instalados e em execução em um mesmo servidor.

Figura 17 – Implantação de microsserviço múltiplo



Fonte: produção do próprio autor, 2016.

O capítulo 5 descreve os experimentos feitos com o protótipo aqui descrito.

5

EXPERIMENTOS, RESULTADOS E DISCUSSÕES

Nas seções a seguir encontram-se os experimentos. Iniciando pela seleção de sistemas gerenciadores de banco de dados, pela análise comparativa de complexidade de implementação.

Para a execução dos experimentos, a primeira etapa foi a de seleção dos sistemas gerenciadores de bancos de dados por meio de regras definidas. As regras e resultados estão descritos na Seção 5.1.

O primeiro experimento foi efetuado sobre a construção das classes DAO, avaliando a diferença de complexidade de códigos fontes na utilização de diferentes modelos de dados. A análise dos resultados podem ser observados na Seção 5.2.

O segundo experimento consistiu de importação de *datasets* de trajetórias disponibilizados publicamente, avaliando se o esquema de dados proposto atende ao requisitos impostos por estes *datasets*. Além disto foi medido o tempo de importação e espaço ocupado. Os *datasets* selecionados e os resultados da experiências estão descritos na Seção 5.3. As experiências com a construção de componentes estão descritas na Seção 5.4. E por fim, a Seção 5.5 apresenta a análise dos resultados.

5.1 Seleção dos sistemas gerenciadores de bancos de dados

A fim de executar os experimentos foi escolhido um sistema de gerenciamento de banco de dados de cada um dos modelos utilizados pela camada de persistência poliglota da arquitetura, sendo estes modelos: Chave-valor, documento, família de colunas e relacional.

Os critérios utilizados para a escolha dos sistemas de gerenciamento de banco de dados são os seguintes:

1. Poder ser utilizado sem a necessidade do pagamento de licenças de uso;

2. Possuir um *driver* de conexão para a linguagem Java;

3. Possuir a maior popularidade de uso dentro da sua categoria de modelo de dados com base nas estatísticas do site DB-Engines (DB-ENGINES, 2016a).

De acordo com os critérios de seleção, o quadro 4 apresenta os bancos de dados selecionados para a avaliação com suas respectivas versões e o indicador de popularidade de Janeiro de 2016. A metodologia utilizada pelo *site* DB-Engines para calcular o indicador de popularidade utiliza os seguintes dados (DB-ENGINES, 2016b):

- O número de resultados nos *sites* de busca Google e Bing;

- Frequência de pesquisas no Google Trends[1];

- Frequência de discussões técnicas nos *sites* Stack Overflow e DBA Stack Exchange;

- Número de empregos ofertados pelos sites Indeed e Simply Hired que pedem conhecimento no banco de dados;

[1] Serviço do Google que apresenta a quantidade de vezes que uma expressão foi usada no seu serviço de buscas.

- Número de perfis no LinkedIn[2] que citam o banco de dados;
- Número de *posts* no Tweeter[3] sobre o banco de dados.

Quadro 4 – SGBD's utilizados nas experiências

Modelo	Produto	Versão	Pop.
Relacional	MySQL	5.7	1.299
Chave-Valor	Redis	3.0.5	101
Documento	MongoDB	3.2	306
Família de Colunas	Cassandra	3.2	130

Fonte: DB-Engines, 2016.

5.2 Análise de complexidade de implementação de DAO

A primeira análise feita foi avaliar a complexidade de programação dos componentes de acesso aos dados (DAO) para cada um dos modelos de dados utilizados pela arquitetura. Esta avaliação teve como intuito comparar a complexidade de implementação dos componentes de acesso a dados dos bancos NoSQL em relação ao do banco relacional. Um dos maiores obstáculos na utilização de um novo modelo ou banco de dados é o aprendizado desta nova tecnologia e a complexidade de implementação que ela oferece. Na literatura, maioria dos trabalhos publicados apenas avaliam aspectos de desempenho dos bancos de dados NoSQL, não avaliando a complexidade de programação (LI; MANOHARAN, 2013; VEEN; WAAIJ; MEIJER, 2012; CATTELL, 2011; TUDORICA; BUCUR, 2011; STONEBRAKER, 2010).

5.2.1 Especificação da avaliação

Para efetuar a avaliação foi utilizada a ferramenta de medição de qualidade Sonarqube, que mede a qualidade do código

[2] Rede social de profissionais e negócios.
[3] Rede social de *microblogging*.

fonte fornecendo a extração de várias métricas (SONARQUBE, 2016).

Para evitar que a complexidade na implementação do DAO para o banco relacional seja mascarada pela utilização de um *framework*, neste experimento todo o código de mapeamento objeto-relacional e acesso ao *driver* foi implementado manualmente, sem a utilização de ferramentas como Hibernate, pois isso iria transferir a complexidade para dentro desta ferramenta, invalidando o comparativo.

Para simplificar o processo de avaliação o objeto a ser persistido contém apenas dois atributos (identificação e descrição). Isso faz com que a medição da complexidade enfatize o código de persistência.

Para medir a complexidade dos componentes de persistência as seguintes métricas foram avaliadas:

- LoC, número de linhas de código necessárias para a implementação das funções do componente;
- MCC, complexidade ciclomática de McCabe, que é medida pelo grafo de fluxo de controle, onde são apresentadas as linhas possíveis de execução (MCCABE, 1976).

5.2.2 Resultado da complexidade

Após a execução da ferramenta de qualidade Sonarqube, os resultados obtidos são apresentados na Tabela 1 na próxima página. Como pode ser observado na tabela, a implementação do DAO para o modelo relacional foi o que apresentou a maior quantidade de linhas de código (LoC) e a maior complexidade ciclomática (MCC). Para fins de comparação, o modelo relacional foi definido como 100% de complexidade e os demais modelos como um percentual relativo a ele, dado que os outros modelos apresentam complexidade menor.

É possível observar que com o uso dos modelos NoSQL a complexidade ciclomática do código-fonte reduziu em média 32% em relação ao código-fonte de acesso ao modelo relacional. Isso é

explicado pelo fato do modelo relacional separar os dados agregados em diferentes relações por causa da normalização. Portanto, é necessário implementar código para armazenar cada parte do agregado em uma relação diferente. Já os modelos NoSQL armazenam os dados de forma agregada. Assim, independentemente da quantidade de agregados que o dado tiver, só será necessária uma função para fazer o armazenamento.

Tabela 1 – Métricas de complexidade

Modelo	**LoC***	**MCC***	**%***
Chave-Valor	81	14	45,16
Colunar	130	21	67,74
Documento	119	23	74,19
Relacional	175	31	100,00

Fonte: produção do próprio autor, 2016.
*Menor melhor

5.3 Importação de *datasets* de trajetórias

O segundo experimento efetuado avalia a aderência do esquema de dados proposto na seção 4.2 aos *datasets* de trajetórias disponíveis publicamente. Para esta avaliação foram utilizados os *Datasets* mais utilizados por outros projetos de análises de trajetórias e que sejam heterogêneos entre si a fim de avaliar a robustez e generalização do esquema proposto. Por este motivo foram selecionados *datasets* de trajetórias com diferentes tipos de meios de transporte (ônibus, táxi, avião, a pé, etc.) e com diferente quantidade de atributos. Além da aderência ao modelo, foram avaliados também tempo de importação e espaço ocupado em disco pelos dados.

