Universidade de Aveiro
Ano 2017

Departamento de Eletrónica, Telecomunicações e Informática

Paulo Sérgio Oliveira Pintor

Processamento analítico de fluxos de dados de tráfego em tempo quase real

Universidade de Aveiro
Ano 2017

Departamento de Eletrónica, Telecomunicações e Informática

Paulo Sérgio Oliveira Pintor

Processamento analítico de fluxos de dados de tráfego em tempo quase real

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, realizada sob a orientação científica do Doutor José Manuel Matos Moreira, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Este trabalho é dedicado às duas Marias da minha vida. A minha mãe e a minha filha que me dão a força necessária para ultrapassar todos os obstáculos e desafios que encontro ao longo da minha vida.

o júri / the jury

presidente / president

Prof. Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Fernando Joaquim Lopes Moreira
Professor Associado do Departamento de Inovação, Ciência e Tecnologia da Universidade Portucalense
(Arguente Principal)

Prof. Doutor José Manuel Matos Moreira
Professor Auxiliar da Universidade de Aveiro
(Orientador)

agradecimentos / acknowledgements

Um agradecimento especial a todos aqueles que estiveram ao meu lado ao longo desta caminhada, amigos e família, que me apoiaram e ajudaram em todos os momentos pois sem eles não seria possível chegar aqui e em especial aqueles que mais diretamente estiveram próximos de mim e me ajudaram nestes últimos momentos. Queria também deixar dois agradecimentos muitos especiais, à minha mãe por tudo e porque sem ela era impossível culminar esta etapa importante da minha vida e ao Professor Doutor José Manuel Matos Moreira por toda a ajuda, por todo o tempo disponibilizado, por toda a paciência, em suma por tudo ao longo deste tempo todo tendo sido um gosto trabalhar ao seu lado.

palavras-chave

Processamento de fluxos de dados, análise de dados em tempo quase real, informação e previsão de tráfego, Apache Storm.

resumo

Nos dias de hoje, as tecnologias com as quais temos contacto geram dados sobre a sua utilização e sobre o utilizador, com uma velocidade e variedade sem precedentes. Cria-se assim a necessidade de gerir os fluxos de dados e de transformar estes dados em informação armazenada de forma estruturada, inferindo sobre a mesma e retirando conclusões. As áreas de aplicação são diversas e uma das vertentes que tem recebido maior atenção é o processamento de dados referentes ao tráfego automóvel obtidos usando dispositivos GPS, que se devidamente tratados permitem dar informação adicional aos utilizadores sobre o estado do trânsito, encontrar os caminhos mais rápidos ou até fazer previsões sobre o tráfego no futuro.

O objetivo desta dissertação consiste em implementar um protótipo que consiga fazer o processamento de um fluxo de dados obtidos em tempo real e estruturá-los de forma a dar respostas sobre o estado do tráfego no momento e no futuro próximo. Para conseguir dar estas respostas, serão considerados não só os dados recebidos em tempo real como também informação adquirida anteriormente, de forma a ser possível fazer comparações e tirar conclusões. O protótipo está dividido em três módulos principais: o pré-processamento e a análise de dados históricos; o processamento de dados de tráfego em tempo quase real; e a apresentação de resultados. O protótipo foi sujeito a testes e os seus resultados sujeitos a avaliação de forma a verificar a validade das respostas devolvidas ao utilizador.

keywords

Data streams processing, near real-time data analysis, information and traffic prediction, Apache Storm

abstract

Nowadays, the technologies we handle generate data about their usage and the user, with an unprecedented rate and variety. This raises the need to manage all the data streams and to transform these data in information. This information is stored in a structured way allowing to infer about it and withdraw conclusions. There is a wide range of application areas, with the car traffic data processing receiving the most attention. These data are obtained from GPS devices and if properly processed, allow the user to have additional information about the traffic status, the faster way to a destination and even predictions on the future traffic status.

This dissertation aims to implement a prototype able to process and structure the data streams in real-time, to ultimately present answers about the traffic status at the moment or even in a near future. These answers are obtained not only by the real-time information but also by previously acquired information. Having two sources of information allows to compare and withdraw statistical conclusions. The prototype is divided in three main modules: the pre-processing and analysis of historical data; the processing of traffic data in near real-time; and the results presentation. The prototype was subject to tests and their results subject to evaluation to verify the answers' assertiveness.

Tabela de conteúdos

Lista de figuras

Lista de tabelas

Glossário

Apache Kafka – Plataforma para transmissão e gestão de fluxo de dados.

C# – Linguagem de programação orientada a objetos.

Cache – Armazenamento temporário em disco de informação, quando relativo a *Web* armazena por exemplo páginas *Web*.

Clojure – Linguagem de programação funcional.

Data mining – Processo de triagem de grandes conjuntos de dados para identificação de padrões.

Data warehouse – Repositório desenhado para *reporting* e processamento analítico de dados.

geoJSON – Formato para codificar dados geográficos.

Java – Linguagem de programação orientada a objetos.

jQuery – Biblioteca de JavaScript para interagir com o HTML.

Junit – Ferramenta para criação de testes unitários automáticos.

Kestrel – Plataforma para transmissão e gestão de fluxo de dados.

MapBox – Fornecedor de mapas *online*.

Map-matching – Processo de alinhamento de uma sequência de posições observadas com uma rede de estradas num mapa.

Open source – Quando referente a código de *software* significa que o código é aberto à comunidade.

PostgreSQL – Sistema de gestão de base de dados relacionais.

Python – Linguagem de programação que suporta diferentes paradigmas de programação.

QGIS – Sistema de referenciação geográfica que permite ver, editar e analisar dados geoespaciais.

Segmento – Corresponde a um troço de estrada entre duas interseções.

Scala – Linguagem de programação funcional.

Single Point of Failure – Parte de um sistema cuja falha causa paragem de todo o sistema.

Tempo quase real – Atraso de tempo, introduzido por sistemas de processamento de dados, entre o instante temporal de um evento e o uso dos dados processados.

Web service – Solução com um conjunto de métodos que podem ser acedidos e invocados por ferramentas na *Web*.

Lista de Acrónimos

CSV – Comma-separated Values

HTML5 – Hypertext Markup Language 5

IDE – Integrated Development Enviroment

IP – Internet Protocol

J2EE – Java Platform, Enterprise Edition

JAR – Java Archive

JDK – Java Development Kit

JVM – Java Virtual Machine

REST – Representational State Transfer

SOAP – Simple Object Access Protocol

WSDL – Web Service Definition Language

XML – eXtensible Markup Language

1 Introdução

Nos dias de hoje, quando se fala em sistemas GPS, principalmente para utilizadores que conduzam veículos automóveis, a grande preocupação reside no fornecimento de informação precisa ao utilizador, assim como melhorar a sua experiência de condução, consoante aquilo que pretenda e os locais pelos quais gostaria de passar. No entanto, para utilizadores diários de aplicações GPS, a maior preocupação é a rapidez com que se consegue chegar ao destino pretendido, escolhendo o melhor trajeto. Mas, para que tal aconteça, é necessária a existência de informação em tempo real que ajude o utilizador a tomar as decisões mais acertadas, existindo muitos fatores que podem modificar o que pode ser considerada "a melhor decisão".

Existem vários fatores que condicionam o trânsito nas estradas, principalmente tratando-se de grandes cidades. Alguns destes fatores podem ser controlados ou previstos, outros são imprevisíveis, sendo o caso mais evidente a existência de outros condutores, pois acaba por ser difícil prever ou controlar os seus comportamentos. Por outro lado, existe informação que pode ser obtida em tempo real ou prevista com dados anteriormente recolhidos, nomeadamente:

- Estado do trânsito – num sistema perfeito, se fosse exequível recolher os dados completos sobre o movimento de todos os veículos, seria possível obter o número de veículos a circular num determinado local, numa data e hora específicas. No entanto, não estando todos os veículos equipados com este sistema, uma maneira de colmatar este fator é utilizar dados recolhidos no passado para conseguir fazer uma previsão do tráfego automóvel nos locais pretendidos. Existem diversas condicionantes às quais é necessário prestar atenção:

    - Horário – o trânsito nas grandes cidades difere de hora para hora, por exemplo, nas horas de ponta – quando os cidadãos se deslocam para o trabalho ou estão de volta a casa.
    - Data – o dia da semana é importante, pois o comportamento do trânsito ao fim-de-semana e feriados é tendencionalmente diferente de um dia normal de semana.
    - Estado do tempo – o estado meteorológico é também uma condicionante para quem está a conduzir. Os condutores têm comportamentos diferentes consoante esteja sol ou a chover e se a estrada está mais ou menos molhada.
    - Obras na via pública – as obras nas vias públicas podem também condicionar o trânsito. Existem intervenções previamente planeadas e outras que têm de ser efetuadas na hora para resolver problemas ocasionais.
    - Acidentes de trânsito – outra condicionante do estado do trânsito são os acidentes que, por norma, podem condicionar a circulação dos veículos devido à intervenção prestada.
- O modo de condução – cada utilizador tem a sua forma de conduzir:
    - Alguns condutores conduzem mais rápido que outros.
    - A sua condução é afetada pelo estado do tempo.
    - Podem ter receio e evitar conduzir em locais com certas características – rotundas, semáforos, passadeiras.

É impossível prever este comportamento, contudo é possível aprender os hábitos dos condutores e, consoante os mesmos, tomar decisões para melhorar o seu trajeto. Informação como o estado meteorológico, acidentes de trânsito, obras na estrada – aqui depende do local e da cidade, pelo que já existem algumas aplicações que fornecem informação sobre estas condicionantes – pode ser obtida em tempo real. A restante tem de ser estimada usando dados históricos, fazendo então uma análise estatística seguida de uma previsão do que poderá acontecer.

Estes dados agrupados e processados podem ser úteis para que seja fornecida ao utilizador informação correta das decisões que poderá tomar, para chegar mais rapidamente ao seu destino. Contudo, atualmente a grande dificuldade encontra-se na transmissão e processamento de todos estes dados em tempo quase real.

O objetivo desta dissertação é estudar soluções eficientes para processamento analítico de fluxos de dados e avaliar o potencial para integrá-los com as aplicações de GPS de forma a melhorar a experiência do utilizador.

Em suma, o propósito desta dissertação assenta na importância de um processamento eficiente para os dados recolhidos a partir de equipamentos GPS. Estes dados, desde que processados e estruturados corretamente, melhoram a informação dada ao utilizador assim como a exatidão da mesma. Pretende-se que o utilizador obtenha a informação sobre rotas possíveis entre dois pontos, mas principalmente informação melhorada não só quanto ao estado do trânsito na estrada em que se encontra mas também nas circundantes da mesma.

Tendo em vista o propósito estabelecido, foi inicialmente realizado um estudo sobre as aplicações já existentes no mercado, numa tentativa de identificar quais as suas diferenças e qual o ponto diferenciador de cada uma. Foi realizada também uma análise de métodos já publicados e relacionados com a problemática em questão. Estes métodos têm como área de atuação várias etapas do processo de análise de dados de tráfego, desde a receção e incorporação dos dados GPS, a estruturação da rede de estradas a nível computacional e ainda a obtenção de respostas sobre o caminho mais curto entre pontos. Por fim, foi necessário compreender o estado atual do conhecimento ao nível do *software* de processamento de fluxo de dados de forma a escolher a tecnologia a usar neste trabalho.

A tecnologia escolhida foi o Apache Storm e esta escolha é justificada pela sua escalabilidade, importante para lidar com o aumento do fluxo de dados permitindo que não haja perda de dados, e também pela fácil adaptação a vários cenários de utilização dependendo do que se pretende com os dados.

Os dados processados pelo Storm serão depois enviados para um *Web service*, responsável por colocá-los em estruturas de dados que irão facilitar o seu manuseamento e dar informação importante sobre o estado do trânsito, como por exemplo o número de veículos que no momento se encontram numa determinada estrada. Com o recurso a algoritmos foi também possível conjugar a informação produzida com os dados recolhidos pelo Storm com conhecimento adquirido anteriormente para ser possível dar por exemplo uma previsão do número de veículos numa determinada estrada num futuro

próximo, entre outras. O conhecimento adquirido anteriormente consiste em médias de veículos numa determinada estrada, o número de veículos que transitam de uma estrada para outra ou os segmentos e interseções que fazem parte da rede de estradas, sendo possível conjugar os dados de forma a obter vários tipos de informação. Para suporte a alguns dos algoritmos realizados foi criada uma estrutura que representará a rede de estradas em questão a partir do conhecimento adquirido.

Por fim, foi criada uma aplicação cliente através da qual se simulou a interação do utilizador com um mapa, mimetizando uma situação da vida real com a informação presente no *Web service*. Trata-se de uma aplicação de demonstração onde se pretende mostrar informação sobre o tráfego em tempo quase real, por exemplo, o número de veículos num local ou estrada e a sua variação ao longo do tempo; a comparação do volume de tráfego atual com o tráfego habitual nesses locais. Para além da informação em tempo quase real, é possível mostrar os caminhos entre estradas e mostrar a previsão do número de veículos numa determinada estrada.

Para todas as análises e processamentos foram utilizados dados provenientes de taxistas na cidade de Pequim na China de forma a ser possível simular situações do quotidiano.

O contributo deste trabalho tem duas vertentes. Numa primeira vertente, conseguiu-se implementar um sistema para o processamento de fluxos de dados em tempo quase real sem perda de dados. Na segunda vertente, foi possível implementar uma solução que possibilita a obtenção de respostas não só sobre o estado atual do tráfego, mas também se consegue inferir sobre o seu estado futuro.

Devido ao grande número de informação produzida nos dias de hoje, esta área poderá vir a solucionar problemas em vários contextos. Os contextos abrangidos englobam desde o tema introduzido nesta dissertação sobre o tráfego automóvel, ao processamento de informação gerada por redes sociais ou até mesmo em áreas cientificas em que seja necessário processar grandes quantidades de dados em tempo real como por exemplo no caso da meteorologia em que é necessário compreender padrões e tendências associados ao comportamento do clima, e ainda em áreas financeiras nas quais é também necessário estabelecer modelos e tendências.

Esta dissertação será organizada nos seguintes capítulos:

- Métodos preditivos e tecnologias de análise de tráfego – capítulo onde será apresentado o contexto atual das tecnologias e métodos existentes associados a problemática desta dissertação.
- Modelação e arquitetura do sistema – capítulo onde será apresentado o modelo e a estrutura do sistema.
- Implementação – capítulo onde será explicado de forma detalhada a concretização do modelo e estruturação do sistema.
- Resultados – capítulo onde vão ser explicados os testes efetuados ao protótipo e apresentados os seus resultados.
- Conclusão e trabalho futuro – capítulo onde serão discutidos os resultados e o potencial associado aos mesmos, finalizando com algumas sugestões para trabalho futuro.

2 Métodos preditivos e tecnologias de análise de tráfego

Neste capítulo será apresentada uma visão geral sobre o tema desta dissertação abordando tópicos como os sistemas de gestão de fluxo de dados e a sua importância nos dias de hoje devido a toda a informação gerada pelos dispositivos informáticos que nos rodeiam no dia-a-dia, a forma como estes dados são guardados e como é possível aproveitá-los para melhorar a experiência do utilizador em várias áreas nomeadamente na área de navegação e na informação sobre estado do trânsito que será a área de atuação nesta dissertação.

Segue-se o tópico sobre tecnologias que procuram dar uma resposta ao problema de dados de tráfego em tempo quase real, tentando assim fornecer ao utilizador dados úteis para chegar mais rápido ao seu destino, bem como a abordagem utilizada por estas tecnologias para solucionar o problema. Sendo esta uma questão abrangente e com muitas variáveis, existem várias maneiras de abordar o problema e de dar uma resposta ao utilizador. Será importante também perceber como estas tecnologias lidam com a informação e como levam a cabo o seu processamento.

No tópico seguinte serão abordadas estratégias para resolver vários problemas inerentes a esta dissertação, destacando-se exemplos de estruturas de dados (grafos, matrizes) para a representação do esquema das estradas possibilitando e auxiliando a aplicação de algoritmos sobre as mesmas, assim como a utilização da informação recebida em tempo real em conjunto com os dados guardados previamente para ser possível uma previsão futura do estado do trânsito. De salientar também as várias abordagens e algoritmos, que com a informação anterior conseguem determinar o caminho mais rápido entre dois locais.

Por fim, no último tópico será abordada a questão do processamento de dados, onde será apresentada a ferramenta que será utilizada nesta dissertação, assim como a comparação com algumas que já existem e o porquê da sua escolha.

2.1 Enquadramento

Para começar é necessário entender o termo *Big Data,* e tudo o que ele implica. O termo *Big Data* refere-se a um grande conjunto de dados que são gerados por todos os dispositivos com que interagimos em termos tecnológicos sendo essa interação feita de vários modos, desde o envio de e-mail, redes sociais, etiquetas RFID, etc. Mas, o grande poder de toda esta informação está na possibilidade de a conseguir estruturar, guardar e combinar com mais informação, conseguindo assim retirar conclusões [1].

Doug Laney definiu três características para explicar o termo *Big Data* (Figura 1): Volume, Velocidade e Variedade [2].

Acerca do volume, existe uma grande variedade de fontes de informação que geram dados, desde as redes sociais, às transações que fazemos, até mesmo à utilização do nosso telemóvel, todos geram um grande volume de dados, que necessita de ser guardado e processado. Em termos de velocidade, atualmente, existe um grande número de dispositivos com ligação à internet e a velocidade de produção dos dados é enorme. Além disso, muitas vezes é necessário lidar com esses dados em tempo quase real de forma a conseguir extrair informação relevante em tempo útil. Por fim, a variedade está relacionada com o tipo de formatos através dos quais a informação é produzida, desde dados estruturados, numéricos, documentos, e-mail, vídeo ou outros, existindo inúmeros formatos e formas de produzir informação.

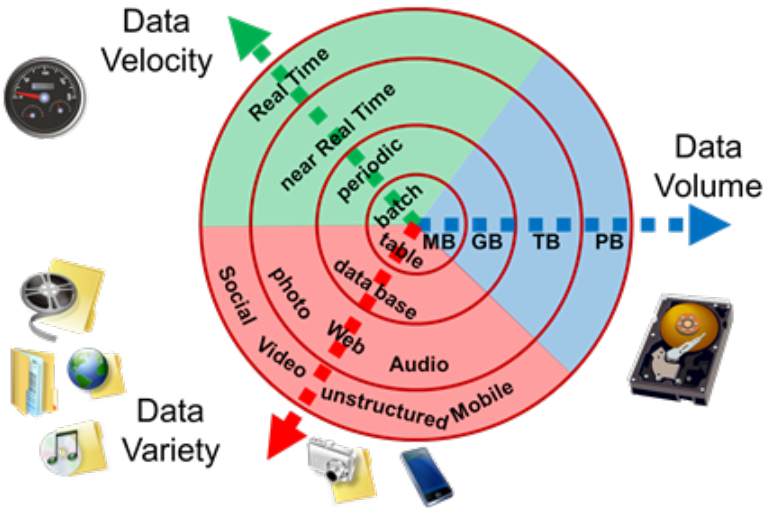


Figura 1 - Exemplo dos 3 V's de BigData (Fonte: [3]*)*

Um dos grandes problemas do *Big Data* é o armazenamento, uma vez que os dados não param de crescer e todos os dias existe mais informação para ser armazenada. Mas, de forma a tirar proveito de toda a informação não é suficiente apenas guardá-la, pois esta tem de ser estruturada para ser possível retirar conclusões.