Conforme Renear, Sacchi e Wickett (2010), um *dataset* é um conjunto de dados semanticamente relacionados com o propósito de contribuir de alguma maneira com uma atividade científica e que possui quatro características principais: agrupamento, conteúdo, relacionamento e finalidade.

5.3.1 *Datasets*

Os três *datasets* selecionados para os experimentos são: GeoLife, T-Drive e ônibus da cidade do Rio de Janeiro. As próximas subseções detalham cada um destes *datasets*.

5.3.1.1 GeoLife

O GeoLife é um serviço de rede social baseada em localização que permite aos usuários compartilhar experiências de vida e construir conexões entre si usando histórico de localização (GEOLIFE, 2007; ZHENG et al., 2008; ZHENG; XIE; MA, 2010).

Os dados deste projeto foram coletados por 182 usuários durante o período de 5 anos (de abril/2007 a agosto/2012) e foram disponibilizados para uso não comercial. Cada trajetória é representada por um sequência de pontos com seu *timestamp*, latitude, longitude e altitude. O total de dados disponibilizados é de 17.621 trajetórias com uma distância total de 1.251.654 km e uma duração total de 48.203 horas. Vale destacar que apenas nos últimos meses de coleta do ano de 2012 foi incluída a informação do meio de transporte.

5.3.1.2 T-Drive

O T-Drive é um serviço inteligente de direção com base em trajetórias GPS oriundas de um grande número de táxis (T-DRIVE, 2010; YUAN et al., 2010; YUAN et al., 2011). O objetivo deste serviço é ajudar os usuários a escolher o melhor caminho baseado no horário de saída e no local de destino.

Este projeto disponibiliza um subconjunto do seu *dataset* contendo os dados de uma semana de trajetórias de 10.357 táxis da cidade de Pequim e possui 15 milhões de pontos abrangendo a distância de 9 milhões de quilômetros (ZHENG, 2011).

5.3.1.3 Dados de ônibus da cidade do Rio de Janeiro

Por meio da iniciativa Dados Abertos da prefeitura da cidade do Rio de Janeiro, mantida pela Federação das Empresas de Transporte de Passageiros do Estado do Rio de Janeiro (FETRANSPOR), é possível acessar os dados de localização dos ônibus que circulam na região metropolitana da cidade do Rio de Janeiro (DATA.RIO, 2014). Este serviço fornece também os dados referentes à velocidade e direção dos ônibus. Esta informação não é fornecida pelos outros dois *datasets* utilizados.

Como o *site* não oferece para *download* os dados históricos dos ônibus, foi criado um programa que de minuto em minuto fazia uma requisição ao *site* pedindo a posição atual dos ônibus da cidade. Foram coletados dados durante 35 dias. Para os experimentos foram usados os dados coletados do dias 11 a 20 de Novembro de 2015 com os dados de 7356 ônibus com 55.387.451 pontos coletados resultando em 71.040 trajetórias.

Um problema encontrado na construção deste *dataset* foi identificar o início e fim de cada trajetória dos ônibus, pois os dados são publicados ininterruptamente enquanto o ônibus estiver em operação. Como solução, foi definido que a cada novo dia uma nova trajetória era gerada. Assim, caso o próximo ponto coletado pertencer ao dia posterior, uma nova trajetória é iniciada para o ônibus em questão.

5.3.2 Resultados do experimento de importação dos *datasets*

Devido ao requisito de *Big Data*, o espaço ocupado pelos *datasets* no disco passa a ser relevante. Apesar do uso de NoSQL facilitar a utilização de *shards*[4], o espaço ocupado em disco é uma informação importante para definir a arquitetura de processamento de dados de trajetórias.

Para esta avaliação, os servidores de bancos de dados foram "zerados", e cada *dataset* foi importado individualmente usando o serviço de persistência implementado para cada um dos

[4] É dado o nome de *shard* aos fragmentos horizontais de um banco de dados distribuído.

modelos. A Tabela 2 apresenta os resultados quanto ao espaço ocupado em disco, a Tabela 3 quanto ao tempo de importação, e a Tabela 4 quanto à quantidade de trajetórias importadas por segundo.

Tabela 2 – Espaço ocupado em disco

Dataset	**Chv-valor**	**Docum.**	**Colunar**	**Relac.**
GeoLife	355 mb	547 mb	313 mb	3.523 mb
BusRJ	1.236 mb	1.736 mb	1.464 mb	10.227 mb
T-Drive	256 mb	441 mb	306 mb	104.272 mb

Fonte: produção do próprio autor, 2016.

Tabela 3 – Tempo de importação

Dataset	**Chv-valor**	**Docum.**	**Colunar**	**Relac.***
GeoLife	7 min	13 min	12 min	5 dias
BusRJ	16 min	104 min	32 min	4,6 dias
T-Drive	6 min	8 min	14 min	39 dias

Fonte: produção do próprio autor, 2016.
* Tempo estimado através de regra de três

Tabela 4 – Trajetórias importadas por segundo

Dataset	**Chv-valor**	**Docum.**	**Colunar**	**Relac.**
GeoLife	44	23	26	<1
BusRJ	74	11	37	<1
T-Drive	191	143	82	<1

Fonte: produção do próprio autor, 2016.

Quadro 5 – Configuração do computador usado nas experiências

Fabricante	Hewlett-Packard
CPU	Intel Core i5 1,7Ghz/2,4Ghz
Memória	8 Gb DDR3
S.O.	Windows 10 64 bits v.1511
H.D.	1TB 5400RPM 8MB Cache SATA

Fonte: produção do próprio autor, 2016.

5.3.3 Considerações

O processo de importação dos três *datasets* foi executado em todos os quatro gerenciadores de banco de dados sem qualquer necessidade de adaptação ao modelo proposto, bastando enviar os dados da trajetória para o serviço REST disponibilizado. Desta forma, é possível inferir que o esquema de dados projetado para a persistência de trajetórias atendeu os requisitos de flexibilidade.

5.3.3.1 Vantagens e desvantagens identificadas dos modelos de dados

Com a importação dos *datasets* nos quatro modelos avaliados foi possível determinar as seguintes vantagens e desvantagens na utilização dos modelos NoSQL no armazenamento de dados de trajetórias de objetos móveis.

- Vantagens:
  1. Melhor aderência do esquema conceitual com o modelo de dados documento.
  2. Melhora considerável no desempenho do armazenamento dos dados de trajetória.
- Desvantagens:
  1. Ainda há pouco ferramental disponível que suporte a análise de trajetórias em modelo NoSQL. Recentemente a empresa Oracle lançou um produto chamado

Oracle Big Data Spatial and Graph que suporta analise de dados NoSQL através de funções geoespaciais (ORACLE, 2015).

5.4 Componentes

Para o experimento de construção e reaproveitamento de componentes foram selecionados dois algoritmos de processamento de dados de trajetórias de objetos móveis: Segmentação por pontos e cálculo de azimute[5].

5.4.1 Segmentação por pontos

Para o primeiro experimento foi implementado o algoritmo que recebe uma trajetória e a separa em segmentos compostos por dois pontos. O algoritmo faz a iteração entre os n pontos da trajetória gerando $n-1$ segmentos. Esse processo gera um novo estado da trajetória que, diferentemente da anterior - que tinha apenas um segmento com n pontos -, tem $n-1$ segmentos compostos por 2 pontos. Este código foi escrito como um programa procedimental a fim de simular a criação de um protótipo utilizado para uma experiência em específico.

Após isto foi criado um microsserviço que encapsulou o código de segmentação criando a interface REST que permite o acesso remotamente via a arquitetura. O processo de encapsular e executar o processo de segmentação ocorreu conforme o esperado permitindo assim a sua reutilização em futuras análises de trajetórias. Este primeiro componente simulou a utilização de um processo sem a utilização da camada de persistência, onde um cliente apenas faz a segmentação de um conjunto de pontos de uma trajetória que é passada como *payload* para o serviço e recebe como resposta a trajetória segmentada. Os códigos fonte desta experiência estão no apêndice D na página 141.

[5] Azimute é uma medida de direção horizontal, definida em graus.

5.4.2 Enriquecer os segmentos com o valor do azimute

Para o segundo experimento foi implementado o algoritmo que realiza o cálculo de azimute de um segmento composto por dois pontos. Portanto, para que os segmentos de uma trajetória possam ser enriquecidos com o valor do azimute, é necessário que ela tenha sido previamente segmentada. Para tal, foi utilizado o componente (microsserviço) criado no experimento anterior para segmentar uma trajetória de dois em dois pontos.

Para o armazenamento desta informação extra foi utilizada a entidade de dados adicionais chamada "SegmentoDado" similar ao utilizado no experimento com os dados dos ônibus do Rio de Janeiro, com a diferença que naquele caso o dado pertence ao ponto e portanto foi armazenando no "PontoDado".

Para realizar o experimento foi criado um microsserviço que requisita os dados de trajetória da camada de persistência, faz uma iteração nos segmentos, executa o método que calcula o azimute e o registra nos dados adicionais criando assim um novo estado da trajetória. Ao final, o novo estado da trajetória é enviada para a persistência e para a aplicação cliente. A criação deste microsserviço seguiu a mesma abordagem de implementação do serviço anterior (de segmentação). A única diferença é que neste experimento foi utilizado o serviço de descoberta e monitoramento de serviços implementado pela ferramenta Eureka. Por este motivo, os serviços foram registrados no serviço de descoberta e monitoramento e sua utilização foi feita após consulta a serviço de descoberta e posterior resolução do endereço de publicação da interface do serviço.

Nesta segunda experiência ficou evidenciada a comunicação entre processamento e persistência e o processo de enriquecimento dos dados, além da utilização do serviço de descoberta e monitoramento. Uma observação relevante evidenciada pelos experimentos é que os demais trabalhos relacionados revisados no capítulo 3 não consideram a existência de segmentos nas trajetórias. Isso faz com que sejam necessárias várias requisições e execuções para a identificação dos segmentos da trajetória.

5.5 Análise de Resultados

Através das experiências os seguintes resultados foram observados:

5.5.1 Requisitos dos dados

Os seguintes requisitos foram levantados com relação ao dados e apresentados na seção 4.1.1 na página 66:

5.5.1.1 Suportar dados estruturados e não estruturados de uma trajetória

Através das experiências efetuadas com os dados dos ônibus dos Dados Abertos da cidade do Rio de Janeiro e da experiência com o cálculo e armazenamento do valor do azimute dos segmentos, foi evidenciado que a camada de persistência atendeu a este requisito através do armazenamento dos dados mínimos estruturados e dados adicionais semi-estruturados.

5.5.1.2 Suportar dados de trajetórias em diferentes estados

Através das experiências de importação de dados de diferentes *datasets* e das experiências de geração de segmentos e cálculo do valor do azimute, foi evidenciado que a camada de persistência atendeu a este requisito, pois as trajetórias foram armazenadas em diferentes estados (estado 1: bruta, estado 2: segmentada e estado 3: com valor de azimute).

5.5.1.3 Suportar *Big Data*

Através das experiências de importação de diferentes *datasets* foi possível observar que o tempo de importação foi consideravelmente menor através da utilização de bancos de dados NoSQL. Isso leva a inferir que mesmo em situações onde o volume de dados seja muito superior ao testado, as soluções NoSQL terão capacidade de suportar o armazenamento com tempo de resposta aceitável.

5.5.2 Requisitos de aplicação

Os seguintes requisitos foram levantados com relação à camada de negócios e apresentados na seção 4.1.2 na página 69:

5.5.2.1 Interoperabilidade com novos serviços

Pelas experiências relacionadas à inclusão dos serviços de segmentação e cálculo de azimute, foi possível demonstrar que com a utilização de microsserviços é possível simplificar o desenvolvimento e implantação de serviços em comparação com servidores tradicionais de soluções SOA. Estes experimentos também mostram que o uso do protocolo REST com dados no formato JSON facilita a interoperabilidade entre eles, pois estas são tecnologias abertas e não proprietárias a um conjunto restrito de tecnologias.

5.5.2.2 Descoberta de serviços

No experimento de cálculo de azimute, o microsserviço fez o registro no componente de descoberta e monitoramento de serviços (Eureka). A aplicação cliente consultou primeiramente este serviço para obter as informações quanto ao endereço e a disponibilidade do serviço de azimute para responder às requisições. Através desta experiência, o requisito de descoberta de serviços requirido foi avaliado e atendido.

5.5.2.3 Escalabilidade horizontal e computação elástica

A escalabilidade horizontal na camada de persistência é alcançada por meio da utilização de bancos de dados NoSQL que são projetados para serem escalados naturalmente da maneira horizontal.

Com a utilização do padrão de microsserviços que permite a criação de várias instâncias de um mesmo serviço em diversos nós em combinação com o serviço de descoberta, que também faz o balanceamento de carga, implementado pela ferramenta Eureka, a elasticidade pode ser alcançada.

5.5.3 Limitações da solução

O presente trabalho propôs um esquema único de dados de trajetória para os quatro modelos de dados avaliados (chave-valor, documento, família de colunas e relacional). Entretanto, devido à utilização particularmente do modelo relacional, o suporte aos dados não estruturados se tornou mais complexo. O fato do modelo relacional não suportar atributos dinâmicos e tipos de dados variáveis, torna necessária a definição de entidades genéricas na forma de tipo cadeia de caracteres (*String*).

5.5.4 Resultados gerais

Pelo exposto acima é possível observar que a arquitetura proposta serve como base para a criação e implantação de serviços para aplicações que processem trajetórias de objetos móveis. A reutilização de processos foi alcançada pela adoção da abordagem de microsserviços. A flexibilização do esquema de dados para armazenamento de dados semi estruturados em conjunto com os dados estruturados das trajetórias dos objetos móveis foi viabilizada pelo esquema de dados proposto e mapeado para quatro diferentes modelos de dados. Também puderam ser observados ganhos de produtividade devido à redução de complexidade de código dos algoritmos de persistência para os modelos NoSQL e do reuso de componentes via o encapsulamento deles em serviços.

6

CONSIDERAÇÕES FINAIS

Neste capítulo são apresentadas as conclusões, contribuições e propostas de trabalhos futuros.

6.1 Conclusões

Devido à proliferação dos dispositivos eletrônicos móveis munidos de sensores GPS a quantidade de dados de trajetórias de objetos móveis produzidas vêm aumentando exponencialmente, enquadrando a análise de trajetórias de objetos móveis como um problema de *Big Data*. Pelekis e Theodoridis (2014) já haviam previsto este cenário.

Considerando que aplicações que necessitam processar grandes volumes de dados podem se beneficiar das características providas pelos bancos de dados NoSQL, estes benefícios poderiam ser também explorados no contexto de dados de trajetórias.