O *software* e as tecnologias utilizadas têm de ser capazes de suportar uma grande quantidade de dados e serem escaláveis de forma a adaptarem-se à quantidade de dados que vão recebendo e garantir que não existem falhas ou quebras em todo o processo. Só assim é possível transformar toda a quantidade de dados em informação útil para mais tarde ser utilizada, ou até mesmo ser possível dar respostas em tempo quase real.

Para o estudo realizado é necessária a conjugação dos últimos dois pontos referidos, ou seja, conseguir processar a informação e guardá-la de forma a que mais tarde seja possível utilizá-la em termos estatísticos e comparativos, e conseguir ainda proceder a uma análise e dar respostas em tempo quase real, utilizando os dados que estão a ser processados no momento em conjunto com dados anteriormente guardados. Desta forma será possível oferecer uma variedade de respostas ao utilizador, dependendo sempre das suas necessidades.

Os dados para esta dissertação estão relacionados com o tráfego automóvel. Atualmente, estão disponíveis na internet diversos conjuntos de dados de tráfego que podem ser utilizados de forma livre. Estes conjuntos contêm percursos de táxis, onde existe uma grande variedade de rotas percorridas, dentro e fora das grandes cidades. Além disso, os taxistas são condutores com um grande conhecimento das estradas e que tendem a utilizar os caminhos

mais rápidos para chegar aos seus destinos. Existem ainda percursos de camiões, onde se encontra novamente uma grande variedade de rotas percorridas e sendo os seus condutores também experientes. É de notar que os itinerários efetuados por estes tendem a ter muitos registos em percursos considerados como rápidos, como são exemplos as autoestradas e vias-rápidas devido ao seu trabalho ser, normalmente, fazer entregas a longas distâncias e para zonas consideradas industriais evitando sempre que possível percorrer os centros das grandes cidades.

Para este estudo a escolha recaiu sobre o conjunto de dados dos táxis por apresentar uma grande variedade de registos em zonas metropolitanas das grandes cidades, zonas essas que os condutores ditos comuns também utilizarão mais no seu quotidiano. Estes dados vão estar compreendidos em períodos curtos, como por exemplos três meses, mas como são afetos a grandes cidades apresentam uma grande quantidade e variedade de informação.

Os utilizadores que usufruem de aplicações para obter indicações de como chegar ao destino que pretendem, têm de usar aplicações com GPS e por norma com uma ligação à internet, de forma a que seja possível carregar mapas e informações úteis sobre as estradas que vão percorrer. Estando os seus dispositivos a transmitir as coordenadas dos trajetos que estão a realizar, é necessário então processar essa informação e torná-la em informação útil para melhorar a sua experiência. Os dados transmitidos pelos estes utilizadores não são mais do que *Big Data*, principalmente se estivermos a falar de utilizadores em grandes cidades ou em grandes centros urbanos.

2.2 Tecnologias de informação de tráfego e navegação

2.2.1 Waze

A aplicação Waze foi criada pela Waze Ltd., uma empresa fundada em 2008, em Israel, mas foi mais tarde vendida à Google. Esta é uma aplicação de navegação GPS que difere de algumas soluções que já existiam e existem no mercado, pois tem por detrás uma comunidade de utilizadores que são os fornecedores de informação para o sistema.

Os utilizadores podem inserir uma variedade de dados na aplicação, desde a localização de radares ou operações realizadas pela polícia, congestionamentos de trânsito, acidentes, ou até o preço dos combustíveis. Com a utilização da aplicação podem ser recolhidos progressivamente dados de localização dos veículos que podem ser usados para obter

conhecimento sobre o estado do trânsito a partir do número de veículos que circulam em certa estrada e a velocidade média dos veículos.

Apesar de ser possível utilizar o Waze em todo o Mundo, não existem mapas completos de todos os países. Sendo uma aplicação em comunidade, a fiabilidade dos dados depende do número de utilizadores que operam a aplicação e que vão partilhando informação sobre as estradas que vão percorrendo, bem como da veracidade dessa mesma informação.

Por último, o Waze não tem a opção de mapas *offline*, ou seja, caso o utilizador perca o acesso à internet é utilizada alguma informação que eventualmente exista em *cache*, aguardando depois que seja possível voltar a estabelecer uma nova ligação [4–6].

2.2.2 Google Maps

A aplicação Google Maps é uma das aplicações mais conhecidas e uma das mais utilizadas em todo o Mundo. Disponível em formato Web ou aplicação móvel para *smartphones,* permite ao utilizador ter vários tipos de interação. Como o próprio nome da aplicação indica, esta aplicação foi desenvolvida pela Google.

O Google Maps permite obter o caminho entre dois pontos, apresentar pontos de interesse como restaurantes, bombas de gasolina, entre outras informações. Na versão Web permite também ver imagens reais de locais que sejam do interesse do utilizador usando a função de Google Street View.

Ao contrário do Waze, permite a navegação usando mapas *offline* e tem mapas que cobrem mais de 75% dos países de todo o Mundo [7], sendo possivelmente a aplicação que maior cobertura tem em termos de mapas ao nível de todo o globo [8].

Recentemente, foi adicionada a esta aplicação uma ferramenta com o nome de Google Traffic que permite, tanto na aplicação *Web* como na *Mobile*, ter acesso a informação sobre as condições de trânsito nas autoestradas ou nas estradas principais das grandes cidades. Com esta nova ferramenta, o Google Maps vai fornecendo informação ao utilizador de como se encontra o trânsito nos locais que este irá percorrer para chegar ao seu destino. Esta informação consiste, por exemplo, em avisar o utilizador que a rota que lhe foi dada está com alguns minutos de atraso. Para além desta informação o Google Maps também tem informação sobre bloqueios nas estradas, por exemplo, quando existem obras que impeçam

a circulação, sendo que esta advém de uma troca de informação que esta aplicação partilha com o Waze [9].

Assim sendo, para adquirir informação sobre as condições de trânsito, o Google Maps está dependente do número de utilizadores que manuseiam a aplicação. Um ponto a seu favor é o elevado número de utilizadores que operam a mesma, o que faz com que supere outras aplicações idênticas existentes no mercado.

2.2.3 INRIX Traffic

De todas as aplicações apresentadas é a mais recente e talvez a menos conhecida. Contudo, a empresa que está por detrás desta aplicação já se encontra no ramo da análise de dados de tráfego há bastante tempo, fornecendo até dados para sistemas de GPS integrados em veículos de marcas como a Lexus.

Esta aplicação é muito semelhante às anteriores. Tal como as outras fornece coordenadas ao utilizador que pretenda deslocar-se de um local até outro, possui informação sobre o trânsito nas estradas, assim como informação sobre alguns locais de interesse para o utilizador. A grande diferença é ter incorporado um sistema baseado numa plataforma na nuvem (*Cloud Platform*) chamado de *Autotelligent*, que aprende os hábitos de condução do utilizador. Desta forma, quando desenha o trajeto do utilizador, tem em atenção todas as estradas a que o utilizador dá preferência enquanto conduz para futuramente ter em consideração esta informação quando calcular as rotas para os destinos do utilizador. Para além disto, também vai adquirindo os hábitos de tolerância do utilizador ao tráfego para futuramente lhe dar rotas que se adequem ao seu perfil [10]. Esta aplicação também sincroniza os dados com o calendário de forma a dar informação ao utilizador sobre o seu itinerário. Como o Waze, também permite aos utilizadores dar informação sobre acidentes de viação, sobre radares e alguns perigos que possam existir na estrada [11].

2.2.4 Here WeGo

Esta aplicação teve origem na aplicação HERE maps, criada pela Nokia que funcionava sobre o sistema operativo da marca chamado Symbian .Para além das opções de navegação normalmente associadas a uma aplicação deste género, esta aplicação tinha uma opção chamada City Lens que era uma solução de realidade aumentada permitindo ao utilizador direcionar a câmara do seu dispositivo para um local (prédio, rua, ou outro) e a aplicação

dava informação sobre pontos de interesse, localizando-os no ecrã e apresentando alguns dados sobre os mesmos. Apesar de a Nokia ter sido comprada pela Microsoft, a aplicação de navegação foi comprada por vários grupos de marcas automóveis alemãs, que reformularam a aplicação e lhe deram o nome por que hoje é conhecida – Here WeGo [12].

Atualmente, esta aplicação está disponível para Android e iOS, tendo perdido a função City Lens, mas ganhando contudo outras funções também elas interessantes. Para além de uma interface simples, a aplicação ajuda o utilizador a encontrar estacionamento e, caso o destino seja um local com pouco estacionamento, a aplicação irá indicar locais à volta deste com parques de estacionamento. Tem também uma função chamada Go Home, que permite ao utilizador saber toda a informação de como chegar a casa por transportes públicos.

A aplicação cobre 130 países diferentes tendo a opção de ver o estado do tráfego em 55 deles. O grande problema desta aplicação será o seu tamanho, aproximadamente 10GBs caso seja descarregada completamente com todos os mapas e informação, no entanto é possível utilizar a aplicação em modo *offline* e ainda é possível filtrar os mapas por continentes e países [13].

De facto, existem inúmeras aplicações de tráfego e navegação, para os mais variados gostos. Aqui foram apresentadas quatro que se enquadram com o tema desta dissertação e que não apresentam qualquer custo para o utilizador, apesar de, por exemplo, a aplicação apresentada em 2.2.3 ter uma versão paga que pode proporcionar mais ferramentas para o utilizador usufruir. Para além destas aplicações, também existem aplicações centradas noutros tipos de transportes, como bicicletas ou para utilizadores que gostem de fazer caminhadas.

Apesar do grande número de aplicações que existem, por norma estas focam-se em determinar o caminho mais curto sem base no estado atual do tráfego, considerando apenas as distâncias entre locais de origem e destino. Também tentam dar informações aos utilizadores de pontos de interesse, desde a restauração, alojamento ou até mesmo locais para visitar.

2.3 Métodos de análise e previsão de tráfego

Como base para esta dissertação, foi feita uma pesquisa e leitura de artigos que se focam em oferecer uma solução para problemas de tráfego automóvel, desde a previsão de tráfego, à

análise de dados, entre outras variáveis que compõe este problema. De seguida serão apresentadas várias soluções e métodos que após esta pesquisa se considerou que deviam ser tomados em consideração para o trabalho que se pretende desenvolver.

2.3.1 Método de previsão de tráfego baseado em computação na nuvem

Em [14] é apresentado um sistema baseado em computação na nuvem (*Cloud Computing*) com o objetivo de fornecer ao utilizador o caminho mais rápido tendo em conta um ponto de partida e um de chegada, conjugando o estado do trânsito atual, que é obtido com a informação enviada por todos os veículos, com a informação já previamente guardada do histórico dos utilizadores, tendo também atenção ao comportamento do utilizador recorrendo a registos sobre a sua condução. Para além destas variáveis, é tido em conta também o estado do tempo que pode influenciar a condução, e o dia da semana e a hora, pois o tráfego automóvel pode sofrer alterações principalmente em horas de ponta, ou dias de feriado perto de zonas históricas.

Para ser possível ter um conjunto de dados de treino que permita fazer inferências com os dados que serão recolhidos em tempo real, é necessário utilizar uma amostra de dados previamente recolhidos. Neste artigo, esse conjunto de dados são de taxistas na China, mais propriamente da cidade de Pequim. Estes dados foram recolhidos durante três meses tendo dado origem a uma quantidade acima de 33 mil registos.

Este trabalho introduz o conceito de grafo baseado em segmentos de referência (*Landmark Graph*). Este é um grafo direcionado que contém um conjunto de referências (*landmarks*), em que estas referências são construídas a partir da análise de dados dos segmentos, onde se encontram padrões de trajetórias distintos consoante o estado do tempo ou o dia da semana e a hora, podendo haver para a mesma zona vários grafos.

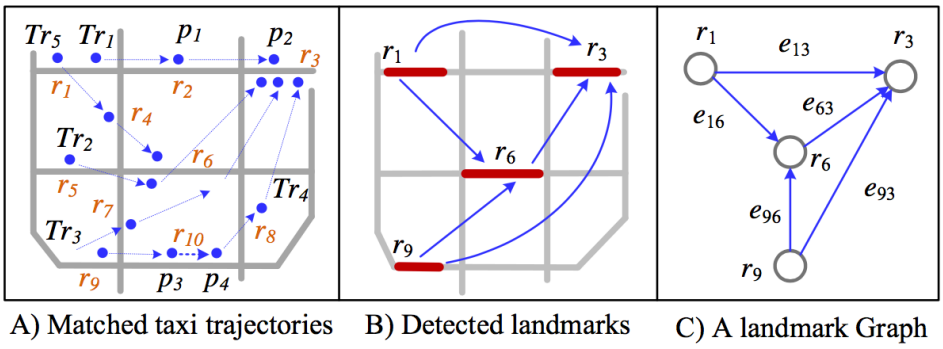


Figura 2 - Exemplo de um grafo baseado em segmentos de referência (Fonte: [14]*)*

A imagem da esquerda na Figura 2 mostra as trajetórias dos táxis sobre as estradas e onde é possível ver alguns padrões nas trajetórias.

A imagem ao centro demonstra os padrões encontrados, que serão os segmentos mais visitados sendo então detetados os pontos de referências dando depois origem a um grafo direcionado que é demonstrado na imagem da direita.

Estes grafos servem para modelar não só a experiência que se vai obtendo com os dados adquiridos pela utilização da aplicação, assim como os padrões das estradas em diferentes momentos temporais tendo sempre em conta o estado meteorológico, ajudando por fim nas inferências efetuadas para dar resposta ao utilizador.

Este método propõe o modo de funcionamento apresentado na Figura 3 que se explica pormenorizadamente de seguida.

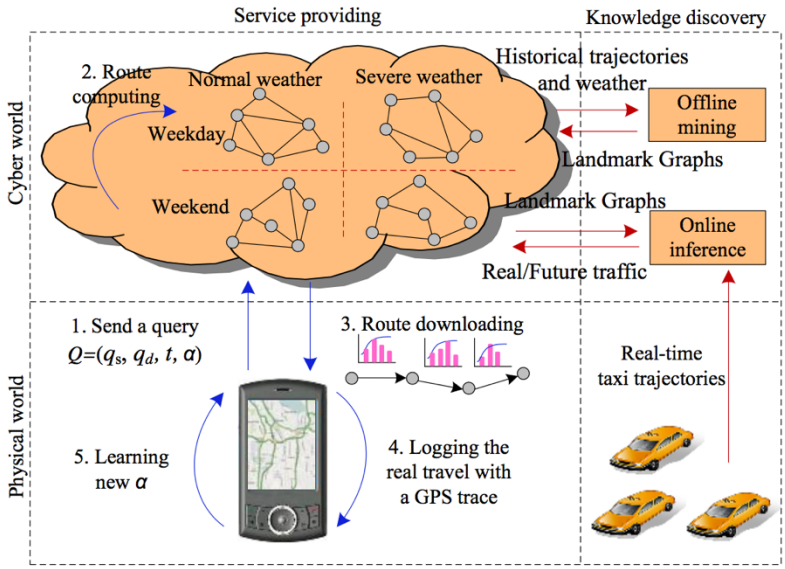


Figura 3 - Arquitetura proposta pelo método (Fonte: [14]

O esquema divide-se em Prestação de serviços (*Service providing*) e Descoberta de conhecimento (*Knowledge Discovery*), ambos baseados em informação física e informação computacional. Na Prestação de serviços, a informação física será fornecida pelo *smartphone* do utilizador que irá enviar dados com o local de partida, o local de chegada, hora de partida e um fator de aprendizagem, este último contendo a informação sobre a condução do utilizador e que é atualizado a cada viagem. De salientar que os resultados da aplicação melhoram ao longo do tempo com a utilização e com o acumular das distâncias percorridas pelo utilizador.

De seguida é feito o cálculo da melhor rota. Após ser encontrado o caminho mais rápido, é enviada a rota correspondente para o *smartphone*, e este deverá ainda guardar a informação do caminho que o utilizador fez (pois este pode não seguir o que lhe foi indicado).

A descoberta de conhecimento divide-se em duas partes: *offline mining* e *online inference*. Na primeira são construídos quatro grafos baseados em segmentos (troços) de estrada de referência que se dividem no estado meteorológico (normal e grave) e no tipo de dias (dias da semana e fim-de-semana). Para ser possível esta construção é necessária informação meteorológica e informação histórica. Esta última, consiste nos dados das trajetórias dos taxistas que foram utilizados como conjunto de treino. O *offline mining* deve ser realizado periodicamente, por exemplo mensalmente, para atualizar os dados que servem de base à fase seguinte, mas devem estar disponíveis sempre que o utilizador necessitar.

Na fase de *online inference* é calculado o tráfego em cada aresta do grafo com a informação que está a ser recebida dos táxis em tempo real. Também é possível estimar o tráfego futuro tendo em conta os dados que estão a ser recebidos em tempo real e a informação recolhida no *offline mining* e nos grafos que aí foram criados. Esta tarefa deve ser realizada periodicamente, por exemplo de dez em dez ou de quinze em quinze minutos. A parte física da descoberta de conhecimento são os veículos, que estão a enviar informação em tempo real.

As inferências sobre o tráfego no futuro são feitas recorrendo a uma cadeia de Markov que utiliza matrizes de transição com informação estatística sobre a transição entre arestas que convergem para um mesmo nó no grafo, ou seja, a probabilidade de um veículo ir da aresta A para a B.

Os resultados obtidos são comparados com [15] onde se conclui que o método apresentado consegue estimar com mais precisão o tempo de viagem de cada rota possível entre dois pontos e consequentemente irá fornecer as melhores rotas para cada utilizador, principalmente porque consegue aprender com o histórico do utilizador, tem em consideração variáveis como o estado do tempo e por conseguir inferir sobre o tráfego no futuro, algo não considerado em [15].

Ligado a este método existem outros que também estudam e referenciam este estudo sobre assuntos importantes que é possível retirar com os dados aqui utilizados.

Em [16] são utilizados os dados dos táxis em conjunto com técnicas de *Data Mining* para detetar padrões no trânsito e detetar anomalias de comportamento. É referido em [16] a importância da análise computacional de dados de tráfego para obter os padrões de tráfego como é feito em [14] que leva à criação dos grafos baseados em segmentos de referência, contudo aqui é realçada a importância de encontrar e entender também as anomalias no fluxo de tráfego e o seu significado. Estas anomalias levam a alteração do fluxo normal do tráfego e podem estar ligadas a acidentes no trânsito ou a outros incidentes que sendo possíveis detetar podem ajudar ao planeamento do impacto dos mesmos de forma a garantir que o fluxo do tráfego sofra um impacto mais suave. Com esta análise é possível concluir que existem anomalias persistentes, que apesar de serem anómalas ao tráfego em geral acontecem muitas vezes e outras que surgem esporadicamente.

O estudo [17] apresenta uma visão global sobre estruturas de dados para representação e análise de dados de tráfego de forma eficiente em termos computacionais e conclui que a escolha depende sempre do fim que se pretende dar aos dados utilizados.

2.3.2 Método para achar o caminho mais curto baseado em tempos e pesos de cada estrada

Em [18], também é abordado o tema do cálculo do caminho mais curto entre dois locais realçando que as aplicações que existem neste momento usam dados estáticos e se baseiam principalmente na distância entre os locais e os limites de velocidade, sendo necessário mais informação para que o caminho apresentado aos utilizadores seja efetivamente o mais rápido.

Os dados utilizados são provenientes da Grécia, mais propriamente da cidade Atenas, e foram recolhidos entre 2000 e 2003.