Portanto, este trabalho teve por objetivo prover uma arquitetura SDI orientada a serviços que possibilite a criação e implantação de serviços para aplicações que processem trajetórias de objetos móveis dando também o suporte ao casos de *Big Data*. O intuito era facilitar a reutilização e escalabilidade dos processos e prover uma camada de persistência poliglota que possibilite o armazenamento de dados semi estruturados associados às tra-

jetórias.

Para alcançar este objetivo foi projetado um repositório de persistência poliglota que armazena trajetórias dos objetos móveis de maneira a ser possível fazer uso das vantagens providas pelos bancos de dados NoSQL. Além disso, foi criado uma arquitetura SDI orientada a serviços utilizando do padrão de microsserviço que facilita o reuso e a escalabilidade/elasticidade dela.

Os experimentos realizados apontam que o arcabouço computacional desenvolvido para a implantação de serviços reutilizáveis suporta o armazenamento e processamento de grandes volumes de trajetórias. Além disso, o esquema de dados proposto possibilita o armazenamento de trajetórias com dados heterogêneos de forma transparente. Estas características providas pelo arcabouço habilitam alavancar o processamento de dados de trajetórias de objetos móveis e suportam as necessidades atuais demandadas pela proliferação de trajetórias geradas pela aumento da utilização de dispositivos móveis.

6.2 Contribuições

Os seguintes itens de pesquisa são considerados contribuições do presente trabalho: i) Modelo de dados proposto suportando trajetórias segmentadas e dados extras semi e não estruturados; ii) a utilização de armazenamento poliglota de trajetórias e a construção de um *framework* SDI baseado em SOA com o padrão microsserviços, no domínio de processamento de dados de trajetórias de objetos móveis.

6.3 Propostas para trabalhos futuros

Com os resultados obtidos com esta pesquisa, observou-se que outras características relacionadas ao problema em questão podem ser exploradas com o objetivo de melhorar a solução proposta e a nortear novas pesquisas. A seguir são apresentadas algumas propostas de trabalhos futuros.

O modelo de dados em grafo não foi abordado no presente trabalho, mas ele demonstra potencial no armazenamento de mapas e no relacionamento de pontos de interesse. Portanto, um dos trabalhos futuros tem por objetivo adicionar o suporte ao banco de dados modelo grafo com um modelo desenhado especificamente para suportar outros tipos de pesquisas em dados de objetos móveis, não somente o de trajetórias.

Como o modelo de dados documento se mostrou mais adequado e correspondente com o modelo proposto de dados, é interessante aprofundar as pesquisas utilizando exclusivamente o modelo documento. Portanto, seria proposto um trabalho futuro que dê foco a este modelo avaliando a utilização de suas funções geoespaciais no armazenamento de dados de objetos móveis (FILHO et al., 2015; BAPTISTA et al., 2014; CHANG, 2014).

Recentemente surgiram gerenciadores de bancos de dados NoSQL multi-modelo que, como o próprio nome sugere, agregam mais de um modelo de dados. Exemplos de sistemas gerenciadores de bancos de dados NoSQL multi-modelo são o OrientDB[1] e o ArangoDB[2] que unificam os modelos documento e grafos. Portanto, é proposto um trabalho futuro para avaliar a persistência poliglota de dados de trajetórias de objetos móveis utilizando um gerenciador de banco de dados NoSQL multi-modelo.

[1] <http://orientdb.com>
[2] <http://www.arangodb.com>

Referências

ALMEIDA, C. A. de S.; PIRES, C. E.; SCHIEL, U. Um modelo de dados para trajetórias de objetos móveis com suporte a agregação de movimentos. *iSys-Revista Brasileira de Sistemas de Informação*, v. 4, 2012.

ALUR, D.; MALKS, D.; CRUPI, J.; BOOCH, G.; FOWLER, M. *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*. [S.l.]: Sun Microsystems, Inc., 2003. ISBN 9780131422469.

AMBLER, S. W. The object-relational impedance mismatch. *Agile Database Techniques, Wiley Publishing*, 2003. Acesso em: 26 mar. 2016. Disponível em: <http://www.agiledata.org/essays/impedanceMismatch.html>.

AMIRIAN, P.; ALESHEIKH, A. A service oriented framework for disseminating geospatial data to mobile, desktop and web clients. *World Appl. Sci. J*, v. 3, n. 1, p. 140–153, 2008.

AMIRIAN, P.; ALESHEIKH, A. A. Implementation of a geospatial web service using web services technologies and native xml databases. *Middle-East Journal of Scientific Research*, v. 3, n. 1, p. 36–48, 2008.

AMIRIAN, P.; BASIRI, A.; WINSTANLEY, A. Implementing geospatial web services using service oriented architecture and NoSQL solutions. In: SDIWC. *The Third International Conference on Digital Information and Communication Technology and its Applications (DICTAP2013)*. [S.l.], 2013. p. 161–169.

ANDRIENKO, G.; ANDRIENKO, N.; BAK, P.; KEIM, D.; WROBEL, S. *Visual analytics of movement*. [S.l.]: Springer Publishing Company, Incorporated, 2013. ISBN 9783642375835.

ANDRIENKO, N.; ANDRIENKO, G. Visual analytics of movement: An overview of methods, tools and procedures. *Information Visualization*, Sage Publications, p. 3–24, 2012.

BAPTISTA, C. de S.; PIRES, C. E. S.; LEITE, D. F. B.; OLIVEIRA, M. G. de. NoSQL geographic databases: an overview. *Geographical Information Systems: Trends and Technologies*, CRC Press, v. 73, 2014.

BOULMAKOUL, A.; KARIM, L.; LBATH, A. Moving object trajectories meta-model and spatio-temporal queries. *arXiv preprint arXiv:1205.1796*, 2012.

BOULMAKOUL, A.; KARIM, L.; MANDAR, M.; IDRI, A.; DAISSAOUI, A. Towards scalable distributed framework for urban congestion traffic patterns warehousing. *Applied Computational Intelligence and Soft Computing*, Hindawi Publishing Corporation, v. 2015, 2015.

BRAZ, F. J.; BOGORNY, V. *Introdução a trajetórias de objetos móveis*. [S.l.]: Editora Univille, 2012. ISBN 9788582090022.

BREWER, E. Cap twelve years later: How the "rules" have changed. *Computer*, IEEE, v. 45, n. 2, p. 23–29, 2012.

BREWER, E. A. Towards robust distributed systems. In: *PODC*. [S.l.: s.n.], 2000. v. 7.

BURKE, B. *RESTful Java with JaX-RS*. [S.l.]: O'Reilly Media, Inc., 2009. ISBN 9781449361341.

CARBONI, E. M. *Análise De Comportamento De Motoristas Através De Trajetórias De Objetos Móveis*. Dissertação (Mestrado) — Universidade Federal De Santa Catarina, Outubro 2014.

CATTELL, R. Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, ACM, v. 39, n. 4, p. 12–27, 2011.

CHANG, C. C.-K. C. *A Primer on Geospatial Data and MongoDB*. 2014. Acesso em: 05 fev. 2016. Disponível em: <http://blog.mongolab.com/2014/08/a-primer-on-geospatial-data-and-mongodb/>.

CISCO, V. N. I. *The zettabyte era–trends and analysis*. [S.l.], 2015.

COSTA, G. H. R. *Geração De Mapas Rodoviários a Partir De Trajetórias De Objetos Móveis Coletadas Por Smartphone – Método Baseado Em Algoritmo Genético*. Dissertação (Mestrado) — Universidade do Estado de Santa Catarina, Fevereiro 2014.