A proposta tem duas componentes importantes, o *Dynamic Travel Time Map (DTTM)* que é uma base de dados espácio-temporal para gerir os tempos de viagem criando pesos dinâmicos que mais tarde serão utilizados para calcular o caminho mais rápido. Os dados utilizados consistem no histórico das viagens recolhido usando dispositivos GPS.

O *DTTM* apresentado é realizado por meio de uma *Data Warehouse* espácio-temporal, onde a tabela de factos contém o tempo das viagens. Agregando os tempos que são recolhidos é possível determinar pesos para cada estrada. É referido também que a redundância é um

elemento chave, pois quanto mais informação for recolhida melhor e mais rigorosa será a informação que será possível retirar da *Data Warehouse.*

A segunda componente é *Floating Car Data (FCD)*. Este é um subproduto em aplicações de gestão de frotas em que dado um número mínimo de veículos e uma distribuição uniforme dos mesmos é possível fazer uma avaliação do tráfego e algumas previsões. Neste método, recorre-se a um componente de monitorização de dados dos veículos de forma a ser possível retirar os tempos de viagem numa rede de estradas.

De forma a compensar alguma falta de dados presente na *Data Warehouse* é proposto também o uso de dois métodos representados na Figura 4: *Neighborhood Method* e *Tiling Method*.

O primeiro método considera que se existe informação sobre uma estrada específica e essa estrada tem outras que são consideradas suas vizinhas, então todas elas se vão comportar da mesma maneira e estão todas na mesma categoria de estradas. A vizinhança é definida com um retângulo cujos limites são determinados usando o tempo de viagem. Este tempo de viagem é retirado da estrada considerada como principal, ou seja, a estrada para a qual existem mais dados disponíveis. Desta forma, os tamanhos dos retângulos podem variar conforme a estrada principal e os seus dados.

O segundo método, que é uma simplificação do anterior, também considera que uma estrada terá vizinhos e que todos se comportam da mesma maneira, contudo aqui a rede de estradas é subdividida por quadrados com o mesmo tamanho, sendo computacionalmente menos dispendioso que o anterior.

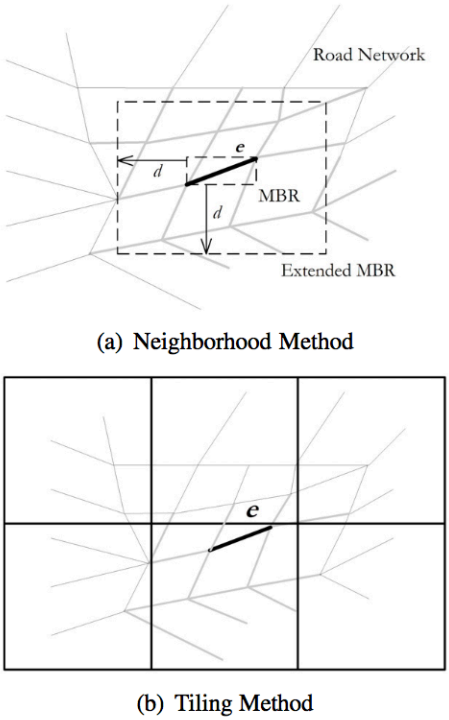


Figura 4 - Neighborhood Method e Tiling Method (Fonte: [18]*)*

Foram também testados alguns algoritmos para encontrar o caminho mais curto, nomeadamente, os algoritmos A*, Dijkstra's e D*. Não foi encontrado o melhor algoritmo para ser utilizado com os dados recolhidos, sendo que na conclusão deste estudo é proposto o uso de algumas variações destes algoritmos.

O estudo apresentando em [19] apresenta um contexto semelhante ao método supracitado. Também aqui são utilizados os dados do *FCD* e também é criada uma *Data Warehouse* para gerir os dados recolhidos. Contudo, neste caso a maior preocupação focou-se na forma de recolha da informação propondo um sistema baseado em *Web services* e em algoritmos de *map-matching*, produzindo como produto final uma plataforma de visualização *online* para apresentar informação sobre as estradas recorrendo a um sistema de cores (vermelho – amarelo – verde). As estradas a vermelho seriam as congestionadas e as verdes estradas onde se circula normalmente, existindo ainda a cor amarela que será o meio termo entre as duas anteriores.

Os dados utilizados neste estudo, também têm em atenção as horas dos dias em que a informação é recolhida assim como o dia da semana em que se encontra. Os algoritmos de *map-matching* são utilizados para colmatar alguma falha nos dados recolhidos principalmente na sua sobreposição nos mapas.

2.3.3 Método para achar o caminho mais curto baseado em padrões de velocidade

Ao contrário dos outros métodos até agora apresentados o foco em [20] é o estudo dos algoritmos de caminho mais curto tendo em conta os padrões de velocidade dos veículos nas estradas. A proposta procura resolver o *Time-Interval All Fastest Path (allFP) query*, ou seja, encontrar todos os caminhos mais rápidos tendo em conta a hora a que se pretende percorrer o trajeto.

Para capturar as alterações de velocidade é utlizado o conceito *CapeCod – Categorized PiecewisE COnstant speed* em que os troços da estrada são categorizados tendo em conta a velocidade e consoante a hora e o dia da semana em que a informação é recolhida. Este padrão é uma extensão do padrão *Flow Speed Model* que foi abordado no artigo [21].

Um exemplo de uma consulta feita para saber os caminhos mais rápidos será – "Vou sair para o trabalho a qualquer momento entre as sete e as nove da manhã, e quero saber quais os caminhos mais rápidos entre o local A e o local B" e a resposta dada será do género "Siga pela rota A se for entre as sete e as sete e quarenta e cinco, caso contrário siga pela rota B".

Os caminhos mais rápidos são encontrados e depois categorizados consoante a hora que for indicada na consulta. Para chegar a esses caminhos, o método irá usar o intervalo de tempo e subdividir esse intervalo em pequenos intervalos de forma a perceber quais são as melhores estradas em cada intervalo de tempo mais pequeno. Para saber quais são os caminhos possíveis entre dois pontos, o método propõe o uso de algumas extensões do algoritmo A*.

O estudo conclui que é possível chegar a resultados precisos e eficientes que podem ser utilizados em aplicações já existentes de forma a melhorar o seu funcionamento. Para trabalho futuro, realça-se que usando trabalho já feito em termos de *spatial queries* como *closest pairs* ou *clustering,* pode ser alcançado um impacto interessante nos resultados retirados.

2.4 Apache Storm – Tecnologia para processamento do fluxo de dados

Para além dos problemas algorítmicos, existe também o problema do processamento de fluxos de dados. Existem já alguns exemplos de aplicações que ajudam neste processo, sendo que para esta dissertação foi escolhido o Apache Storm [22], um dos mais recentes no mercado, mas que ainda assim alcançou uma boa recetividade devido à sua eficácia e

evolução.

O Apache Storm é uma plataforma desenvolvida para processar grandes quantidades de informação em tempo real, de maneira distribuída e escalável de forma horizontal. Permite aos seus utilizadores uma fácil manipulação de dados, sendo simples e fácil de adequar aos problemas específicos de cada um. A distribuição do Storm, é feita num ambiente de grupo (*Cluster*) existindo vários processos a desempenhar a mesma tarefa de uma forma paralela. Neste caso, o ambiente de grupo é controlado por outro serviço, também ele da Apache, chamado *ZooKeeper*, este terá uma interação com atores que existem na estrutura de grupo do Storm designadamente Nimbus e Supervisor.

O Storm é considerado sem estado (*Stateless*) – não guarda estados, isto é, não existe informação sobre as interações anteriores e as tarefas são processadas inteiramente com a informação que contém no momento – contudo, é na interação com o ZooKeeper que o Storm acaba por não ser inteiramente sem estado porque este vai guardando alguns estados, sendo mais à frente especificado o porquê desta pequena interação.

O Storm está escrito em Java e Clojure e foi inicialmente criado por Nathan Marz e a equipa BackType (Empresa de Análise Social - *Social Analytics*), tendo sido mais tarde adquirido pelo Twitter e transformado em *Open-source*.

Apesar de ser uma tecnologia recente e ainda em constante crescimento, o Storm constitui uma excelente opção de mercado em especial quando o processamento é feito para analisar dados em tempo real, tendo já um conjunto de empresas como Twitter, The Weather Channel, Yahoo, Spotify, entre outros, a usar esta tecnologia no seu modelo de negócio.

Estrutura de elementos do Storm

- Tuplo (*Tuple*) – é a estrutura de dados do Storm, sendo uma lista de elementos ordenados com a qual é possível passar qualquer tipo de dados e informação entre as estruturas do Storm. Internamente, os elementos desta lista encontram-se separados por vírgulas.
- Stream – é uma sequência não ordenada de tuplos.
- Spouts – é a base do Stream. Existem várias formas de transmitir informação aos Spouts, desde criar um Spout para ler a partir de uma fonte de dados, ou usar plataformas como Apache Kafka, Kestrel, etc. Para a implementação de Spouts

existem algumas interfaces, sendo que a principal é a *ISpout*, existindo também a *IRichSpout*, *KafkaSpout*, entre outras. Estas interfaces fornecem as classes e a estrutura base para o funcionamento deste componente.

- Bolts – são as unidades de processamento lógico. Recebem informação dos Spouts fazendo o processamento da mesma, podendo filtrá-la, agregá-la, entre outras operações. A informação tratada pode depois ser passada para outro Bolt ou ser enviada para outra plataforma, como por exemplo para uma base de dados, dependendo sempre da lógica que está a ser aplicada. Assim como os Spouts, os Bolts têm interfaces para fornecer as classes e as estruturas, existindo a principal *IBolt*, podendo, no entanto, se utilizadas outras como *IRichBolt* ou *IBasicBolt*, por exemplo.

Para que todos os componentes funcionem e comuniquem entre si, é necessária uma estrutura que indique a sua lógica e a sua ordem. No caso do Storm, essa estrutura é chamada de topologia (*Topology*) e não é mais do que um grafo orientado, onde os vértices são os processos e as arestas a informação que é passada. É nesta fase que se descreve a forma como os Bolts transferem informação entre si, definindo-se se a informação vai novamente para outro Bolt ou se tem o seu término ali, sendo normativo que toda a topologia começa com um ou vários Spouts.

O Storm pode correr várias topologias em simultâneo e estas só terminaram quando o utilizador der indicações para o seu término. Depois de definida a topologia, a lógica e a ordem dos componentes Spouts e Bolts, surge a fase de execução que tem o nome de tarefa (*Task*). Os componentes (Spouts e Bolts) podem ter múltiplas instâncias a correr em simultâneo em múltiplas *threads,* dependendo da definição da topologia. Sendo este um ambiente distribuído, as tarefas são distribuídas por múltiplos trabalhadores (*Workers*), sendo que este último componente possui vários nós com a função de executar a(s) tarefa(s).

Em todo este processo, existe um fluxo de informação que vai passando na estrutura de tuplos entre os componentes, sendo necessário entender como se comporta o fluxo desta informação ao longo de todo o processo. Para isto, existe um parâmetro que no Storm é chamado de agrupamento de fluxo (*Stream Grouping*).

O fluxo de tuplos possui várias formas de comportamento:

- Agrupamento aleatório (*Shuffle Grouping*) – um número igual de tuplos são

distribuídos de forma aleatória por todos os trabalhadores que estão a ser executados nos Bolts.

- Agrupamento por campo (*Field Grouping*) – o tuplo contém o nome das variáveis que o compõem e posteriormente será atribuído um valor a estas variáveis. Nesta situação, o fluxo dos tuplos é organizado consoante o nome da variável e o seu valor, por exemplo, os tuplos com o campo "curso" e com o valor "msi", serão todos direcionados para o mesmo trabalhador.
- Agrupamento de chaves parciais (*Partial Key grouping*) – em termos de distribuição, é similar ao anterior, agrupando pelas variáveis, no entanto tenta encontrar um balanceamento entre os Bolts, distribuindo a informação consoante o Bolt que estiver mais disponível. Deve ser utilizado quando a informação não é recebida de uma forma equilibrada, havendo por isso vários momentos em que existem Bolts parados ou em fim de processamento.
- Agrupamento global (*Global Grouping*) – todos os tuplos gerados, independentemente do seu valor, são encaminhados para uma instância.
- Todos os agrupamentos (*All Grouping*) – neste caso são enviadas cópias de cada tuplo para todas as instâncias de Bolt. É utilizada, preferencialmente, para enviar sinais entre os Bolts.
- Nenhum agrupamento (*None Grouping*) – em termos práticos é igual ao agrupamento aleatório, a única diferença é que quando executados com outros agrupamentos são sempre os últimos a serem enviados para processamento.
- Agrupamento direto (*Direct grouping*) – é um agrupamento especial em que o emissor do tuplo decide para que tarefa este será encaminhado e só funcionará se o fluxo tiver sido definido como direto.

Como referido inicialmente nesta explicação, a distribuição do Storm é feita num ambiente de grupo, e é este ambiente que permite que o mesmo seja tolerante a falhas, rápido e sem *Single Point of Failure*.

De seguida, será apresentada toda a estrutura de grupo, assim como os seus componentes, e uma breve explicação do seu funcionamento.

O Apache Storm tem dois tipos de nós, Nimbus e Supervisor, que comunicam entre si utilizando um sistema interno e distribuído de mensagens. O primeiro nó constitui o

componente principal do Storm, que analisa a topologia e recolhe as tarefas que têm de ser executadas, distribuindo estas últimas pelos Supervisors. O Supervisor, por seu turno, é um nó de trabalho (*Worker Node*), que irá enviar as tarefas para os processos dos trabalhadores (*Workers Process*).

O Supervisor não executa as tarefas, mas criará *threads* para efetuar a execução, podendo haver múltiplas *threads* a ser processadas. Estas *threads* têm o nome de executores (*Executors*), sendo que estes podem executar várias tarefas, mas todas elas têm a obrigatoriedade de pertencer ao Spout ou ao Bolt para o qual os executores foram instanciados.

Por fim, existe o Apache Zookeeper, que coordena todas operações nos nós do grupo, e para além de coordenar, é também o responsável pela existência de informação a ser partilhada bem como pela inexistência de perda de informação em todo o processo do fluxo de dados. Para que no caso de falha do Nimbus a informação não seja perdida e o processo seja recomeçado onde parou, este nó é monitorizado e é guardada informação sobre o seu estado, acontecendo o mesmo com os Supervisors. Esta informação para além de permitir controlar as falhas ajuda também à comunicação entre o Nimbus e o(s) Supervisor(s), sendo isto tudo responsabilidade do Apache Zookeepper, e também a razão pela qual o Storm não é completamente sem estado, como foi referido inicialmente.

Para finalizar, será feita de seguida uma comparação do Storm com outras ferramentas semelhantes, e que tentam também eles dar uma resposta ao problema do processamento de grandes volumes de dados.

Apache Storm VS Hadoop [23]

- O Apache Storm faz processamento de fluxos (*Streaming processing*) em tempo real, enquanto o Hadoop realiza processamento por lote (*Batch processing*).
- O Storm não tem estados, enquanto o Hadoop tem estados (*Stateful*).
- Os dois usam uma arquitetura Mestre-Servo (*Master-Slave*), mas enquanto o Storm utiliza sempre o Zookeeper, o Hadoop pode utilizar ou não. No caso do Storm, o mestre é o Nimbus e os servos os Supervisors, no Hadoop o primeiro é o Job Tracker, e o segundo é o Task Tracker.
- O Storm pode processar uma grande quantidade de informação por segundo no

grupo. O Hadoop usa o *Hadoop Distributed File System* (HDFS) com a estrutura MapReduce para processar grande quantidade de informação que pode demorar minutos ou horas.

- No Hadoop as tarefas de MapReduce são processadas de forma sequencial até acabarem. No entanto, no Storm as topologias só acabam por um erro que não tenha recuperação, ou por ordem do utilizador.
- No Storm se um Nimbus ou um Supervisor tiverem algum problema, eles podem recomeçar do ponto em que se encontravam, contrariamente ao que acontece no Hadoop, pois se houver uma falha no JobTracker todo o trabalho é perdido.

Apache Storm VS Apache Samza [24]

- São muito parecidos e os dois são sistemas de processamento de fluxos de dados;
- O Storm não tem estado, mas o Samza tem estados.
- Ambos funcionam com uma arquitetura Mestre-Servo, mas no Samza é o Apache Hadoop YARN o responsável de gestão.
- No Storm existem os tuplos para enviar informação, no Apache Samza a informação é enviada por mensagens.
- Ao contrário do Storm, onde a informação pode ser extraída de vários tipos de fonte, o Apache Samza utiliza apenas o Apache Kafka.
- Em termos de entrega de informação, também aqui o Storm e o Samza partilham a mesma categoria – *At-least-once* – ou seja, garantem que a mensagem é sempre entregue, podendo ser entregue duas vezes, garantindo a ausência de perdas, mas podendo contudo haver duplicação.
- Em termos de suporte de linguagens de programação, o Samza só suporta Scala e Java, enquanto o Storm suporta várias linguagens desde Java, Python, entre outras.

Apache Storm VS Apache Spark [25]

- Ao contrário do Storm, o Spark realiza processamento por lote mas também é capaz de fazer processamento em micro-lotes (*Micro-batching Streaming)*.
- O Spark, ao contrário do Storm, tem estados.
- O Spark, em termos de processamento de grandes quantidades de informação, é uma extensão do modelo MapReduce e utiliza o *Resilient Distributed Datasets* (RDD), que é a peça fundamental para a estrutura do Spark.

- Em termos de entrega de informação o Storm entra na categoria de *At-least-once*, enquanto o Spark está na categoria de *Exactly-once*, ou seja, a informação é entregue uma vez e apenas uma vez, sem perda de informação e sem duplicação de informação.

Esta comparação foi efetuada com o propósito de demonstrar que existem outras ferramentas que abordam as questões do processamento de dados de maneiras diferentes, relembrando que a escolha sobre qual utilizar depende também do que o utilizador pretende fazer e obter. É possível concluir então que existem três categorias: o Hadoop que faz um processamento por lotes; o Samza e o Storm que fazem processamento de fluxos de dados, e por fim o Spark, que é um sistema hibrido onde é possível fazer processamento por lotes ou processamento por micro-lotes. Contudo, existe também a possibilidade de o Storm fazer o processamento por micro-lotes, usando para isso uma extensão chamada Trident, sendo que através desta extensão se torna mais comparável ao Spark até mesmo na garantia de entregas de informação [26].

3 Modelação e arquitetura do sistema

Neste capítulo será feita uma explicação sobre as necessidades e os requisitos deste sistema, assim como uma abordagem prática dos dados utilizados e quais os resultados que com estes podemos obter. O protótipo e a sua idealização serão também descritos com detalhe, sendo explicado numa primeira instância o sistema e posteriormente a sua possível aplicação no dia-a-dia.

3.1 Requisitos do sistema

O sistema que se pretende conceber tem como objetivo encontrar uma solução eficaz para o processamento de dados, de forma a que seja possível fornecer indicações a um utilizador de um sistema GPS sobre as condições de tráfego em tempo quase real. Tipicamente, os Sistemas GPS calculam a forma mais rápida de chegar do ponto A ao ponto B e têm ainda em consideração, algumas preferências que o utilizador possa ter como por exemplo, evitar autoestradas ou evitar portagens. Contudo, nada garante que o caminho dado não esteja congestionado, ou até mesmo com alguma impossibilidade de trânsito, que faça com que o caminho sugerido não seja efetivamente o melhor percurso do ponto A para o B. Para contornar esta situação, é necessário ser possível conjugar a informação de tráfego em tempo real com alguma informação sobre o comportamento do tráfego em momentos anteriores e desta forma ter melhor informação sobre o tráfego nas estradas. O foco desta dissertação será conseguir lidar com um fluxo de dados sobre a localização de um grande número de veículos em tempo quase real e deste fluxo obter informação e devolver ao utilizador informação atual, informação sobre intervalos de tempo, comparação com o passado e previsões sobre o estado do tráfego.

Nesta dissertação será utilizado um conjunto de dados de táxis, sendo estes registos afetos à China, mais propriamente à Cidade de Pequim, que serão tratados de forma a facilitar a sua

rápida utilização. A escolha destes dados está relacionada com a cidade em si. Pequim é uma cidade muito movimentada, onde será possível obter uma amostra significativa de registos para estudar o comportamento do tráfego.

Estes dados foram retirados de [27] onde se realizou um trabalho de limpeza e pré-processamento, culminando o trabalho na criação de uma *Data Warehouse*. Deste trabalho, também foi usado um mapa com as estradas da Cidade de Pequim que também foi processado de forma a decompor cada estrada em segmentos (ou troços). Desta forma, assegura-se que cada segmento corresponde a um troço de estrada entre duas interseções. Cada interseção corresponde a um ponto de adjacência entre um grupo de segmentos e representa cruzamentos, entroncamentos ou rotundas em termos rodoviários. Desta forma, o mapa de estradas pode ser representado como um grafo, sendo por isso possível utilizar os algoritmos bem conhecidos na área de teoria de grafos.

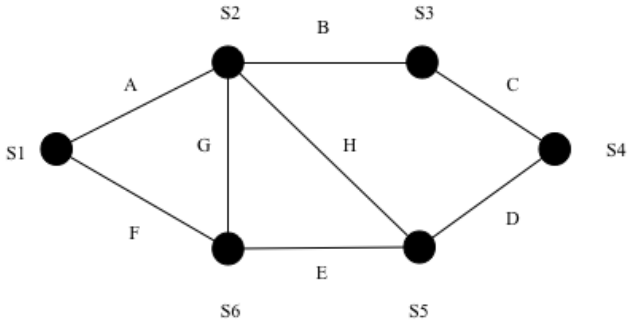
As localizações obtidas via GPS (pontos) têm um desvio, sendo que ao serem apresentados num mapa, podem não estar precisamente no local desejado (sobre a estrada). Para contornar tal problema, foi utilizado um algoritmo de *map-matching* para fazer com que os pontos sejam coincidentes com as estradas. Desta forma, foi possível construir uma tabela em que cada registo representa uma observação com a identificação e a localização de um táxi num determinado horário, assim como, a identificação do segmento de estrada onde se encontrava e a sua posição nesse segmento. O intervalo entre observações consecutivas de um mesmo táxi varia entre 30 segundos e cinco minutos. O valor mais comum entre amostras é de 90 segundos.

3.1.1 Descrição do contexto

Atualmente, já existem aplicações GPS disponíveis no mercado e na internet que disponibilizam informação em tempo quase real e que permitem ao utilizador alguma interação de forma a selecionar opções ao seu gosto, mas ainda não disponibilizam informação sobre o tráfego previsível num futuro próximo. Em termos de investigação, já existem métodos como os descritos em [14], onde se procura estimar o caminho mais rápido entre dois locais com base na previsão de tráfego no futuro. Contudo este, tal como outros trabalhos equivalentes, não abordam o problema do processamento do fluxo de dados de forma eficiente e escalável.

Neste contexto, pretende-se com este trabalho criar um protótipo que consiga dar resposta ao processamento do fluxo de dados, de forma a ser possível responder a perguntas sobre o comportamento do tráfego numa zona citadina em tempo quase real e fazer estimativas sobre o volume de tráfego previsível nessa zona no futuro próximo.

Com base nos dados sobre a localização dos táxis ao longo do tempo é possível fazer uma estimativa sobre o número médio de táxis por segmento de estrada e por hora. Estes dados permitem saber qual o volume de tráfego esperado em cada segmento de estrada numa região citadina em estudo, por exemplo, durante as horas de ponta e as horas normais, ou distinguir o volume de tráfego estimado em dias úteis e em dias feriado ou fins-de-semana. Além disso, os dados sobre a localização dos táxis ao longo do tempo também permitem fazer uma análise sobre as transições dos táxis entre segmentos que convergem numa interseção. Desta forma, é possível associar a cada interseção uma matriz de probabilidades de transição onde cada entrada representa a probabilidade de um táxi que segue num segmento (A) transitar para ao segmento (B) na interseção (S_2), tal como exemplificado na Figura 5.



Matriz de Probabilidade da interseção S2

	A	B	G	H
A	0	0,3	0,7	0
B	0,4	0	0,6	0
G	0,5	0,5	0	0
H	0,5	0,3	0,2	0

Figura 5 - Exemplo de um mapa de estradas com uma matriz de probabilidade

A informação descrita acima pode ser obtida pela análise de dados históricos, isto é, os registos sobre a localização dos táxis obtidos no passado. Por outro lado, se acrescentarmos ao sistema a capacidade de processar os dados sobre a localização dos táxis em tempo real (fluxo de dados), torna-se então possível manter uma estrutura de dados dinâmica indicando quantos táxis estão em cada segmento ao longo do tempo e que pode ser combinada com a informação extraída a partir dos dados históricos. Desta forma, torna-se possível fazer inferências sobre o estado atual do tráfego, por exemplo, identificar quais são os segmentos

com um volume de tráfego superior ao habitual, e mesmo fazer previsões para o futuro, tal como exemplificado na Figura 6.

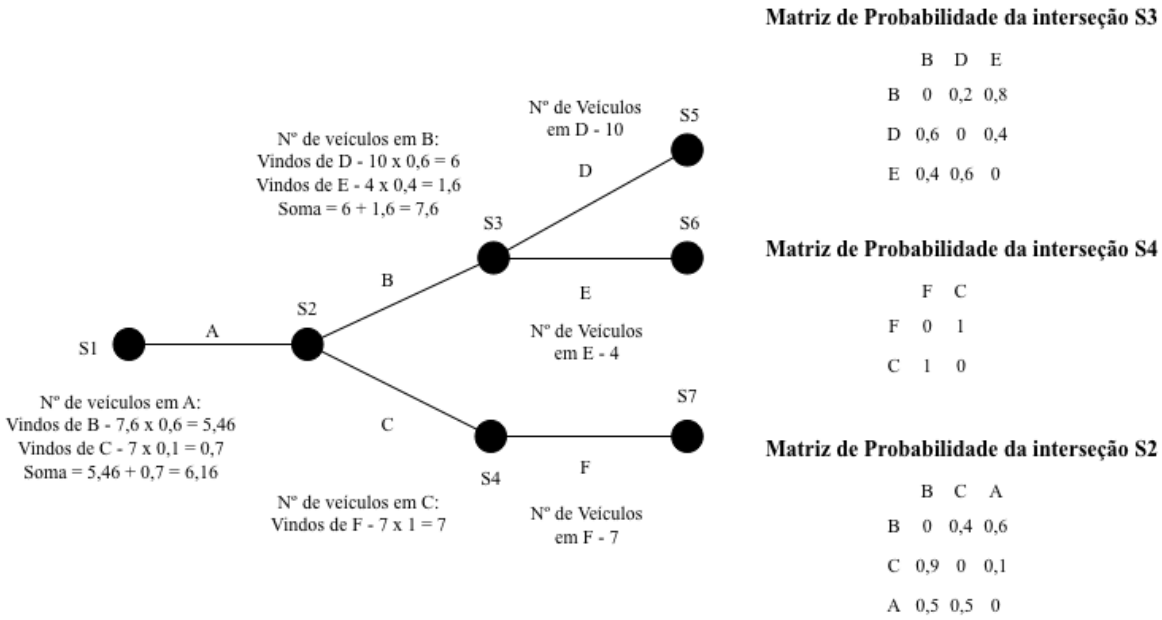


Figura 6 – Previsão do número de veículos no segmento A

A Figura 6 pretende responder à pergunta – Qual o número de veículos que irão estar no segmento A no futuro? Para tal é necessário calcular quantos veículos vão estar nos segmentos adjacentes de A – segmentos B e C. Nestes segmentos, o número de veículos vai depender dos segmentos D e E para o primeiro caso (B) e de F para o segundo caso (C). Como é possível ver em D, E e F existe a informação atual do número de veículos que com o apoio das matrizes de transição S_3 e S_4 permitem estimar o número de veículos em B e C. Esse número será a soma da multiplicação das probabilidades de transição entre segmentos por o número de veículos atualmente no segmento. Depois de obtida esta soma, o mesmo procedimento é repetido para B e C sendo que neste caso serão usadas as somas obtidas anteriormente e a matriz de transição S_2, obtendo no final o número estimado de veículos em A – 6,16.

3.1.2 Cenário de utilização

Esta secção descreve cada etapa do sistema e o que acontece em cada uma, de forma a que seja possível passar informação à próxima etapa. Será igualmente explicada a ordem de comunicação e como esta deve decorrer. A Figura 7 mostra o que se espera que seja uma aplicação final aplicada no contexto real com dados reais e o que se adaptou no protótipo.

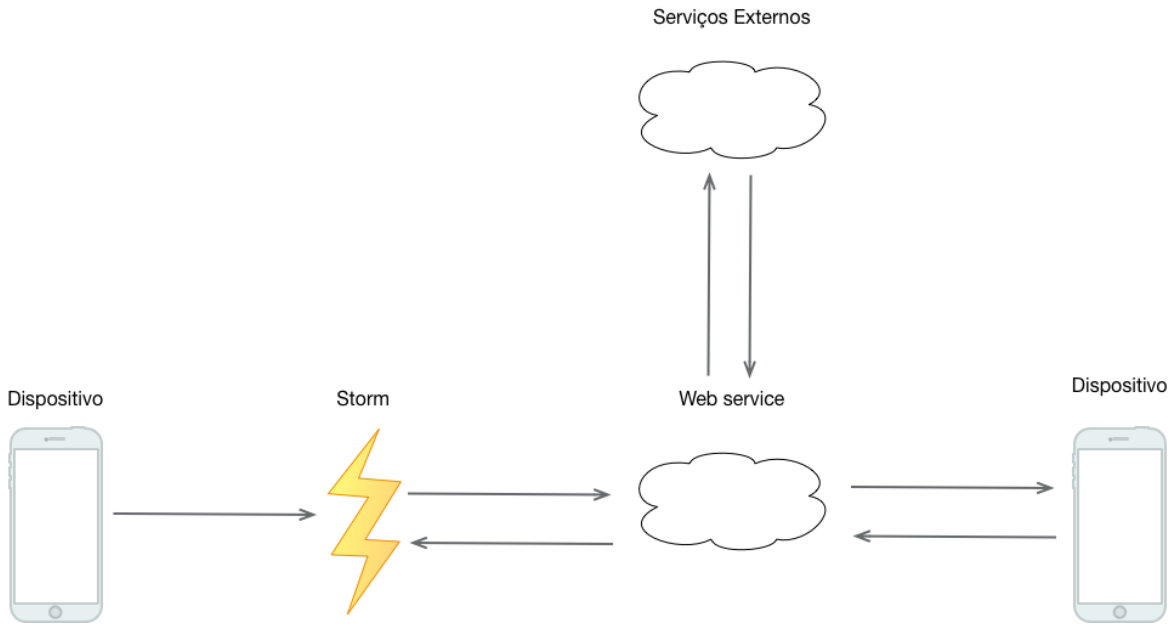


Figura 7 – Esquema de um cenário tipo de utilização

Num cenário de utilização real em que será possível obter dados reais a partir de dispositivos GPS instalados em táxis ou outros veículos, os dados sobre a localização do veículo são enviados para o Storm que irá processando os dados desse utilizador e de todos os outros utilizadores que usem a aplicação. Depois o Storm envia esses dados para um *Web service* que colecionará toda a informação sobre o tráfego automóvel numa região definida *a priori* e disponibiliza a informação conseguida de volta ao dispositivo consoante este necessitasse. O *Web service* poderia ainda usar ferramentas externas para obter dados que ajudassem nas inferências, como por exemplo o estado meteorológico.

Como é possível perceber o mesmo dispositivo irá comunicar tanto com o Storm como com o *Web service* sendo que para o primeiro envia informação e no segundo faz um pedido e recebe uma resposta.

No protótipo, para simular a entrada de dados será utilizado um ficheiro e para simular o pedido e receber a resposta será utilizada uma aplicação *Web*. Para além disso, nesta fase do protótipo ainda não houve interação com ferramentas externas, mas é importante perceber que essa interação terá sempre de ser na fase do *Web service* onde são realizadas as inferências.

Processamento do fluxo de dados (Storm)

Este sistema será dividido em duas partes sendo a primeira referente ao Spout e a segunda referente aos Bolts. O encadeamento destes dois elementos é definido na topologia, sendo que os dados provenientes do Spout serão distribuídos por vários Bolts usando um agrupamento aleatório como está representado na Figura 8.

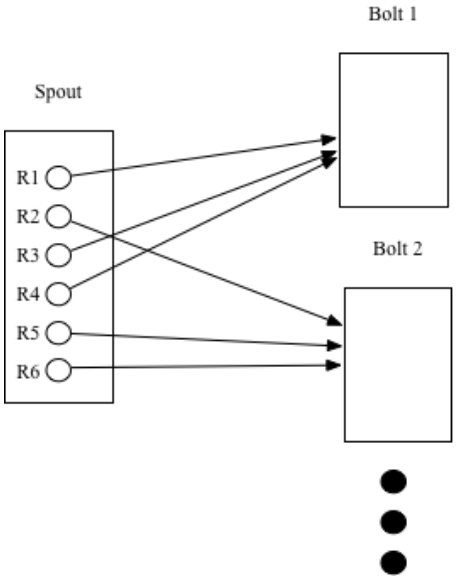


Figura 8 - Topologia utilizada no Storm

Como é possível ver na Figura 8, o Spout contém vários registos (R1, R2, etc) que serão enviados de forma sequencial para um Bolt escolhido aleatoriamente.

Se necessário é possível alterar a topologia de forma a melhorar o desempenho do sistema, sendo possível mudar a forma de agrupamento, aumentar o número de Spouts e/ou o número de Bolts, tendo em atenção que qualquer alteração de número está dependente do número de máquinas disponíveis e da sua capacidade.

Uma reflexão a considerar na conclusão desta dissertação consoante os resultados obtidos é se a estrutura do encadeamento dos Spouts e Bolts conseguiu responder ao propósito deste trabalho. Na estrutura de encadeamento, é possível que os dados ao saírem dos Bolts sejam enviados para outros Bolts para ser efetuado mais processamento ou análise nestes e por aí em diante.

Tratando-se de um protótipo será necessário simular a entrada de dados no sistema, sendo esse o ponto de partida levado a cabo pelo Spout. Este deverá ler um ficheiro com informação sobre o movimento de veículos, onde em cada registo está contida informação sobre o

veículo, o segmento, a localização e o instante temporal do registo, colocando essa informação numa estrutura própria. Esta estrutura será percorrida de forma a simular o movimento de veículos sendo os dados de cada entrada nessa estrutura posteriormente enviados para a segunda fase.

Os Bolts, que representam a segunda fase, ao receberem os dados devem proceder a uma análise sobre os mesmos de forma a permitir que sejam preenchidas ou atualizadas as duas estruturas de dados presentes nos mesmos. Essas estruturas representam respetivamente o número de veículos na última hora num determinado segmento e o segmento em que se encontra cada veículo.

Sempre que cada uma das estruturas é atualizada, é feita uma chamada ao *Web service*, a dois métodos distintos consoante qual das estruturas foi atualizada. Esta primeira análise e estruturação dos dados é benéfica de forma a retirar alguma carga ao processamento que seria feito no *Web service* mas também faz com que a informação chegue a este de uma forma mais estruturada e mais simples.

Estimação de resultados (Web service)

Este sistema que irá disponibilizar as respostas aos utilizadores é constituído por um *Web service* e está dividido em dois tipos de dados, o fluxo de dados recebidos em tempo quase real que é possível considerar como dados *online* e dados *offline* que estão guardados numa base de dados e que contém registos adquiridos anteriormente.

Alguns dos dados *offline* serão carregados na primeira vez que se inicializar o *Web service* para estarem acessíveis em memória e como já referido vão também ajudar a criação de algumas estruturas importantes como as matrizes de contadores, probabilidades e o grafo de segmentos. Desta forma, e estando em memória será computacionalmente mais rápido e fácil aceder aos mesmos, sendo que a sua utilização será efetuada com alguma regularidade para auxiliar os algoritmos criados. Os dados que forem utilizados com menos frequência serão pedidos diretamente à base de dados.

Os dados *online* estarão representados em estruturas que contêm o número de veículos presentes na última hora nos segmentos e o número atual de veículos nos segmentos. Estas estruturas devem ser atualizadas com os dados enviados em tempo quase real pelo Storm.

Este modelo de dados *online* e *offline* foi também referido em [14] onde na nuvem ao serem feitas as inferências era tido em conta o que estava a ser recebido com os dados já adquiridos, dados esses davam origem aos grafos baseados em segmentos de referência.

Com todos estes dados recolhidos e estruturados é possível então dar informação ao utilizador. Quando pedida informação sobre o número de veículos num determinado segmento ou o número de veículos que circularam na última hora num segmento, basta aceder às estruturas criadas e atualizadas para tal.

É possível também comparar o número de veículos em tempo quase real com a média de veículos nesse segmento e nos seus adjacentes, fazendo uso do grafo para descobrir os adjacentes, utilizando o número atual de veículos e por fim acedendo à base de dados para receber a média. Esta média é dada consoante a hora em que foi feito o pedido, existindo na base de dados para o mesmo segmento vários registos divididos por horas.

Quando se pretende saber o caminho entre dois segmentos essa informação é dada com apoio no grafo criado inicialmente e por fim é possível dar a previsão sobre o número de veículos num determinado segmento sendo utilizadas todas as estruturas que foram carregadas inicialmente, desde o grafo à matriz de probabilidades e utilizando ainda a informação atual sobre o número de veículos.

Este *Web service* será feito em SOAP. Esta escolha deve-se a motivos de segurança e integridade dos dados, para além de ser possível fazer a comunicação da melhor forma consoante o sistema em que for utilizado, não estando dependente de HTTP/HTTPS, como um *Web service* feito em REST. Também o encapsulamento da informação é importante, não sendo necessária a passagem pelo endereço de chamada como aconteceria no REST, sendo que neste momento quase todas as linguagens de programação têm métodos que permitem facilmente a gestão com o WSDL.

Sistema final (dispositivo do utilizador)

O sistema final é uma aplicação *Web* simples mas que permite simular o pedido de informação e tirar conclusões com as suas respostas. É também aqui que é possível testar se todo o processamento de dados está a funcionar sendo possível visualizar se os segmentos estão a ter fluxo de veículos e as suas contagens estão a mudar. É possível testar também o

funcionamento dos algoritmos e a rapidez da resposta dos mesmos para que no fim seja possível tirar conclusões.

Numa versão final, esta aplicação seria dirigida a dispositivos móveis sendo então necessário que o utilizador em poucos passos conseguisse aceder à informação que necessita. Desta forma, esta aplicação também foi pensada para ser possível ao utilizador ter interação com o mapa carregando nas estradas sobre as quais pretende obter informação e sendo-lhe desde logo disponibilizado o nome das mesmas. A aplicação contém ainda um pequeno menu onde é possível escolher que tipo de informação se pretende obter tornando simples e dinâmica a interação.