DATA.RIO. *GPS dos ônibus*. 2014. Acesso em: 20 jan. 2016. Disponível em: <http://data.rio/dataset/gps-de-onibus>.

DB-ENGINES. *DB-Engines, Knowledge Base of Relational and NoSQL Database Management Systems*. 2016. Acesso em: 20 jan. 2016. Disponível em: <http://db-engines.com/en/>.

DB-ENGINES. *Method of calculating the scores of the DB-Engines Ranking*. 2016. Acesso em: 23 jan. 2016. Disponível em: <http://db-engines.com/en/ranking_definition>.

DORBAND, J.; PALENCIA, J.; RANAWAKE, U. Commodity computing clusters at goddard space flight center. *Journal of Space Communication*, v. 1, n. 3, p. 113–123, 2003.

DOUGLAS, D. H.; PEUCKER, T. K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, University of Toronto Press, v. 10, n. 2, p. 112–122, 1973.

DROPWIZARD. *Dropwizard Production-ready, out of the box.* 2016. Acesso em: 02 fev. 2016. Disponível em: <http://www.dropwizard.io>.

ECKERSON, W. W. Three tier client/server architectures: achieving scalability, performance, and efficiency in client/server applications. *Open Information Systems*, v. 3, n. 20, p. 46–50, 1995.

EDEN, A. H. Three paradigms of computer science. *Minds and machines*, Springer, v. 17, n. 2, p. 135–167, 2007.

EDLICH, S. *NoSQL Definition.* 2015. Acesso em: 23 set. 2015. Disponível em: <http://nosql-database.org>.

ELMASRI, R.; NAVATHE, S. B. Sistemas de banco de dados. Pearson Education do Brasil, 2011.

ERL, T. *Soa: principles of service design.* [S.l.]: Prentice Hall Upper Saddle River, 2008. ISBN 978013234482.

ERL, T.; CARLYLE, B.; PAUTASSO, C.; BALASUBRAMANIAN, R. *SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST.* [S.l.]: Prentice Hall Press, 2012. ISBN 0076092045267.

EVANS, C. C. *YAML Ain't Markup Language (YAML^{TM}) Version 1.2.* 2009. Acesso em: 27 jan. 2016. Disponível em: <http://yaml.org/spec/1.2/spec.html>.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures.* Tese (Doutorado) — University of California, Irvine, 2000.

FILETO, R.; MAY, C.; RENSO, C.; PELEKIS, N.; KLEIN, D.; THEODORIDIS, Y. The baquara^2 knowledge-based framework for semantic enrichment and analysis of movement data. *Data & Knowledge Engineering*, Elsevier, v. 98, p. 104–122, 2015.

FILHO, W. B.; OLIVERA, H. V.; HOLANDA, M.; FAVACHO, A. A. Geographic data modeling for NoSQL document-oriented databases. In: IARIA. *GEOProcessing 2015 : The Seventh International Conference on Advanced Geographic Information Systems, Applications, and Services*. [S.l.], 2015.

FILIPPO, D.; PIMENTEL, M.; WAINER, J. Metodologia de pesquisa científica em sistemas colaborativos. *Sistemas Colaborativos*, v. 1, p. 379–404, 2011.

FORD, N. *Polyglot programming*. 2006. Acesso em: 20 nov. 2014. Disponível em: <http://memeagora.blogspot.com/2006/12/polyglot-programming.html>.

FOWLER, M. *Patterns of enterprise application architecture*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0076092019909.

FOWLER, M. *Richardson Maturity Model: steps toward the glory of REST*. 2010. Acesso em: 08 set. 2015. Disponível em: <http://martinfowler.com/articles/richardsonMaturityModel.html>.

FOWLER, M. *Microservice Trade-Offs*. 2015. Acesso em: 22 jan. 2016. Disponível em: <http://martinfowler.com/articles/microservice-trade-offs.html>.

GEOLIFE. *GeoLife: Building social networks using human location history*. 2007. Acesso em: 20 jan. 2016. Disponível em: <http://research.microsoft.com/en-us/projects/geolife/>.

GILBERT, S.; LYNCH, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, ACM, v. 33, n. 2, p. 51–59, 2002.

GRAY, J. The transaction concept: Virtues and limitations. In: *VLDB*. [S.l.: s.n.], 1981. v. 81, p. 144–154.

HAJARI, H.; HAKIMPOUR, F. A spatial data model for moving object databases. *International Journal of Database Management Systems (IJDMS)*, v. 6, n. 1, 2014.

HEDGES, L. V. How hard is hard science, how soft is soft science? the empirical cumulativeness of research. *American Psychologist*, American Psychological Association, v. 42, n. 5, p. 443, 1987.

HIBERNATE. *Hibernate. Everything data.* 2016. Acesso em: 02 fev. 2016. Disponível em: <http://hibernate.org>.

HOBERMAN, S. *Data Modeling for MongoDB: Building Well-Designed and Supportable MongoDB Databases*. [S.l.]: Technics Publications, 2014. ISBN 9781935504702.

HOHPE, G.; WOOLF, B. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. [S.l.]: Addison-Wesley Professional, 2003. ISBN 9780321200686.

HURWITZ, J.; NUGENT, A.; HALPER, F.; KAUFMAN, M. *Big data for dummies*. [S.l.]: John Wiley & Sons, 2013. ISBN 9781118504222.

ITU. *Rec. ITU-T Y.3600 Big data – cloud computing based requirements and capabilities*. [S.l.]: The International Telecommunication Union, 2015.

JACKSON. *Jackson Project*. 2016. Acesso em: 02 fev. 2016. Disponível em: <http://wiki.fasterxml.com/JacksonHome>.

JERSEY. *Jersey - RESTful Web Services in Java.* 2016. Acesso em: 02 fev. 2016. Disponível em: <http://jersey.java.net>.

JETTY. *Jetty - Servlet Engine and Http Server*. 2016. Acesso em: 02 fev. 2016. Disponível em: <http://www.eclipse.org/jetty/>.

KARIMI, H. A. *Big Data: Techniques and Technologies in Geoinformatics*. [S.l.]: CRC Press, 2014. ISBN 9781466586512.

LAKATOS, E. M.; MARCONI, M. d. A. Fundamentos da metodologia científica. In: *Fundamentos da metodologia científica*. [S.l.]: Altas, 2010.

LEBERKNIGHT, S. *Polyglot Persistence*. 2008. Acesso em: 20 nov. 2014. Disponível em: <http://www.nearinfinity.com/blogs/scott_leberknight/polyglot_persistence.html>.

LEWIS, J. Micro services – java the unix way. In: 33RD DEGREE. *33rd Degree Conference 2012*. [S.l.], 2012.

LEWIS, J.; FOWLER, M. *Microservices*. 2014. Acesso em: 19 out. 2015. Disponível em: <http://martinfowler.com/articles/microservices.html>.

LI, Y.; MANOHARAN, S. A performance comparison of SQL and NoSQL databases. In: IEEE. *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*. [S.l.], 2013. p. 15–19.

MARINO, J.; ROWLEY, M. *Understanding SCA (Service Component Architecture)*. [S.l.]: Pearson Education, 2009.

MARTIN, R. C. *Agile software development: principles, patterns, and practices*. [S.l.]: Prentice Hall PTR, 2003. ISBN 9780135974445.