3.1.3 Requisitos não funcionais

A próxima lista apresenta os requisitos não funcionais específicos deste protótipo.

- **Escalabilidade** – esta característica, quando aplicada a *Software*, é uma característica desejável em todos os sistemas. Todos os sistemas devem ser capazes de aumentar os seus recursos caso seja necessário, para responder da melhor forma e com uma performance adequada à porção de trabalho que devem processar, quer seja para isso necessário aumentar os recursos de *hardware* ou *software*. No primeiro implicará um aumento de memória virtual ou física, entre outras alterações que possam ser feitas, no segundo incluirá um aumento de *threads* que ajudem a melhorar a resposta. Neste trabalho, a escalabilidade será, principalmente, de *software,* e estará presente na fase de processamento de dados sendo necessário ter em atenção o número de Spouts e Bolts que serão necessários para o Storm dar a melhor resposta possível. **Prioridade:** Alta.
- **Alta Disponibilidade** – pretende-se que o sistema esteja sempre disponível para dar resposta, garantindo que, sempre que os dados e as inferências que deles se pretendem obter sejam necessários para o funcionamento de uma aplicação externa, a resposta seja devolvida. Pressupõem-se então que devem ser reduzidas, ou até mesmo inexistentes, as situações em que o sistema esteja parado ou a devolver respostas de forma mais lenta. O sistema deve ainda conseguir solucionar os problemas de comunicação com o exterior – algum pedido em que seja necessário recorrer a um sistema externo – uma vez que estas situações não são controláveis. No sistema apresentado, este requisito pode então ser divido em dois problemas:

- **Spouts e Bolts** – caso um dos Spouts ou Bolts falhe, é necessário que a informação ainda assim consiga ser distribuída pelos restantes, ou então até entrar no requisito anterior e aumentar o número do elemento que falhou. **Prioridade:** Alta
- **Falha de comunicação com sistemas externos** – caso o sistema necessite de comunicar com algum elemento externo, como por exemplo aplicações para obter o estado meteorológico, este deve ter a capacidade de armazenar pelo menos os últimos registos que tenha conseguido obter. Prevenindo assim a falha de comunicação com estes elementos. **Prioridade:** Baixa

- **Modularidade –** esta capacidade é um requisito não funcional cada vez mais importante nas aplicações de *software*. Não basta apenas conseguir que o projeto esteja dividido em módulos e que no final todos juntos formem um só bloco, mas também permitir que possam ser criados e incorporados novos módulos no sistema sem necessidade de fazer uma alteração em tudo o que foi alcançado até ao momento. Neste sistema, o Storm é um bom exemplo de Modularidade onde para além de possibilitar a criação de topologias, é possível acrescentar mais módulos (Spouts e Bolts) dentro dos quais pode ser modelada a lógica pretendida, consoante a melhor opção para o cenário abordado no momento, atendendo à informação recebida e enviada. Desta forma, sem necessidade de parar o sistema, ou de fazer alterações ao mesmo, torna-se simples conseguir adicionar um novo módulo que processe a informação de maneira diferente, recorrendo apenas a pequenas configurações. **Prioridade:** Alta
- **Segurança** – a segurança é sempre um requisito importante, tornando-se mais ou menos sensível consoante a informação que circula dentro do sistema, ou consoante a importância do sistema no funcionamento normal no meio que o rodeia. Neste sistema, uma das principais preocupações quanto à segurança reside na informação, pois atravessa várias fases e deve garantir-se a integridade da mesma, assegurando que ninguém tenha acesso aos dados sem autorização. Apesar de não existirem dados pessoais em circulação no sistema, pois o identificativo dos veículos é atribuído pelo sistema, não deixa de ser uma informação com valor atendendo a todas as informações sobre os percursos dos utilizadores que podem ser extraídas e obtidas por terceiros. O Storm garante desde logo algum encapsulamento, com os tuplos

referidos anteriormente, adicionalmente o *Web service* garante também encapsulamento devido ao protocolo de envio WSDL, que é próprio dos *Web service* do tipo SOAP. Quanto à receção e envio de informação, é também necessário efetuar uma encriptação para garantir que não são afetados. **Prioridade:** Média

3.2 Arquitetura

Nesta fase da dissertação, irá ser explicada a arquitetura do sistema, mencionando as tecnologias utilizadas para o funcionamento da mesma. Serão também explicadas as estruturas de dados, que permitem que a informação seja guardada e transmitida de etapa para etapa. Desta forma será mais fácil a perceção do funcionamento interno do sistema, assim como das suas especificações e caraterísticas que o tornam distinto de um sistema tradicional de processamento de dados.

3.2.1 Modelo de domínio

O modelo de domínio descreve as estruturas de dados existentes ao longo de todo o percurso dos dados no sistema, sendo no entanto necessário fazer algumas considerações antes desta explicação. Normalmente, seria nesta fase que seria apresentado o diagrama da base de dados. Contudo, os dados que fazem parte deste sistema são diferentes: internamente, cada passo do sistema terá de ter uma estrutura de dados distinta, uma vez que deverá dar respostas a situações diferentes, ou seja, os dados de entrada serão sempre diferentes dos dados de saída, sendo que a etapa seguinte terá uma estrutura de dados diferentes da anterior.

Quanto aos dados externos como já referido estão armazenados numa *Data warehouse* e são provenientes de [27].

Dados de entrada

Os dados originais com o movimento dos veículos encontram-se em ficheiros com o formato CSV. Este é um ficheiro de texto comum onde cada linha representa um registo em que os valores estão separados por vírgulas.

As linhas são compostas por um primeiro valor que será o identificador da linha, seguido pelo identificador do veículo, a data em que o registo foi efetuado, a latitude e a longitude, o identificador do segmento onde circulava o táxi e os últimos dois valores voltam a ser a latitude e longitude, mas antes de lhes ter sido aplicado o *map-matching*.

Dados Storm – Spout

Cada registo do ficheiro CSV será colocado numa variável do tipo *list* que recebe esses dados em forma de classe e que contém os seguintes atributos:

- Classe *GpsDate*
  - Dataid (*integer*) – será um inteiro que servirá de identificador do registo.
  - Carid (*integer*) – o inteiro que serve de identificador do veículo.
  - Dt (*date*) – representa a data em que foi obtida a amostra da localização do táxi.
  - Dt_timestamp (*long*) – é a transformação da variável anterior num número do tipo *long*. É um auxiliar para o sistema e ajudará a comparar as datas sem ser necessário fazer uma comparação de todos os parâmetros da variável anterior. É um instante temporal em milissegundos.
  - Segmentid (*integer*) – identificação do segmento de estrada onde se encontrava o táxi no momento da amostra.
  - Lat (*long*) – valor da latitude obtido do algoritmo de *map-matching* de forma a que as coordenadas coincidam com a estrada.
  - Lon (*long*) – acontece o mesmo que na variável anterior, mas aplicando-se à longitude.

Dados Storm – Bolt

Nos Bolts, são criadas duas estruturas de dados em memória de forma a organizar a informação recebida anteriormente. Estes dados vão estar organizados numa tabela de dispersão (Map<key,value>) onde a chave (*key*) é um identificador único e o valor (*value*), pode ser ajustado consoante o que for necessário para armazenar informação. Estas estruturas de dados são dinâmicas, isto é, são atualizadas sempre que chegam novos dados.

Neste caso existirá uma tabela de dispersão para indicar qual o segmento onde um táxi foi observado pela última vez:

- carID – é a chave da tabela de dispersão e consiste num inteiro que identifica o veículo.
- roadID – é o valor e consiste no identificador do segmento de estrada em que o veículo se encontra.

A partir daqui são conseguidas várias interpretações e contagens na etapa seguinte, como por exemplo, saber quantos veículos se encontram numa determinada estrada.

A segunda tabela de dispersão permite representar a lista de táxis que se encontram em cada segmento de estrada:

- roadID – a chave da tabela de dispersão é o identificar do segmento de estrada.
- O valor neste caso é um *arrayList* da classe *segmentDate*, que tem a seguinte estrutura:
    - carID – valor inteiro que identifica o veículo.
    - timestamp –valor do tipo *long* que irá corresponde à data e hora em que o veículo foi observado pela última vez.

Com esta classe é possível tirar outro tipo de ilações, como, por exemplo, é possível saber quantos veículos passam num certo segmento num determinado intervalo de tempo.

As duas estruturas são parecidas e seria possível condensar todas apenas numa. A sua divisão prende-se com a facilidade de enviar informação para a etapa seguinte, e no facto de que o *Web service* não terá tanto trabalho para ter a informação disponível, tirando assim proveito dos Bolts.

De seguida na Figura 9, será apresentado o diagrama de classes que constituem o Storm, as interfaces de apoio às mesmas e as ligações que existem entre as classes.

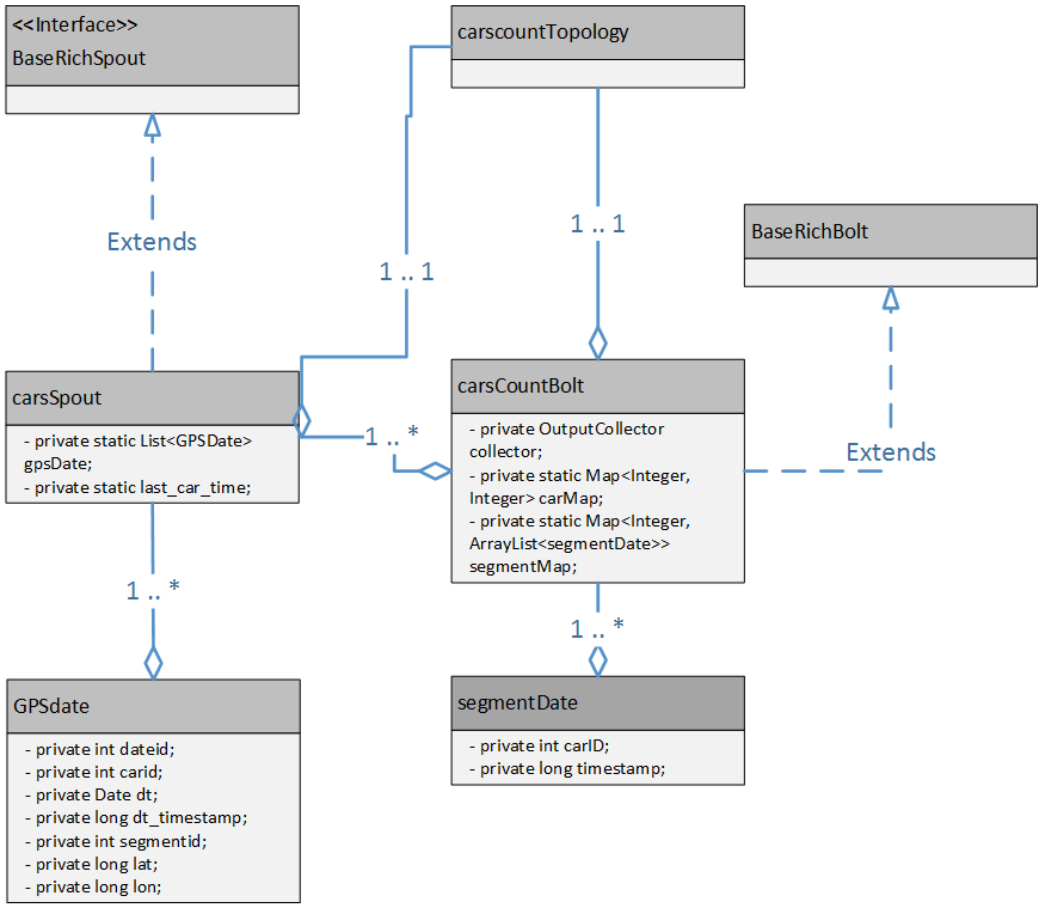


Figura 9 - Diagrama de Classes presentes no Storm

Dados do Web service

As estruturas de dados no *Web service* são preenchidas em dois momentos distintos: as estruturas que recebem os dados *offline* são carregadas quando o *Web service* é inicializado enquanto que as estruturas associadas ao processamento do fluxo de dados (dados *online*) vão sendo preenchidas ao longo do tempo, consoante a chegada de dados.

Como estas estruturas de dados são complexas, são apresentadas primeiramente com uma descrição breve e depois com recurso a um diagrama para ajudar a entender todo o processo. Os métodos e as ações desenvolvidas posteriormente por cada estrutura para ajudar no problema serão explicados no capítulo 4 desta dissertação – Implementação.

Será com o apoio destas estruturas que serão feitas as comparações e inferências necessárias para que seja possível dar informação ao utilizador, tendo sido criadas sempre na perspetiva de que é necessário ter acesso rápido aos dados. A sua composição e ligação entre elas tem de ser consistente para não criar problemas de consistência ou duplicação de dados.

Estruturas de dados que vão receber os dados *offline*:

- Classe *HashMatriz*
    - Matrizes – uma tabela de dispersão em que a chave é um inteiro que será o identificador de um cruzamento e o valor corresponde a uma estrutura de dados (*MatrizTrans*) explicada à frente.
    - MatrizGrafo – é um *array* bidimensional de inteiros que serve de suporte para criação de um grafo de conectividade entre os segmentos de estrada. Esta matriz será construída com todos os segmentos de estrada existentes e com base nas matrizes de contadores, contida na classe *MatrizTrans* que será explicada de seguida. Sempre que na matriz contadores existir um valor de contagem superior a zero é colocada nesta matriz o valor um, significando que aqueles segmentos estão ligados podendo ser bidirecionais, caso exista contagem na transição do segmento A para o segmento B e vice-versa ou unidirecionais, caso só exista contagem na transição do segmento A para o segmento B.
    - AllSegments – é uma estrutura do tipo *list*, neste caso uma lista de inteiros, que servirá de apoio à *MatrizGrafo* para a sua construção e preenchimento. Está lista irá conter todos os segmentos e a posição dos mesmos nesta lista será o número da linha e da coluna na matriz.
    - Graph – será uma estrutura do tipo *Graph,* que será explicada mais adiante. Esta estrutura representa o grafo de ligações entre segmentos sendo construída com base na *MatrizGrafo*, contendo apenas as ligações entre segmentos onde se tenha verificado a transição de veículos.
- Classe *MatrizTrans*
    - Contadores – é um *array* bidimensional de inteiros com o número de veículos que transitam de um segmento para outro segmento numa interseção, constituindo assim uma matriz de transição, onde as linhas e as colunas serão os segmentos que fazem parte de cada interseção.
    - Probabilidades – é um *array* bidimensional de números reais que representam a probabilidade de transição de um veículo de um segmento para outro segmento numa interseção. Existe uma matriz para cada interseção.

    - IdSegmentos – uma variável do tipo *list* que recebe inteiros. Esses inteiros serão os identificadores dos segmentos que constituem a interseção, servindo esta lista como apoio às duas Matrizes anteriores, sendo que o índice do identificador de um segmento nesta lista será o número da linha e coluna onde se encontra nas matrizes.
- Classe *Graph*
    - Map – é uma tabela de dispersão em que a chave consiste num nó do grafo (segmento) e ao valor é uma lista dos nós adjacentes ao mesmo.

Todas estas classes têm uma variável com o mesmo nome – serialVersionUID – esta variável será um identificador para a questão da serialização que será abordada no capítulo 4 – Implementação.

Os dados *online* estão organizados em duas tabelas de dispersão. A primeira tabela indicará o número de veículos presentes num dado segmento num dado intervalo de tempo e é constituída pelos seguintes elementos:

- roadID - a chave da tabela é um valor inteiro que consiste no identificador de um segmento de estrada.
- O valor é uma instância da classe *nCarstime* que é composta pelo seguinte:
    - NCars – número inteiro que indica o número de veículos que se encontram naquele segmento.
    - TimeStamp – que é uma variável do tipo *long* que representa a data e hora da última atualização do número de veículos.

A segunda tabela irá indicar o número de veículos atual num determinado segmento, composta pelos seguintes elementos:

- roadID – a chave é um valor inteiro que consiste no identificador de um segmento de estrada.
- nCars - O valor será o número de veículos que se encontram naquele momento exato no segmento.

A Figura 10 mostra as classes que compõem o *Web service* assim como as suas ligações.

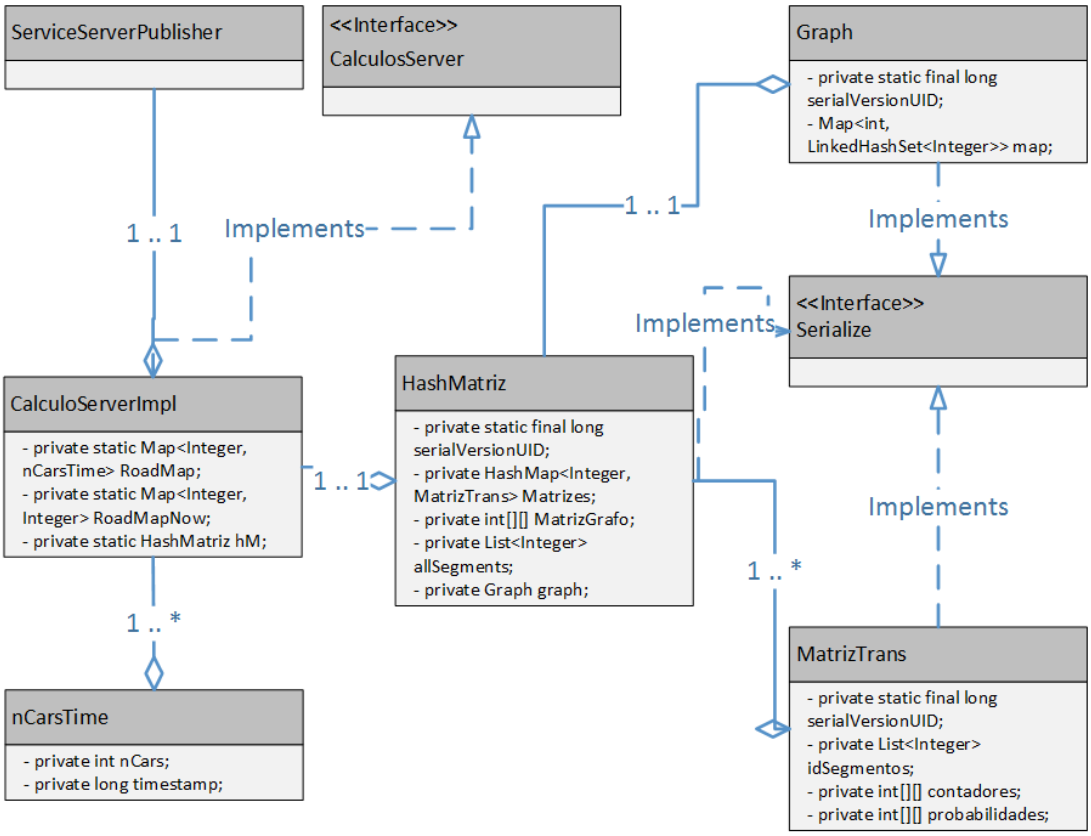


Figura 10 - Diagrama de Classes presentes no Web service

3.2.2 Modelo físico e tecnológico

Nesta secção, será apresentado o modelo físico e tecnológico do sistema. Para tal será utilizada a representação de um diagrama de implementação, onde estará demonstrada toda a estrutura do projeto sendo de seguida feita uma explicação do mesmo, assim como uma descrição de todas as tecnologias que são utilizadas em cada bloco e em cada ligação.

Diagrama de Implementação

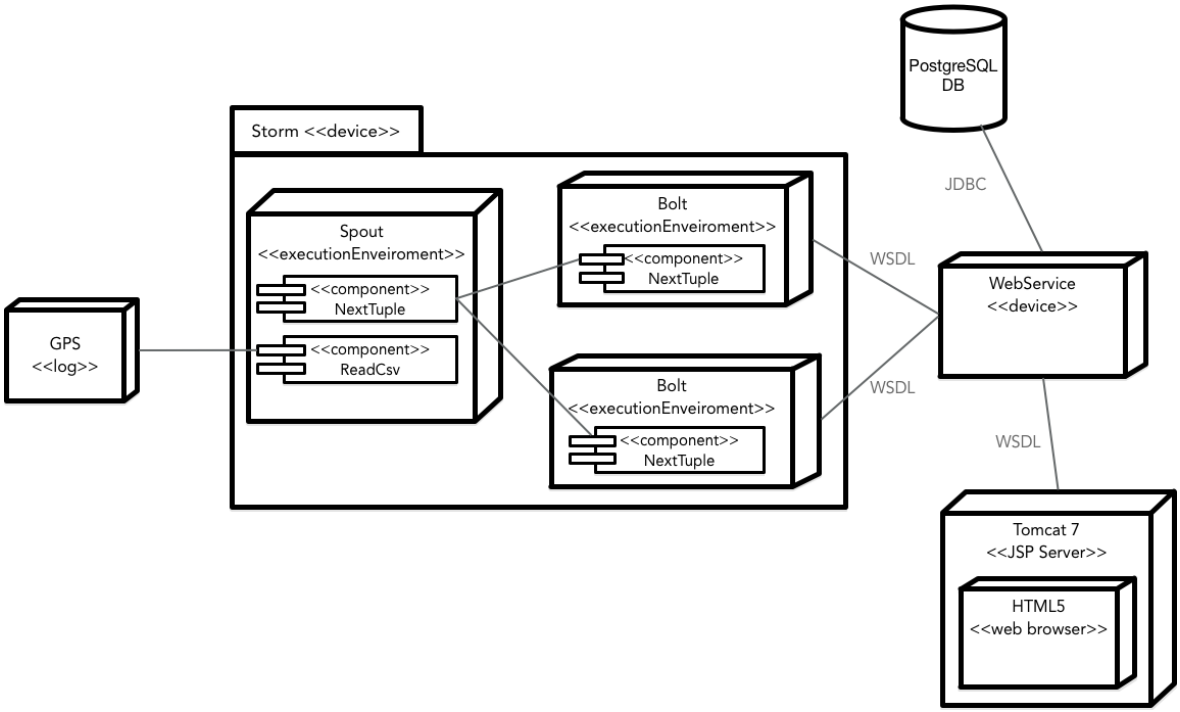


Figura 11 - Diagrama de Implementação do Sistema

Para ser possível testar todo o sistema é necessário simular algumas etapas, que não são possíveis de realizar em ambiente de produção, começando desde logo com o movimento de veículos e o envio de informação para o sistema. Sendo assim para que fosse possível alcançar algo semelhante ao movimento de veículos numa cidade e enviar coordenadas dos veículos ao longo do tempo, foi criado um GPS log com os dados de vários veículos – táxis – que serão lidos para depois ser possível simular a receção de informação em tempo real dos veículos em andamento na estrada.

Esses dados são então processados e lidos por um método que se encontrará no Spout do Storm que irá estruturar a informação sendo depois enviada para o Bolt. O envio de informação entre Spout e o Bolt é garantida por um método nativo do Storm que é responsável por enviar os tuplos com a informação entre os elementos.

No diagrama acima representado, foram apenas colocados dois Bolts, contudo é possível consoante a configuração do Storm ter mais que estes Bolts a funcionar ao mesmo tempo, assim como Spouts. Como já foi referido anteriormente é possível escalar toda a arquitetura do Storm, tendo sempre em conta a capacidade das máquinas que estiverem a ser utilizadas.

A leitura do ficheiro de simulação, assim como todos os processos do Storm são feitos com recurso à linguagem Java.

Os Bolts serão responsáveis por enviar a informação para o *Web service*, estando este último criado em SOAP e a comunicação com o mesmo será feita por WSDL.

A base de dados é onde se encontra a informação já recolhida sobre os veículos e onde o *Web service* deve guardar os dados que vai recebendo de forma a criar um histórico de informação. Para fazer a comunicação com a base de dados, será utilizado JDBC que é um conjunto de interfaces nativas do Java para comunicação com bases de dados, que neste caso faz o envio de comandos para as bases de dados de forma a permitir a comunicação.

Por fim, este irá comunicar com uma página *Web* também esta feita em Java, utilizando J2EE, que está a correr sobre um servidor Tomcat na versão 7. Para apresentação de informação nesta página é utilizado HTML5 e jQuery de forma a que seja possível alguma interação com a mesma e comunicação com a parte do servidor do J2EE.

4 Implementação

Neste capítulo será explicada toda implementação deste projeto desde a instalação e as suas dependências até à pormenorização da sua forma de funcionamento, explicando as classes que o constituem e a sua interação.

4.1 Maven

O Maven é um *software* criado pela Apache, utilizado para integração de projetos, ajudando na gestão de dependências e automatização de *builds*, apoiando assim no ciclo de desenvolvimento de um projeto desde a compilação, ao controlo de bibliotecas e documentação. A sua simplicidade por ter um mecanismo de configuração declarativa permite criar aplicações com um pequeno ficheiro de configuração.

A unidade básica de configuração do Maven é um ficheiro XML com o nome de POM (*Project Object Model*), que se encontra na raiz do projeto e no qual é descrito todo o projeto, incluindo informação sobre as dependências entre os módulos e componentes externas que possam existir, diretórios e *plugins* que sejam também eles necessários, assim como a ordem de compilação de toda a sua estrutura.

É nestas últimas características que foram especificadas que o Maven assegura a sua facilidade e a sua extensibilidade, para além de que pode ser utilizado com várias linguagens de programação (exemplos - Java, C#) e de já ter integração com vários IDEs como o exemplo do Netbeans e do Eclipse.

O Maven tem ainda algumas ferramentas para ajudar a melhorar e a acelerar todo o processo de criação de projetos, por exemplo, os *archetypes*. Em termos de *build*, ou seja, a fase de construção e distribuição da aplicação, o Maven é baseado no conceito – ciclo de vida –, tendo como ciclo padrão as seguintes etapas:

- Validar (*Validate*) – valida se o projeto está correto e tem toda a informação necessária disponível.
- Compilar (*Compile*) – faz a compilação do código-fonte do projeto.
- Teste (*Test*) – testa o código-fonte que foi compilado na fase anterior, usando uma estrutura do Unit Testing Framework.
- Pacote (*Package*) – usando o código compilado, garante a transformação num pacote de distribuição final, como por exemplo JAR.
- Verificar (*Verify*) – executa os testes de controlo de qualidade para verificar se tudo está em conformidade.
- Instalar (*Install*) – instala o pacote criado num repositório local, para poder ser utilizado como dependência em outros projetos locais.
- Implantar (*Deploy*) – é feito no ambiente *build*, copiando o pacote final para um repositório remoto para partilhar com outros programadores e projetos.

4.2 Implementação e funcionamento do Storm

O Storm funciona num sistema onde devem existir várias máquinas para distribuição de Spouts e Bolts, sendo isto conseguido com o apoio do Zookeeper que não só controla as interações entre os componentes, como tem a configuração dos mesmos, desde os endereços IP's de cada máquina, às portas que deve aceder entre outra informação importante para o funcionamento.

Nesta dissertação o foco passa por perceber toda a estrutura e arquitetura das topologias e testar as suas potencialidades e funcionalidades, para facilitar a manipulação das mesmas e também para uma maior facilidade em efetuar testes. O ambiente do Storm foi criado com recurso a uma máquina virtual Java (JVM) e o sistema foi desenvolvido utilizando o IDE Eclipse, para criar um projeto Java com o auxílio do Maven.

Devido ao uso do Maven, no ponto anterior foi já apresentada uma breve descrição do mesmo para facilitar o entendimento de alguns passos necessários para a criação do projeto e para garantir o seu funcionamento.

Para começar foi necessário configurar o ficheiro POM onde é colocada a dependência do Storm para que assim que compilado o projeto descarregue desde logo todas as bibliotecas necessárias para o funcionamento, como por exemplo o Zookeeper e a biblioteca *core* do

Storm. É possível consultar o ficheiro na secção anexos desta dissertação, acompanhado de uma explicação.

Para que o projeto mantivesse alguma organização, foram criados dentro da pasta "src/main/java", dois pacotes de forma a subdividir as classes que seriam necessárias para que o Storm funcionasse da forma pretendida. O primeiro pacote com o nome de "com.webservice.connection", será o local para as classes que fazem a ligação com o *Web service* que serão utilizadas nos Bolts e só com o seu auxílio será possível fazer a comunicação com as funções necessárias para que sejam enviadas as contagens dos veículos. De forma a criar estas classes de forma automática foi utilizado o seguinte comando - wsimport -s . http://127.0.0.1:9876/calc?wsdl - comando este que foi executado na linha de comandos do computador, dentro da pasta relativa ao pacote anteriormente referido. O comando constrói as classes necessárias para a utilização do *Web service* recorrendo ao seu WSDL, sendo que antes de executar o comando foi necessário publicar o serviço no endereço IP que se encontra no comando, que neste caso é o *localhost* na porta 9876.

O comando 'wsimport' é uma ferramenta Java, que está incluída no seu JDK.

O segundo pacote com o nome "com.storm.topology" tem as classes necessárias para o funcionamento do Storm. São criadas três classes – *carscountTopology*, *carsSpout*, *carsCountBolt* – para além das duas já referidas anteriormente (*GpsDate* e *segmentDate*).

4.2.1 Estrutura da Topologia

A classe *carscountTopology*, é a classe principal (*main*) onde é definida a estrutura que será utilizada pelo Storm, onde são definidos os números de Spouts e Bolts necessários para que o sistema funcione dentro daquilo que é expectável e onde é submetida a topologia para que a máquina virtual Java entenda e simule o seu comportamento. Para os testes iniciais, foram definidos um Spout que aponta para a classe *CarsSpout* e dois Bolts que apontam para a classe *carsCountBolt* onde estará definido o seu comportamento.

Após esta fase é submetida a topologia num ambiente grupo local, sendo atribuído o nome de "CarsTopology".

É possível ainda definir na topologia o tempo de funcionamento do sistema assim como comandos necessários para a sua terminação.

4.2.2 Estrutura do Spout

Na classe *carsSpout*, é definido o comportamento do Spout, sendo esta uma extensão da classe *BaseRichSpout*, que indica quais os métodos que necessita de implementar para o seu correto funcionamento. Para além dos métodos base que serão explicados de seguida, é possível ainda sobrepor outros como, por exemplo, "close*",* "ack"*,* "fail", dependendo da implementação e das necessidades do utilizador. Neste trabalho foram apenas utilizados os métodos base.

Os métodos base para o funcionamento da topologia são – "open"*,* "nextTuple" *e* "declareOutputFields" – sendo que a estes ainda foi adicionado o método "readCSV" que servirá de apoio para a simulação da circulação de veículos.

Apesar de não serem sobrepostos e terem o comportamento padrão, é importante entender os métodos "ack" e "fail". Estes servem para auxiliar a comunicação entre o Spout, o sistema e os Bolts. No caso do "ack"*,* funciona quando o tuplo que foi enviado for completamente processado enviando o identificador da mensagem que o Spout providenciou para o Storm, permitindo desta forma que o Spout saiba que pode avançar e que não existiram erros. Quanto ao "fail"*,* funciona caso haja um erro no processamento do tuplo, por exemplo um erro por ter sido esgotado o tempo (*time-out*) de execução e devido ao qual não tenha sido possível processar totalmente o tuplo. No caso do funcionamento do "fail" é enviado o mesmo identificador que no "ack" e o Spout fica com a informação que este falhou.

Quanto aos métodos sobrepostos, o "open" que é o primeiro a funcionar para iniciar o Spout, recebe três parâmetros: uma variável do tipo *Map* onde é possível colocar algumas configurações ou variáveis que sejam definidas na topologia, uma variável do tipo *TopologyContext* que contém informação sobre a topologia como o identificador do processo que está a correr e por fim uma variável do tipo *SpoutOutputCollector* que é responsável por emitir os tuplos. O funcionamento do Spout depende apenas da utilização da última variável descrita sendo as outras duas de utilização opcional, sendo assim apenas a variável do tipo *SpoutOutputCollector* irá iniciar uma variável do mesmo tipo contida no Spout.

Será nesta fase de iniciação do Spout que será feita a simulação de circulação de veículos, sendo para isso utilizado um ficheiro que contém informação sobre táxis na China, para simular a movimentação de veículos nos segmentos.

Esse ficheiro é lido pelo método "readCSV" que cria uma lista de objetos do tipo *GpsDate*. Os dados estão ordenados por data e hora, e para simular um fluxo de dados realista o método de leitura de dados tem um temporizador para controlar o momento de emissão de tuplos. Se quisermos simular uma situação real, o tempo de espera para emitir um novo tuplo deve ser igual à diferença entre a data e a hora (*timestamp*) do último registo lido do ficheiro com a data e a hora do registo anterior. Este processo pode ser facilmente acelerado ou retardado para simular cargas de processamento mais ou menos elevadas, multiplicando esse tempo de espera por uma constante. Se o valor da constante for zero, o tempo de espera será nulo e a emissão de tuplos é executada à velocidade que o processador das máquinas permitir e consoante o trabalho dos Bolts e o tempo que demoram a enviar os "ack".

O método "declareOutputFields" define a estrutura do tuplo que será emitido aos Bolts e que tem as seguintes variáveis:

- carID – o identificador do veículo do qual foi recebido o registo.
- roadID – o identificador do segmento em que o veículo se encontra.
- timestamp – a variável numérica que identifica o instante temporal em que os dados sobre a localização do veículo foram recolhidos.
- listSize – o tamanho da lista dos registos, aqui utilizado como um auxiliar, pois caso se tratasse de um produto final, esta variável não seria necessária. Com está variável é possível imprimir na consola do IDE quantos registos ainda existem, facilitando na fase de testes para perceber se a simulação dos veículos está a funcionar e para ser possível ter uma noção de quantos registos ainda faltam processar.

Para enviar informação que contenha datas, é sempre utilizado um instante de tempo que é por norma uma variável do tipo *long*. É necessário ter em atenção que existem vários módulos (Storm, *Web service*, Cliente), que é necessário transferir informação entre os mesmos e que estes podem não estar na mesma linguagem de programação. Apesar de não ser o caso deste projeto foi tido esse cuidado, considerando sempre que a linguagem de programação pode mudar e que cada uma delas lida com as variáveis que contenham datas de forma diferente, contudo todas elas têm variáveis primitivas como é exemplo o *long*.

Por fim, sempre que o método "nextTuple" receber um "ack" e caso não haja nenhum erro ou a topologia receba a indicação que foi terminada, o método vai ler o primeiro registo da lista obtida pelo "readCSV", enviar a informação retirada para o Spout com a estrutura que

foi declarada no método "declareOutputField" e alterar a variável *last_car_time* com o valor da variável *timtestamp* deste registo e ao terminar este processo apaga o registo.

4.2.3 Estrutura do Bolt

A classe que define os Bolts também é uma extensão de outra classe, *BaseRichBolt*, que lhe irá indicar os métodos que serão necessários para o seu funcionamento e que podem ser sobrepostos. Neste caso específico serão sobrepostos o "prepare" e o "execute". O método "prepare" é muito parecido ao "open" do Spout, e também este recebe três variáveis sendo o *Map* novamente a variável onde é possível acrescentar alguma informação como variáveis vindas da topologia. As restantes variáveis são o *TopologyContext* de onde é possível tirar informação sobre o contexto, por exemplo o identificador do processo que está a correr esta tarefa, e por fim o *OutputCollector*, um tipo de variável diferente da que existia no Spout, mas também esta responsável por emitir tuplos caso exista encadeamento e enviar o "ack" ao Spout ou ao Bolt, mais uma vez dependendo da estrutura. No caso específico deste projeto, uma vez que se trata do fim da linha de processamento, a variável do tipo *OutputCollector* será só responsável por enviar o "ack".

O método "prepare" é responsável por iniciar a variável do tipo *OutputCollector* assim como as tabelas de dispersão. A tabela responsável por indicar o segmento em que sucedeu a última observação de um táxi terá o nome de *carMap* e a tabela com a lista de táxis que se encontram em cada segmento tem o nome de *segmentMap.*

O método "execute" garante que o tuplo é recebido e consoante essa informação irá atualizar as tabelas de dispersão. Para a atualização da tabela *carMap* o método irá verificar a existência do veículo na mesma, caso exista e o segmento correspondente seja diferente do recebido então atualiza o segmento, caso não exista o veículo então adiciona-o com o segmento respetivo. Quanto à tabela *segmentMap* será necessário recorrer a dois métodos criados - "fillSegmentDate" e "checkArray" – para proceder à sua atualização. Estes métodos vão auxiliar a gestão da lista de veículos sendo que para este protótipo esta lista representará a lista de veículos que se encontram no segmento na última hora, sendo possível alterar este intervalo de tempo.

Sempre que as tabelas forem atualizadas são chamados os métodos do *Web service* "setCountNow", quando é atualizada a *carMap*, e "setCountByHour", quando é atualizada a *segmentMap*.

4.3 Implementação e funcionamento do Web service

O *Web service* é responsável por dar resposta aos utilizadores conjugando os dados recebidos do Storm com dados anteriormente recolhidos e retirando dados estatísticos que permitem responder a várias questões que podem ajudar o utilizador a entender o estado do tráfego e a tomar decisões. Como no Storm, também as classes foram divididas por pacotes, como forma de separar as classes que vão guardar o conhecimento adquirido anteriormente, das classes que estão a receber dados em tempo real e são também responsáveis pelo funcionamento do *Web service* em termos de comunicação.

Para começar vão ser explicadas as classes onde serão guardados os dados do conhecimento já adquirido, simplificando e ajudando depois a explicação de todo o processo de funcionamento do *Web service*. Este pacote tem o nome de "statisticsData" e contém as classes *MatrizTrans*, *HashMatriz* e *Graph*.

A classe *HashMatriz* será responsável por guardar os identificadores das interseções e cada interseção terá ligada a si uma classe *MatrizTrans* que irá conter a lista de segmentos que fazem parte daquela interseção. Com esta informação a classe *HashMatriz* poderá então construir o grafo de segmentos que irá apoiar algumas decisões que serão tomadas.

Outra nota importante, é que todas as classes que se encontram neste pacote implementam a interface nativa do Java *Serializable* permitindo assim que todas elas, sempre que necessário sejam guardadas de forma persistente em disco e sempre que necessário efetuar o carregamento (*deserializable*) das mesmas. Foi utilizado este procedimento para que os testes fossem feitos de forma mais rápida, pois um dos pontos que demorava algum tempo era a inicialização do *Web service* em que este carregava informação para estas classes.

4.3.1 Classe *MatrizTrans*

A classe *MatrizTrans* é responsável por guardar a lista de segmentos que fazem parte de uma interseção e terá também duas matrizes, uma matriz de contadores onde irá guardar a quantidade de veículos que transitaram de um segmento para outro e uma matriz de probabilidades que será preenchida após a primeira estar completa.