MAURO, T. *Adopting Microservices at Netflix: Lessons for Architectural Design*. 2015. Acesso em: 08 jan. 2016. Disponível em: <https://dzone.com/articles/adopting-microservices-netflix>.

MCCABE, T. J. A complexity measure. *Software Engineering, IEEE Transactions on*, IEEE, n. 4, p. 308–320, 1976.

MCCREARY, D. G.; KELLY, A. M. *Making Sense of NoSQL - A guide for managers and the rest of us*. [S.l.]: Manning Publications, 2013. ISBN 9781617291074.

MONIRUZZAMAN, A.; HOSSAIN, S. A. NoSQL database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*, 2013.

MUEHLEN, M. Z.; NICKERSON, J. V.; SWENSON, K. D. Developing web services choreography standards—the case of rest vs. soap. *Decision Support Systems*, Elsevier, v. 40, n. 1, p. 9–29, 2005.

MURAKAMI, E.; SARAIVA, A. M.; RIBEIRO, L. C.; CUGNASCA, C. E.; HIRAKAWA, A. R.; CORREA, P. L. An infrastructure for the development of distributed service-oriented information systems for precision agriculture. *Computers and Electronics in agriculture*, Elsevier, v. 58, n. 1, p. 37–48, 2007.

NETFLIX. *Eureka at a glance*. 2014. Acesso em: 29 jan. 2016. Disponível em: <https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance>.

NEWMAN, S. *Building Microservices*. [S.l.]: O'Reilly Media, Inc., 2015. ISBN 9781491950357.

NURSEITOV, N.; PAULSON, M.; REYNOLDS, R.; IZURIETA, C. Comparison of json and xml data interchange formats: A case study. *Caine*, v. 9, p. 157–162, 2009.

OPEN API INICIATIVE. *Open API Iniciative Specification*. 2016. Acesso em: 27 jan. 2016. Disponível em: <https://openapis.org/specification>.

OPENSTREETMAP. *OpenStreetMap Stats Report*. 2016. Acesso em: 22 jan. 2016. Disponível em: <http://www.openstreetmap.org/stats/data_stats.html>.

ORACLE. *Oracle Big Data Spatial and Graph*. 2015. Acesso em: 26 jan. 2016. Disponível em: <http://www.oracle.com/technetwork/database/database-technologies/bigdata-spatialandgraph>.

PARENT, C.; SPACCAPIETRA, S.; RENSO, C.; ANDRIENKO, G.; ANDRIENKO, N.; BOGORNY, V.; DAMIANI, M. L.; GKOULALAS-DIVANIS, A.; MACEDO, J.; PELEKIS, N. et al. Semantic trajectories modeling and analysis. *ACM Computing Surveys (CSUR)*, ACM, v. 45, n. 4, p. 42, 2013.

PELEKIS, N.; THEODORIDIS, Y. *Mobility Data Management and Exploration*. [S.l.]: Springer, 2014. ISBN 9781493903917.

PRITCHETT, D. Base: An acid alternative. *Queue*, ACM, v. 6, n. 3, p. 48–55, 2008.

REED, C. *OGC Standards and Geospatial Big Data*. [S.l.]: CRC Press, 2014. 279 p. ISBN 9781466586550.

RENEAR, A. H.; SACCHI, S.; WICKETT, K. M. Definitions of dataset in the scientific and technical literature. *Proceedings of the American Society for Information Science and Technology*, Wiley Online Library, v. 47, n. 1, p. 1–4, 2010.

RICHARDS, M. *Microservices vs. Service-Oriented Architecture*. [S.l.]: O'Reilly Media, Inc., 2015. ISBN 9781491952429.

RICHARDSON, C. *Microservice architecture patterns and best practices*. 2015. Acesso em: 10 nov. 2015. Disponível em: <http://microservices.io>.

RICHARDSON, L. Justice will take us millions of intricate moves. In: . [S.l.]: QCon San Francisco, 2008.

RINZIVILLO, S.; PEDRESCHI, D.; NANNI, M.; GIANNOTTI, F.; ANDRIENKO, N.; ANDRIENKO, G. Visually driven analysis of movement data by progressive clustering. *Information Visualization*, SAGE Publications, v. 7, n. 3-4, p. 225–239, 2008.

ROBINSON, I.; WEBBER, J.; EIFREM, E. *Graph databases*. 2. ed. [S.l.]: O'Reilly Media, Inc., 2015. ISBN 9781491930892.

SADALAGE, P. J.; FOWLER, M. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. [S.l.]: Pearson Education, 2012. ISBN 9780321826626.

SANOU, B. The world in 2015: Ict facts and figures. *International Telecommunications Union*, 2015.

SONARQUBE. *SonarQube*. 2016. Acesso em: 24 jan. 2016. Disponível em: <http://www.sonarqube.org/>.

STONEBRAKER, M. SQL databases v. NoSQL databases. *Communications of the ACM*, ACM, v. 53, n. 4, p. 10–11, 2010.

STONEBRAKER, M.; CETINTEMEL, U. "one size fits all": an idea whose time has come and gone. In: IEEE. *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*. [S.l.], 2005. p. 2–11.

T-DRIVE. *T-Drive: Driving Directions based on Taxi Traces*. 2010. Acesso em: 20 jan. 2016. Disponível em: <http://research.microsoft.com/en-us/projects/tdrive/>.

TEE, J. *Lacking NoSQL standards more dangerous than proprietary vendor lock-in?* 2013. Acesso em: 26 jan. 2016. Disponível em: <http://www.theserverside.com/feature/Lacking-NoSQL-standards-more-dangerous-than-proprietary-vendor-lock-in>.

TIOBE. *TIOBE Index*. 2016. Acesso em: 31 jan. 2016. Disponível em: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>.

TUDORICA, B. G.; BUCUR, C. A comparison between several NoSQL databases with comments and notes. In: IEEE. *Roedunet International Conference (RoEduNet), 2011 10th*. [S.l.], 2011. p. 1–5.

VEEN, J. S. Van der; WAAIJ, B. Van der; MEIJER, R. J. Sensor data storage performance: SQL or NoSQL, physical or virtual. In: IEEE. *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. [S.l.], 2012. p. 431–438.

VIEIRA, M. R.; FIGUEIREDO, J. M. d.; LIBERATTI, G.; VIEBRANTZ, A. F. M. Bancos de dados NoSQL: conceitos,

ferramentas, linguagens e estudos de casos no contexto de big data. *Simpósio Brasileiro de Banco de Dados. São Paulo*, 2012.

W3C. *Web Services Glossary*. 2004. Acesso em: 06 jan. 2016. Disponível em: <http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211>.

WARD, J. S.; BARKER, A. Undefined by data: a survey of big data definitions. *arXiv preprint arXiv:1309.5821*, 2013.

WAZLAWICK, R. *Metodologia de pesquisa para ciência da computação*. 2. ed. [S.l.]: Elsevier Brasil, 2014. ISBN 9788535277821.

WEBBER, J.; PARASTATIDIS, S.; ROBINSON, I. *REST in practice: Hypermedia and systems architecture*. [S.l.]: O'Reilly Media, Inc., 2010. ISBN 9780596805821.

YUAN, J.; ZHENG, Y.; XIE, X.; SUN, G. Driving with knowledge from the physical world. In: ACM. *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.], 2011. p. 316–324.