As matrizes neste caso serão sempre matrizes quadradas, com o mesmo número de linhas e colunas, nunca existindo apenas um segmento, pois se é uma interseção tem de unir sempre pelo menos dois segmentos e se une os segmentos, todos eles estão ligados entre si. Pode

ocorrer que segundo as regras de trânsito seja impossível passar de um segmento para outro – algo que se irá refletir na matriz de contadores – sendo a contagem zero nestes casos.

A posição dos segmentos nas matrizes será a mesma posição que o segmento terá na lista dependendo da ordem segundo a qual forem inseridos, sendo assim, se o segmento X estiver na posição dois, será essa a sua posição nas linhas e nas colunas, sendo que a posição da linha dois e coluna dois será sempre zero, constituindo assim a diagonal das matrizes.

Esta classe contém vários métodos, alguns de apoio e outros com alguma lógica de processamento. Como método construtor desta classe existe o método com o mesmo nome – "MatrizTrans" – que recebe um identificador do primeiro segmento que será adicionado à lista.

Os métodos de apoio que compõem esta classe são os seguintes: - método para adicionar segmentos à lista; - método para iniciar a matriz de contadores consoante o tamanho da lista de segmentos; - método que indica a existência de segmentos; - métodos que devolvem a contagem de transição entre dois segmentos assim como as probabilidades dos mesmos; - método para incrementar a contagem na matriz contadores; - método para devolver a lista de segmentos; - métodos que imprimem as matrizes;

Os métodos para imprimir as matrizes serão apenas utilizados na fase de teste para se compreender se os dados inseridos estão a ser colocados de forma correta.

De seguida serão apresentados os dois métodos que já envolvem alguma lógica:

- "setProbabilidades" – este método irá preencher a matriz de probabilidades percorrendo a matriz de contadores. O valor de cada célula nesta matriz é igual ao valor do contador da célula correspondente a dividir pelo somatório dos contadores nessa linha, ambos na matriz de contadores.
- "segmentConnection" – este método recebe o identificador de um segmento e de seguida irá percorrer a matriz de contadores na linha correspondente ao segmento recebido, para verificar com que outros segmentos existe um valor superior a zero, ou seja onde houve transição de veículos. Devolve uma lista de inteiros com todos os identificadores de segmento onde se verifica a condição anterior. Este resultado permite criar um grafo ligando apenas os segmentos onde existe contadores não nulos.

4.3.2 Classe *Graph*

Nesta classe será guardada a estrutura dos segmentos e as suas ligações, representando assim o mapa de segmentos. Para isso, existe uma estrutura de dados do tipo *Map* que guarda como chave o identificador de um nó (segmento) e o valor, uma lista de segmentos de estrada. A lista que contém cada chave será a lista de segmentos adjacentes.

Para ser possível adicionar e retirar informação do grafo existem os seguintes métodos:

- "addEdge" – um método que recebe dois identificadores de segmentos, em que o primeiro será a chave e o segundo serve para adicionar a lista de nós adjacentes. Este método é então responsável pela adição progressiva de nós (segmentos) ao grafo.
- "addTwoWayVertex" – mais uma vez, este método recebe dois identificadores de segmento e chama o método anterior duas vezes, invertendo a ordem dos identificadores. Desta forma o grafo recebe a informação que é possível ir de A para B e de B para A.
- "isConnected" – um método que recebe dois identificadores dos segmentos e que devolve uma variável do tipo *boolean*, indicando se o primeiro segmento está ligado, ou não, ao segundo segmento.
- "adjacentNodes" – recebe um identificador de um segmento e caso este exista no grafo, devolve a lista de identificadores de segmentos que se encontram adjacentes a este.

4.3.3 Classe *HashMatriz*

Será esta classe que irá interligar as classes contidas neste pacote e anteriormente descritas, de forma a ser possível aceder à informação e retirar os dados necessários para dar respostas à classe de implementação do *Web service*.

Para começar, quando esta classe é inicializada tem um método de construção, com o mesmo nome da classe mas que não recebe nenhum parâmetro e inicializa então o *HashMap Matrizes* que tem como chave um inteiro que será o identificador da interseção e o valor que será um objeto do tipo *MatrizTrans*.

Como na classe *MatrizTrans* existem alguns métodos que funcionam como apoio, apresentando uma lógica simples, alguns deles até aplicando métodos de outras classes,

existindo também métodos que apresentam mais alguma lógica inerente e que serão explicados mais pormenorizadamente.

Os métodos de apoio que fazem parte desta classe são os seguintes: - método para adicionar interseções e respetivos segmentos; - método que indica a existência de uma interseção; - métodos para inicializar as matrizes; - métodos para imprimir as matrizes; - método para incrementar a contagem na matriz de contadores; - método para devolver as probabilidades de transição entre dois segmentos;

Os métodos de impressão de matrizes como na última classe servem apenas de apoio para testes.

Para ser possível preencher a classe *Graph* e criar um grafo, que irá ser uma componente fundamental para as conclusões estatísticas, é necessário o cumprimento de algumas etapas que apresentam alguma lógica inerente e que se encontram divididas em três métodos:

- "createMatrizGraph" – este método irá percorrer todas as interseções contidas na variável *Matrizes* chamando o método que devolve a lista de segmentos da classe *MatrizTrans* de forma a obter todos os segmentos de cada interseção, adicionando-os depois à variável *allSegments* caso estes não existam na mesma. Por fim, inicializa a variável *MatrizGrafo* com o tamanho da lista *allSegments* e chama o método "fillMatrizGrafo".
- "fillMatrizGrafo" – este método irá percorrer todos os segmentos contidos em *allSegments*, percorrendo também todas as interseções e sempre que esse segmento existir numa interseção irá recorrer ao método "segmentConnection" da classe *MatrizTrans* para obter a lista de segmentos para os quais existe um valor superior a zero. Ao obter essa lista irá preencher com o valor um as colunas que representarem os segmentos obtidos na lista anterior na linha do segmento que se encontra a percorrer.
- "createGraph" – este método será chamado quando o anterior terminar e irá percorrer a *MatrizGrafo* linha a linha e sempre que encontra o valor um numa coluna vai adicionar à variável *graph* a ligação entre o segmento da linha com o segmento da coluna recorrendo ao método "addEdge" da classe *Graph*.

Relativamente a esta classe carecem ainda de explicação dois métodos que são recursivos.

Método de previsão do número de veículos num determinado segmento

Este método tem como nome "traverse" e irá devolver uma variável do tipo *float* que será a previsão do número de veículos num determinado segmento em função do número de níveis que forem pedidos. Nesta situação, os níveis representam quantos segmentos adjacentes ao segmento de interesse se pretende percorrer no grafo. Este método recebe cinco variáveis:

- N – é o identificador do segmento para o qual se quer saber o valor aproximado de veículos.
- Level – é o nível em que se pretende começar, por exemplo, é possível não querer contabilizar os segmentos adjacentes e neste caso o Level será um, sendo que para começar desde o início este valor deverá ser zero.
- LevelStop – é o indicador do nível em que a função deve parar.
- Array – será uma lista de valores que serve apenas de apoio ao método, guardando identificadores que o algoritmo não deverá percorrer.
- Map – é a estrutura de dados onde estão guardados todos os segmentos e os números de veículos que existem nesses mesmos segmentos naquele preciso momento.

A Figura 12 mostra parte do grafo de segmentos, onde se pretende obter o número de veículos que vão estar no segmento A com três níveis de segmentos adjacentes.

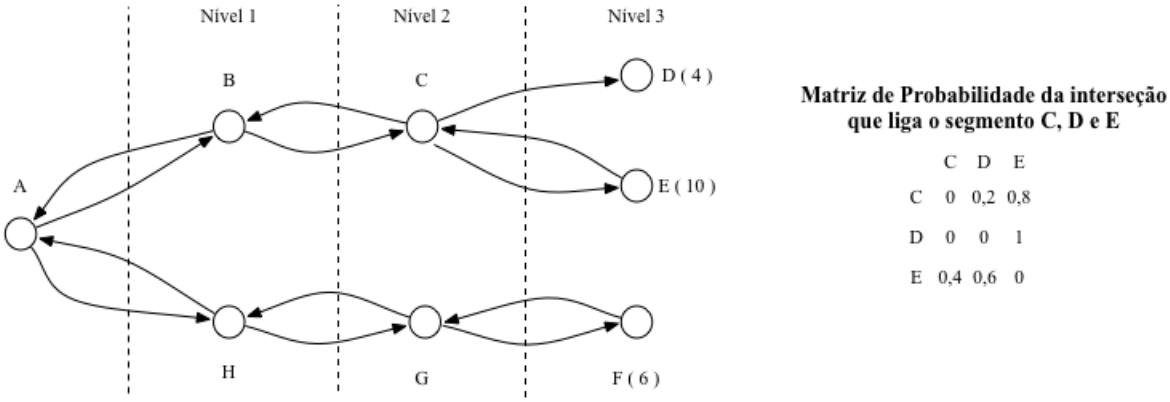


Figura 12 - Exemplo de um grafo de segmentos

As letras identificam os segmentos de estrada que são os nós do grafo e no nível três é possível ver entre parêntesis o número de veículos que se encontram no segmento D, E e F. Sendo este um grafo direcionado contém arestas direcionadas que indicam se é possível ir do segmento A para o segmento B e vice-versa, neste exemplo é possível ver que um veículo poderá ir do segmento C para D mas o inverso já não acontece. Ao lado do grafo existe o

exemplo de uma matriz de probabilidades de transição correspondente a interseção que une os segmentos C, D e E. Como é possível ver na matriz existe ligação entre D e E mas a mesma não se encontra reproduzida na Figura 12, porque essa ligação iria corresponder ao nível quatro de segmentos adjacentes onde ainda seria necessário percorrer os segmentos adjacentes de D e de E.

O algoritmo ao receber o identificador do segmento do qual se pretende a previsão – neste caso o segmento A – começa por percorrer os adjacentes deste segmento e para cada um deles irá percorrer os seus respetivos adjacentes até chegar ao nível três que neste exemplo corresponde ao nível final.

Ao chegar ao nível final, o algoritmo vai obter o número de veículos que se encontram nesse instante nos segmentos correspondentes a esse nível e vai multiplicar esse número pela probabilidade contida na matriz de transição, somando depois os números obtidos. No exemplo da Figura 12 o algoritmo ao alcançar o nível três, para a parte superior do grafo, iria multiplicar o número de veículos em D e E pelas probabilidades de estes transitarem para C, neste caso de D não é possível ir para C portanto a multiplicação daria zero e de E para C daria quatro. Sendo assim a soma seria quatro e o algoritmo iria voltar ao nível dois, onde iria multiplicar a soma obtida anteriormente pela probabilidade de transição de C para B e assim sucessivamente até chegar a A. O mesmo aconteceria para a parte inferior do grafo. O número de veículos em A seria a soma dos veículos da parte superior do grafo com a parte inferior do mesmo.

Método para encontrar os caminhos possíveis entre dois segmentos

Este método tem o nome de "getCaminhos" e serve para obter todos os caminhos possíveis entre o segmento A e o segmento B.

O algoritmo deste método vai percorrer todo o grafo e verificar as ligações entre segmentos para que seja possível perceber quais os segmentos que um veículo deve percorrer para alcançar o ponto B atendendo a que está no ponto A.

Este método recebe as seguintes variáveis:

- Visited – é a lista de segmentos que contém os segmentos já percorridos. Inicialmente irá conter apenas o segmento de partida.
- END – identificador do segmento para onde o utilizador se pretende dirigir.

- Paths – será uma lista que contém objetos do tipo *string* que irá conter todos os caminhos possíveis.

Contendo a variável *Visited* o segmento de partida, este método vai obter a lista dos seus segmentos adjacentes. Se algum desses segmentos adjacentes for o segmento da variável *END* o método encontrou um caminho possível, contudo não irá parar. Vai continuar com os restantes segmentos percorrendo a lista recursivamente até não haver mais possibilidades. Sempre que o método encontra um caminho ele transforma a lista *Visited* numa *string*, separando os identificadores por um hífen e coloca na lista de caminhos.

No pacote "statisticsData" já estão descritas todas as classes e métodos que fazem parte deste, faltando apenas o pacote "implementation", onde estão as classes responsáveis pelo funcionamento do *Web service*.

4.3.4 Interface *ServiceServer*, Classe *ServiceServerPublisher* e classe *nCarsTime*

Esta interface e estas classes serão explicadas em conjunto devido à sua simplicidade. A interface é onde estão descritos quais os métodos que o *Web service* irá implementar e que vão estar acessíveis aos serviços do exterior.

A classe *ServiceServerPublisher* é a classe principal (*main*) do *Web service* que contém apenas uma linha de código que é o método que publica o *Web service*, que neste caso recebe o endereço IP onde será publicado e a porta. No caso concreto desta dissertação será no endereço local (*localhost*) e na porta 9876 e o objeto que será implementado será a classe *ServiceServerImpl*, onde se encontra toda a implementação dos serviços.

A classe *nCarstime* tem a estrutura já apresentada na secção 3.2.1 em "Dados do *Web service*" e os seus métodos são apenas os de "Get" e "Set" das variáveis que a compõem. Servirá de apoio a variável *Roadmap* e ao método "getRoadCount", que serão explicados mais à frente.

4.3.5 Classe *ServiceServerImpl*

Esta classe será a responsável pelos algoritmos que cada método irá utilizar de forma a responder às questões do utilizador ou para receber informação vinda do Storm. Esta classe

implementa a interface *ServiceServer* que indica quais os métodos que tem de utilizar assim como quais as variáveis que cada um desses métodos recebe.

Quando a classe *ServiceServerPublisher* inicializa o *Web service* são carregados os dados *offline*. Para efetuar essa operação existem dois métodos:

- "setIntersect" – este método é responsável por aceder à base de dados e carregar para a variável *hM* que é do tipo *HashMatriz* todas as interseções e preencher a lista de segmentos que fazem parte da mesma.
- "setMatrixCars" – após o método anterior estar finalizado, é chamado este método que irá à base de dados buscar informação sobre os veículos e vai preencher as matrizes de contadores da variável *hM*. A informação que é adquirida da base de dados tem o registo dos veículos e os segmentos que estes percorreram. Esta informação é percorrida para se perceber o movimento dos veículos e sempre que estes transitam entre dois segmentos é necessário verificar na base de dados se existe uma interseção que ligue estes segmentos e caso haja é adicionado na matriz de contadores que houve um movimento do segmento A para o segmento B naquela interseção.

Devido ao espaço temporal entre os dados, pode não existir uma interseção que faça a união entre os segmentos que o veículo transitou, pois como os dados são recolhidos com alguns minutos de diferença o veículo pode ter avançado vários segmentos que não se encontram nos registos, sendo assim a verificação feita à base de dados no método "setMatrixCars" pode não obter resultados.

Após terminado o método "setMatrixCars", a matriz de contadores está completamente preenchida com os dados contidos na base de dados sendo então necessário chamar três métodos da classe *HashMatriz* para terminar todo o processo, pela ordem que se segue – "setProbabilidades", "createMatrizGraph", "createGraph".

Terminado este processo estão preenchidas todas as matrizes e criado o grafo de segmentos.

Para ser possível o Storm comunicar com o *Web service* existe os seguintes métodos:

- "setCountNow" – este método será responsável por preencher a estrutura *Map* com o nome de *RoadMapNow* que é composta por uma chave e um valor do tipo *integer* e que irá conter a informação sobre o número de veículos que se encontram em cada

segmento em tempo quase real. Este método recebe quatro variáveis que são: o identificador do segmento para onde o veículo transitou, o segmento onde estava antes, o tamanho atual da lista que está a ser processada no Spout e uma variável com o instante temporal em que foi obtido o registo. Estes dois últimos servem apenas para teste. O método tem quatro verificações e no final devolve ao Storm um *boolean* que é verdadeiro se tudo correr bem ou falso se algum erro tiver ocorrido. As verificações são:

  - Se a estrada nova existe e se antiga também, adiciona um veículo à estrada nova e retira um à estrada antiga.
  - Se a estrada nova não existe e se a antiga existe, adiciona a estrada nova ao *Map* com o valor um e retira veículo da estrada antiga.
  - Se a estrada nova existe e se antiga não existe, é adicionado um veículo à estrada nova.
  - Se nenhuma delas existir é apenas adicionado ao *Map* a nova estrada e colocado o valor um.

- "setCountByTime" – Este método irá preencher o *Map* com o nome *RoadMap* que é uma estrutura que contém como chave o identificador da estrada e como valor a estrutura de dados *nCarsTime*, e que desta forma irá guardar informação sobre os veículos que passaram num certo segmento num certo intervalo de tempo. Este método recebe quatro variáveis: identificador do segmento, número de veículos na última hora, o tamanho atual da lista que está a ser processada no Spout e uma variável com o instante temporal em que foi obtido o registo. Os dois últimos como no método anterior servem apenas de apoio. Este método verifica se o segmento existe no *Map* e se sim altera o número de veículos presentes na última hora na classe *nCarsTime* guardando também a hora que procedeu a essa alteração. Sendo assim será possível disponibilizar informação ao utilizador não só do número de veículos em tempo quase real num segmento assim como o número de veículos que circularam naquele segmento na última hora.

Para ser possível dar informação à aplicação cliente existem também os seguintes métodos:

- "getRoadCount" –Este método devolve o número inteiro de veículos que estiveram naquele segmento na última hora, verificando sempre se a última vez que a

informação do segmento foi alterada não é superior a uma hora comparativamente à atual. Caso seja superior, devolve o valor menos um.

- "getGraphPaths" – este método recebe dois identificadores de segmentos e devolve uma lista de caminhos entre os dois segmentos. Ao receber a lista transforma-a numa *string* com os caminhos separados por ponto e vírgula. Recorde-se que os caminhos que vêm na lista já estão separados por um hífen.
- "getTraverse" - chama o método "traverse" recorrendo à variável *hM* passando a esse método o identificador que recebeu do utilizador e devolve o resultado obtido pelo "traverse".
- "getRoadCountNow" - recebe como parâmetro o identificador de um segmento e devolve o número de veículos que se encontram nesse exato momento no segmento assim como a mesma informação sobre os seus segmentos adjacentes.

Para finalizar esta classe, existe ainda o método "roadState" que recorre à base de dados onde existe uma tabela que tem a média de veículos que passaram num certo segmento numa determinada hora. Usando essa informação, este método pode comparar o que está guardado na base de dados com os dados que o *Web service* tem sobre o segmento em tempo real. De forma a dar ainda mais informação ao utilizador será enviado também informação sobre os segmentos adjacentes ao segmento principal.

Sendo assim, este método verifica se existem nós adjacentes, e caso não existam vai buscar à base de dados a média de veículos para o segmento principal com base na hora em que foi efetuado o pedido. Caso exista informação, compara a média recebida com o número de veículos neste momento, havendo três hipóteses:

- O número de veículos é menor ou igual à média, então o *Web service* retorna uma *string* com o identificador do segmento e a palavra "*green*" (verde) separados por dois pontos;
- O número de veículos é superior à média, mas é mais pequeno que a média mais 25% da mesma (média + média x 0.25), retorna o identificador mas desta vez com a cor amarela (*yellow*);
- O número de veículos é superior à média mais 25%, devolve a cor vermelha (*red*);

As cores servem para dar indicação visual ao utilizador de como se está a comportar aquele segmento segundo a média normal de veículos que transitam na mesma. Para não ser apenas

verde e vermelho, ou seja, o número de veículos está mais baixo ou mais alto que a média, foi criado o momento intermédio que é a media mais 25% da mesma.