YUAN, J.; ZHENG, Y.; ZHANG, C.; XIE, W.; XIE, X.; SUN, G.; HUANG, Y. T-drive: driving directions based on taxi trajectories. In: ACM. *Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems*. [S.l.], 2010. p. 99–108.

YUE, P.; JIANG, L. Biggis: how big data can shape next-generation gis. In: IEEE. *Agro-geoinformatics (Agro-geoinformatics 2014), Third International Conference on*. [S.l.], 2014. p. 1–6.

ZHENG, Y. *T-Drive trajectory data sample*. 2011. Acesso em: 06 jan. 2016. Disponível em: <http://research.microsoft.com/apps/pubs/default.aspx?id=152883>.

ZHENG, Y.; WANG, L.; ZHANG, R.; XIE, X.; MA, W.-Y. Geolife: Managing and understanding your past life over maps.

In: IEEE. *Mobile Data Management, 2008. MDM'08. 9th International Conference on*. [S.l.], 2008. p. 211–212.

ZHENG, Y.; XIE, X.; MA, W.-Y. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, Citeseer, v. 33, n. 2, p. 32–39, 2010.

ZHONG, Y.; HAN, J.; ZHANG, T.; FANG, J. A distributed geospatial data storage and processing framework for large-scale webgis. In: IEEE. *Geoinformatics, 2012 20th International Conference on*. [S.l.], 2012. p. 1–7.

ZIKOPOULOS, P.; EATON, C. et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. [S.l.]: McGraw-Hill Osborne Media, 2012. ISBN 9780071790536.

A

Esquema JSON do modelo de dados de trajetórias

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Trajetoria",
  "description": "dados de trajetorias",
  "definitions": {
    "objetoMovel": {
      "type": "object",
      "properties": {
        "id": {
          "description": "id do objeto",
          "type": "integer"
        },
        "descricao": {
          "description": "descricao do objeto",
          "type": "string"
        }
      },
      "additionalProperties": false
    },
    "dadosAdicionais": {
      "type": "array",
      "items": [
        {
          "type": "object",
```

```
25            "properties": {
26              "chave": {
27                "type": "string"
28              },
29              "valor": {
30                "type": "string"
31              }
32            },
33            "additionalProperties": false
34          }
35        ],
36        "additionalItems": false
37      }
38    },
39    "type": "object",
40    "properties": {
41      "id": {
42        "description": "identificador unico",
43        "type": "integer"
44      },
45      "descricao": {
46        "description": "descricao da trajetoria",
47        "type": "string"
48      },
49      "estados": {
50        "description": "vetor de estados",
51        "type": "array",
52        "items": [
53          {
54            "description": "dados do estado",
55            "type": "object",
56            "properties": {
57              "sequencia": {
58                "description": "sequencia do estado",
59                "type": "integer",
60                "minimum": 1
61              },
62              "objeto": {
63                "$ref": "#/definitions/objetoMovel"
64              },
65              "tipo": {
```

```
66                  "description": "tipo de dados da trajetoria",
67                  "type": "string"
68                },
69                "estadoAnterior": {
70                  "description": "numero do estado anterior",
71                  "type": "integer"
72                },
73                "ultimaModificacao": {
74                  "description":
75                      "data e hora da ultima modificacao",
76                  "type": "integer"
77                },
78                "segmentos": {
79                  "description": "vetor dos segmentos",
80                  "type": "array",
81                  "items": [
82                    {
83                      "description": "segmento",
84                      "type": "object",
85                      "properties": {
86                        "pontos": {
87                          "description":
88                              "vetor dos pontos do segmento",
89                          "type": "array",
90                          "items": [
91                            {
92                              "type": "object",
93                              "properties": {
94                                "latitude": {
95                                  "type": "number"
96                                },
97                                "longitude": {
98                                  "type": "number"
99                                },
100                               "altitude": {
101                                 "type": "number"
102                               },
103                               "timestamp": {
104                                 "type": "number"
105                               },
106                               "dadosAdicionais": {
```

```
                                "$ref":
                                 "#/definitions/dadosAdicionais"
                              }
                            },
                            "required": [
                              "latitude", "longitude"
                            ],
                            "additionalProperties": false
                          }
                        ],
                        "additionalItems": false
                      },
                      "tipoTransporte": {
                        "description":
                            "tipo de transporte utilizado",
                        "type": "string"
                      },
                      "dadosAdicionais": {
                        "$ref": "#/definitions/dadosAdicionais"
                      }
                    },
                    "additionalProperties": false
                  }
                ],
                "additionalItems": false
              },
              "dadosAdicionais": {
                "$ref": "#/definitions/dadosAdicionais"
              }
            },
            "additionalProperties": false
          }
        ],
        "additionalItems": false
      }
    },
    "required": [
      "id", "estados"
    ],
    "additionalProperties": false
}
```

B

JSON com dados de trajetória

No seguinte exemplo temos uma trajetória com um estado e um segmento com dois pontos:

```
{
  "id": 15,
  "descricao": "2237",
  "estados": [
    {
      "estado": 1,
      "objeto": {
        "id": 2237,
        "descricao": "glaucio"
      },
      "tipo": "RAW",
      "ultimaModificacao": 1449946946802,
      "segmentos": [
        {
          "pontos": [
            {
              "latitude": 39.97505,
              "longitude": 116.30368,
              "timestamp": 1201967542000
            },
            {
              "latitude": 39.98575,
              "longitude": 116.34702,
```

```
                "timestamp": 1201968742000
              }
            ],
            "tipoTransporte": "carro"
          }
        ]
      }
    ]
}
```

C

Especificação da interface REST de persistência

```
swagger: '2.0'
info:
  version: "1.0.0"
  title: Persistencia Chave-Valor de Trajetorias
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /trajectorykeyvalue:
    get:
      description: |
        Retorna todos os objetos de `Trajetoria`.
      responses:
        200:
          description: OK
          schema:
            title: ArrayOfTrajetorias
            type: array
            items:
              $ref: '#/definitions/Trajetoria'
    post:
```

```
25        description: |
26          Armazena ou modifica uma ‘Trajetoria‘.
27        parameters:
28          - name: trajetoria
29            in: body
30            required: true
31            schema:
32              $ref: '#/definitions/Trajetoria'
33        responses:
34          200:
35            description: OK
36            schema:
37              $ref: '#/definitions/Trajetoria'
38    /trajectorykeyvalue/{trajId}:
39      get:
40        description: |
41          Retorna apenas uma ‘Trajetoria‘.
42        parameters:
43          - name: trajId
44            in: path
45            required: true
46            type: integer
47            format: int64
48        responses:
49          404:
50            description: Trajetoria nao encontrada
51          200:
52            description: OK
53            schema:
54              $ref: '#/definitions/Trajetoria'
55      delete:
56        description: |
57          Exclui uma ‘Trajetoria‘.
58        parameters:
59          - name: trajId
60            in: path
61            required: true
62            type: integer
63            format: int64
64        responses:
65          404:
```

```
66            description: Trajetoria nao encontrada
67          200:
68            description: OK
69  definitions:
70    ObjetoMovel:
71      type: object
72      title: ObjetoMovel
73      properties:
74        id:
75          type: integer
76          format: int64
77        descricao:
78          type: string
79    DadosAdicionais:
80      type: array
81      title: ArrayOfDadoAdicional
82      items:
83        type: object
84        title: DadoAdicional
85        properties:
86          chave:
87            type: string
88          valor:
89            type: string
90    Trajetoria:
91      title: Trajetoria
92      type: object
93      required:
94        - id
95        - estados
96      properties:
97        id:
98          type: integer
99          format: int64
100       descricao:
101         type: string
102       estados:
103         type: array
104         title: ArrayOfEstado
105         items:
106           type: object
```

```
107             title: Estado
108             properties:
109               sequencia:
110                 type: integer
111               objeto:
112                 $ref: '#/definitions/ObjetoMovel'
113               tipo:
114                 type: string
115               estadoAnterior:
116                 type: integer
117               ultimaModificacao:
118                 type: integer
119               segmentos:
120                 type: array
121                 title: ArrayOfSegmento
122                 items:
123                   title: Segmento
124                   type: object
125                   properties:
126                     pontos:
127                       type: array
128                       title: ArrayOfPonto
129                       items:
130                         title: Ponto
131                         type: object
132                         required:
133                           - latitude
134                           - longitude
135                         properties:
136                           latitude:
137                             type: number
138                           longitude:
139                             type: number
140                           altitude:
141                             type: number
142                           timestamp:
143                             type: number
144                           dadosAdicionais:
145                             $ref:
146                               '#/definitions/DadosAdicionais'
147                     meioDeTransporte:
```

```
148                     type: string
149                   dadosAdicionais:
150                     $ref: '#/definitions/DadosAdicionais'
151               dadosAdicionais:
152                 $ref: '#/definitions/DadosAdicionais'
```

D

Código fonte do experimento de componentização

D.1 Fonte original de segmentação

O seguinte código fonte faz a divisão da trajetória em segmentos de n pontos:

```
1  package br.udesc.mca.segmenter;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import br.udesc.mca.trajectory.model.TrajectoryPoint;
6  import br.udesc.mca.trajectory.model.TrajectorySegment;
7
8  public class SegmenterByPoints {
9    public static List<TrajectorySegment> segmenter(
10       TrajectorySegment segment, int points) {
11     long timestamp = System.currentTimeMillis();
12     List<TrajectorySegment> ret = new ArrayList<>();
13     int aux = 0;
14     TrajectorySegment ts = new TrajectorySegment();
15     ts.setTransportationMode(segment
16         .getTransportationMode());
17     List<TrajectoryPoint> ltp = segment.getPoints();
18     for (int i = 0; i < ltp.size(); i++) {
19       TrajectoryPoint tp = ltp.get(i);
```

```
20          aux++;
21          TrajectoryPoint taux = new TrajectoryPoint();
22          taux.setH(tp.getH());
23          taux.setLat(tp.getLat());
24          taux.setLng(tp.getLng());
25          taux.setTimestamp(timestamp);
26          ts.addPoint(taux);
27          if (aux == points) {
28            ret.add(ts);
29            ts = new TrajectorySegment();
30            ts.setTransportationMode(segment
31                .getTransportationMode());
32            aux = 0;
33            i--;
34          }
35        }
36        return ret;
37      }
38    }
```

D.2 Fonte do serviço REST de segmentação

O seguinte código fonte, reaproveita o código da seção D.1 fazendo o encapsulamento em um microsserviço com interface REST:

```
1   package br.udesc.mca.segmenter.service;
2
3   import java.util.List;
4   import javax.ws.rs.Consumes;
5   import javax.ws.rs.POST;
6   import javax.ws.rs.Path;
7   import javax.ws.rs.PathParam;
8   import javax.ws.rs.Produces;
9   import javax.ws.rs.core.MediaType;
10  import br.udesc.mca.segmenter.SegmenterByPoints;
11  import br.udesc.mca.trajectory.model.Trajectory;
12  import br.udesc.mca.trajectory.model.TrajectorySegment;
13  import br.udesc.mca.trajectory.model.TrajectoryType;
14  import br.udesc.mca.trajectory.model.TrajectoryState;
```

```
15
16 @Path("/segmenter")
17 @Produces(MediaType.APPLICATION_JSON)
18 public class SegmenterResource {
19
20   @POST
21   @Consumes(MediaType.APPLICATION_JSON)
22   @Path("/segmentByPoints/{points}")
23   public Trajectory segmentByPoints(
24       @PathParam("points") int points,
25       Trajectory trajectory) {
26     List<TrajectoryState> ltv = trajectory.getStates();
27     TrajectoryState tv = ltv.get(ltv.size() - 1);
28     List<TrajectorySegment> lts = tv.getSegments();
29     TrajectorySegment ts = lts.get(lts.size() - 1);
30     lts = SegmenterByPoints.segmenter(ts, points);
31     TrajectoryState tv2 = new TrajectoryState();
32     tv2.setSequence(2);
33     tv2.setType(TrajectoryType.SEGMENTED);
34     for (TrajectorySegment s : lts) {
35       tv2.addSegment(s);
36     }
37     trajectory.addState(tv2);
38     return trajectory;
39   }
40 }
```

E

Bancos de dados utilizados nos experimentos

Para a validação do protótipo, os seguintes bancos de dados foram utilizados:

Cassandra
Cassandra é um servidor de banco de dados do modelo família de colunas que baseia seu modelo de distribuição no Amazon Dynamo.

<http://cassandra.apache.org>

MongoDB
O MongoDB (*hu**mongo**us*) é um servidor de banco de dados do modelo documento que utiliza o formato formato JSON.

<https://www.mongodb.org>

MySQL
O MySQL é um servidor de banco de dados relacional com suporte à SQL destinado tanto à sistemas de missão crítica como para uso em *software* embarcado.

<http://www.mysql.com>

Redis

Redis é um banco NoSQL do modelo chave-valor. Ele também é referido como um servidor de estrutura de dados por causa do seu suporte à diversos tipos de dados. Por padrão, o Redis salva todos os dados na memória mas também pode armazenar os dados em disco.

<http://redis.io>

F

Bibliotecas utilizadas nos experimentos

Afim de acelerar o desenvolvimento do protótipo, as seguintes bibliotecas *open-source* foram utilizadas:

Dropwizard

Dropwizard é um *framework* de desenvolvimento Java para serviços Web RESTful que para isso encapsula os componentes necessário para inicar um processo servidor. Inicialmente foi construído pelo Yammer[1] para ser usado como a base dos seus sistemas de *backend*.

<http://www.dropwizard.io>

Hibernate

Hibernate é um projeto que tem como objetivo fornecer uma solução completa para o problema de gerenciamento de dados persistentes em Java. Seu ORM consiste em um núcleo, um serviço de base para a persistência com bancos de dados SQL, e uma API proprietária nativa.

<http://hibernate.org>

Jackson JSON Processor

Jackson é um processador de JSON de alto desempenho

[1] Yammer é uma rede social empresarial.

para Java. Ele faz o mapeamento de uma estrutura de objetos em memória para o formado JSON e vice-versa.

<http://wiki.fasterxml.com/JacksonHome>

Jersey

Jersey é um *framework* para o desenvolvimento de serviços REST que implementa a especificação JAX-RS.

<https://jersey.java.net/>

Jetty

Jetty provê um servidor web com suporte à Java Servlets e *Websockets*. Tem como principal vantagem de ser ser fácilmente embarcado em aplicações que necessitem acesso HTTP.

<http://www.eclipse.org/jetty/>

Netflix Eureka

Eureka é um serviço REST com base que é usado principalmente na nuvem para a localização de serviços para fins de balanceamento de carga e *failover* de serviços da camada intermediária.

<https://github.com/Netflix/eureka/wiki>