Caso existam nós adjacentes, o método vai fazer este mesmo processo para o segmento principal e para os seus adjacentes, criando uma *string* com a seguinte formatação – 1:green;2:yellow;3:red – em que o identificador está separado da cor correspondente por dois pontos e os segmentos estão separados por ponto e virgula.

Por fim, se não existir informação sobre algum dos segmentos a cor associada a esse segmento será a amarela.

4.4 Implementação e funcionamento da aplicação cliente

A aplicação cliente criada para comunicar com o *Web service* e fazer testes, é uma Aplicação *Web*, criada com recurso ao Java utilizando os seus projetos *Web* com o uso de JSP (JavaServer Pages) e Servelets.

As páginas em JSP são construídas com HTML5. Da vertente do servidor, serão utilizados os Servelets que vão comunicar com o *Web service*. A comunicação entre a vertente do cliente e a vertente do servidor será feita com auxilio ao jQuery e os seus métodos de "Post" e "Get". A página web terá o seguinte alinhamento.

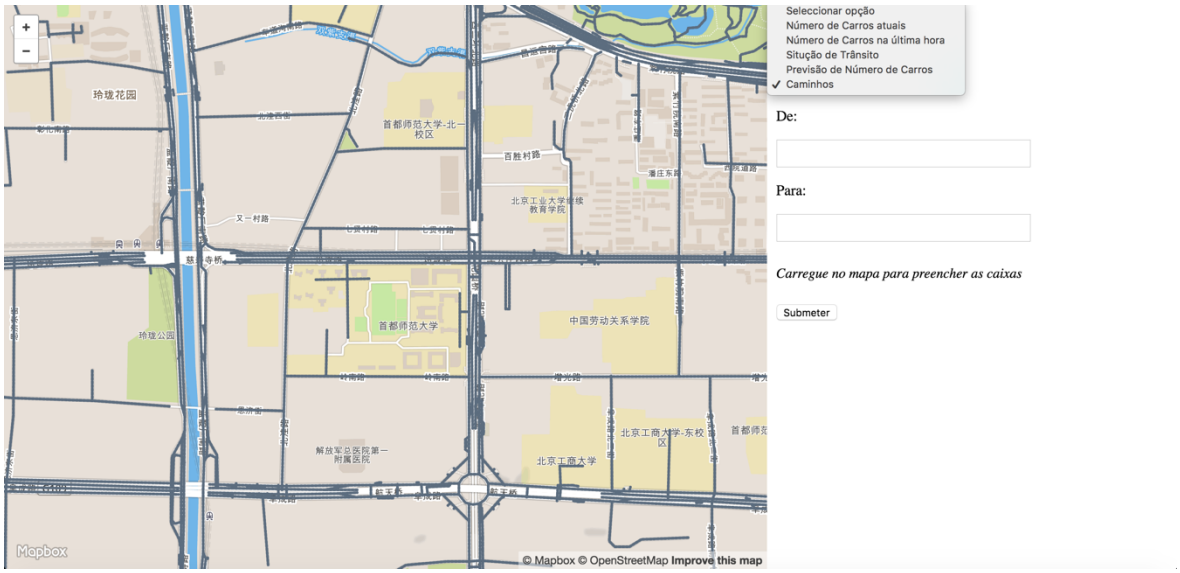


Figura 13 - Aplicação Cliente

Como é possível ver do lado esquerdo irá existir um mapa que foi criado com recurso ao MapBox. Para ser possível o utilizador ter interação com as estradas e clicar sobre as mesmas

estas foram desenhadas no mapa. Para tal foi necessário recorrer à base de dados e com ferramentas do PostgreSQL criar um ficheiro no formato geoJSON. Este ficheiro contém não só a localização das estradas, mas também o identificador que se encontra na base de dados que será importante para comunicar com o *Web service*.

Quando é iniciada a aplicação, o jQuery chama o método "createMap" que recorre ao método do MapBox "geoJson" que lê o ficheiro e desenha as estradas com a cor cinzenta tal como visualizado na Figura 13, dando-lhe uma largura maior para que seja possível clicar facilmente com o cursor. Também é acrescentado às estradas o evento clique que será um apoio ao formulário que se encontra à direita.

À direita temos um pequeno formulário para ser possível escolher uma opção, sendo elas: número de veículos atuais, número de veículos na última hora, situação do trânsito, previsão do número de veículos e caminhos.

De forma a que o utilizador não tenha de escrever os nomes das estradas ou saber algum tipo de identificador, sempre que é efetuado um clique numa estrada é preenchido o campo "DE" ou "PARA" (consoante a opção) com o nome da estrada. Para que o nome apareça, no jQuery é utilizado o método "Ajax" para comunicar com o Google e como cada estrada no mapa tem guardadas as coordenadas, o jQuery envia a latitude e longitude para o Google que devolve o nome da estrada.

Consoante a opção escolhida e depois de preenchido(s) o(s) campo(s) obrigatórios basta carregar no botão submeter e o jQuery comunica com os Servlets, usando o método "Post" e ao obter a resposta dará a informação ou no mapa ou apresentando uma mensagem caso seja uma contagem.

Os projetos Java na vertente Web têm um ficheiro chamado – Web.xml – onde contém a informação necessária para o funcionamento das páginas, como por exemplo qual será a página principal e no caso deste projeto qual o caminho para os Servlets. Cada opção da página apresentada, irá comunicar com um Servlet específico.

Os Servlets estão criados numa pasta com o nome "src" no pacote "com.webservice.servlets" e todas as classes Servlet devem ser extensões à classe *HttpServlet*. Todos eles têm sempre três métodos próprios dos Servlets, que será o construtor, o "doPost" e o "doGet". O "doPost" irá ser chamado sempre que houver uma chamada Post, que será o caso deste projeto e o "doGet" será chamado sempre que houver uma chamada Get.

Para além disso, foram sempre criados métodos para apoiar o método "doPost". Estes métodos recebem o(s) parâmetro(s) enviado(s) pelo jQuery para o "doPost" e fazem a chamada ao *Web service*, devolvendo o resultado "doPost" que por sua vez devolve para o jQuery.

Como foi explicado na parte do *Web service*, algumas das respostas vêm em *string* e para ser possível entender o resultado é necessário separar a *string* pelos separadores definidos. Essa parte será sempre feita pelo jQuery, para garantir também a interação com o Mapa e com a interface, sendo assim os Servlets da forma como recebem as respostas será a forma como as enviam.

Existe ainda um pacote, chamado "com.weservice.connection" que como no Storm irá conter as classes necessárias para comunicar com o *Web service*, classes essas criadas com recurso ao comando também já anteriormente referido.

5 Resultados

Neste momento o protótipo consegue simular o movimento de veículos e processar o fluxo de dados para no final enviar dados sobre o tráfego para um *Web service*. No *Web service* esses dados são estruturados e são aplicados algoritmos para ser possível obter informação para dar ao utilizador. Para simular a comunicação com o *Web service* foi criada uma aplicação *Web* que pede informação ao *Web service* e ao obter a resposta apresenta-a no ecrã. Este capítulo apresenta os testes que foram realizados para avaliar a solução desenvolvida.

Antes da apresentação dos testes e dos resultados, num primeiro ponto serão explicados mais detalhadamente os dados utilizados.

5.1 Estrutura de dados

Os dados utilizados nesta dissertação foram trabalhados por um aluno da Universidade de Aveiro no âmbito da sua dissertação de Mestrado [27]. Com base nesses dados, nos ficheiros utilizados e na base de dados criada, foram extraídos dados que auxiliaram ao desenvolvimento deste projeto.

De uma forma a segmentar a informação e a direcioná-la para as necessidades desta dissertação foram criadas diversas tabelas auxiliares. As primeiras tabelas, foram as tabelas que contêm informação sobre as interseções e os segmentos. A tabela das interseções contém os identificadores das interseções e as suas posições geográficas. A segunda tabela contém todos os segmentos, com o identificador dos mesmos e a informação geográfica destes.

Por fim, foi construída uma tabela com a informação das duas tabelas anteriores e que irá fornecer a informação sobre o facto de um determinado segmento pertencer a uma determinada interseção, tendo como colunas da tabela o identificador da interseção, o

identificador do segmento, as coordenadas no tipo de dados *geometry* e as coordenadas em texto dívidas por vírgulas.

Esta última tabela é utilizada no *Web service* para construir a lista de interseções e segmentos, e para determinar se dois segmentos convergem numa interseção.

Como o sistema se encontra ainda numa fase de testes, as duas primeiras tabelas foram criadas restringindo os valores a uma zona específica do mapa. Esta zona foi definida de acordo com as seguintes coordenadas: 116.3, 39.93; 116.4, 39.92 – e de forma a comparar zonas foi ainda definida uma segunda zona de acordo com as seguintes coordenadas: 116.41, 39.94; 116.47, 39.89 – sendo que para ambas as zonas foi efetuado o mesmo processo e criadas três tabelas relativas a cada zona. Nos testes apresentados de seguida será sempre utilizada a primeira zona apresentada, com exceção do último teste onde foram tidas as duas zonas em consideração.

Sendo a cidade de Pequim uma das maiores cidades do Mundo e tendo uma densidade de tráfego elevada, as restrições dos dados consoante zonas específicas ajuda a compreender melhor os dados e dessa forma facilita a implementação dos testes.

Após a execução deste passo, foi criado um ficheiro e uma tabela com dados dos veículos, que como já referido são táxis que percorrem a capital chinesa, Pequim. A criação desta tabela foi feita com recurso às tabelas já existentes que contêm dados que podem ser obtidos na internet [28]. Esta tabela deu origem a 578474 registos onde existiam com 551 táxis distintos e com 10098 segmentos diferentes. Na Figura 14 é possível ver uma amostra dos dados retirados a partir do QGis onde é possível ver os segmentos e os registos dos veículos.

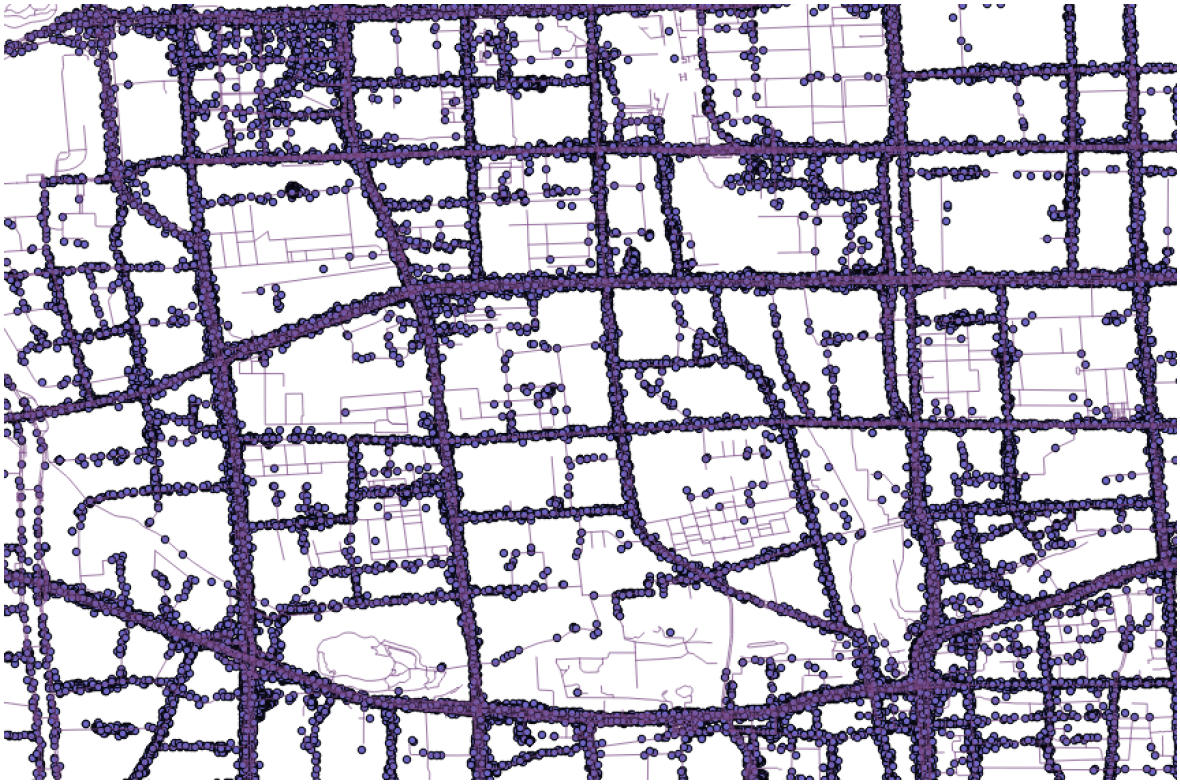


Figura 14 - Amostra de dados retirada a partir do QGIS

A tabela e o ficheiro têm o mesmo formato, sendo que na tabela os dados estão divididos nas colunas e no ficheiro por ponto e vírgula, e apresentam o formato explicado na secção 3.2.1 em "Dados de entrada". O ficheiro foi utilizado para simular a circulação dos veículos no Storm e a tabela foi utilizada para preencher as matrizes no *Web service*.

Por último, foi criada a tabela onde é possível obter o número médio por hora do dia de veículos em cada segmento com base nesses dados. Esta tabela tem como colunas, o identificador do segmento, a hora, o número médio de veículos, o comprimento do segmento, a velocidade média em metros por segundo e por fim a velocidade média em quilómetros por hora.

Infelizmente, devido à taxa de amostragem, a quantidade de registos em que se verifica o preenchimento das duas últimas colunas é muito reduzida, contudo caso estes preenchimentos se verificassem era possível ter adicionado esta informação à tabela dos segmentos e das interseções de forma a melhorar o método "traverse" que existe no *Web service*, tornando assim possível calcular mais rapidamente o tempo que é necessário para atravessar um segmento recorrendo ao tamanho e à velocidade média.

5.2 Número de veículos em tempo quase real

O primeiro teste efetuado com recurso à aplicação cliente, consiste em pedir ao *Web service* o número de veículos que se encontravam naquele instante num segmento específico.

Este teste foi feito em dois momentos diferentes para que fosse possível entender se havia movimento dos veículos, tanto na estrada principal como nas adjacentes.

No primeiro momento foi possível observar a resposta ilustrada na Figura 15.

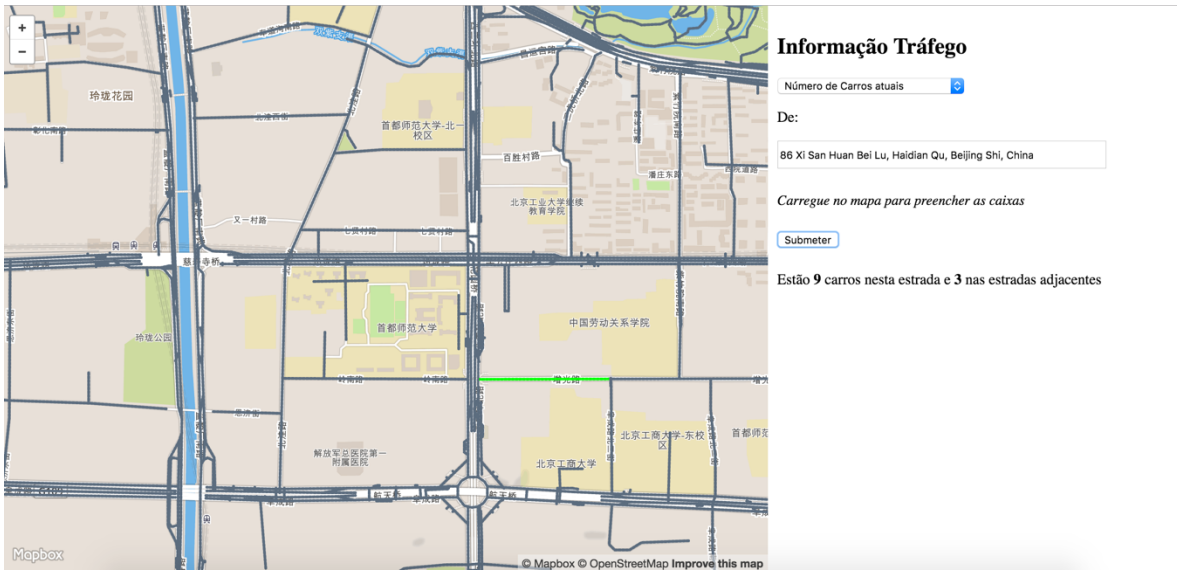


Figura 15 - Número de veículos: Teste 1

Como é possível verificar pela Figura 15, existem nove veículos no segmento colorido a verde e três veículos nos segmentos adjacentes ao mesmo.

Num segundo momento foram apurados os resultados ilustrados na Figura 16.

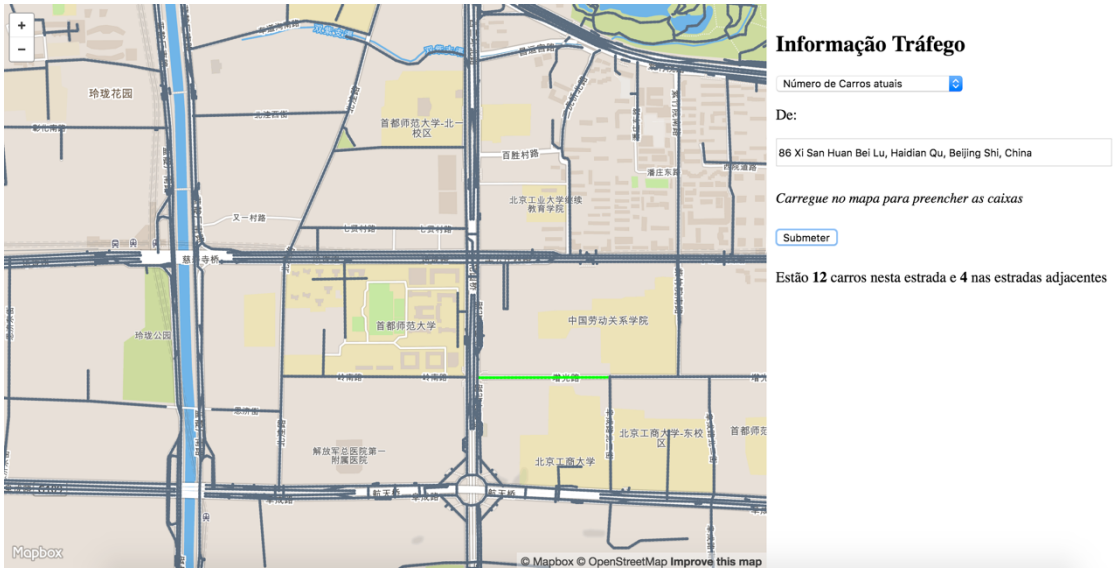


Figura 16 - Número de veículos: Teste 2

Este momento foi retirado, aproximadamente cinco minutos após o anterior e, sendo possível verificar que o número de veículos sofreu alterações tendo aumentado tanto no segmento principal como nos seus adjacentes.

5.3 Número de veículos na última hora

A informação obtida neste teste é complementar à informação obtida através do teste anterior, sendo estas informações importantes em termos estatísticos uma vez que podem ser conjugadas de forma a tirar conclusões. Para além de que também dão uma visão diferente ao utilizador sobre o estado da estrada anteriormente, podendo ele saber como se encontra a estrada neste momento e na última hora. Este teste para além de dar informação sobre a última hora ao utilizador, permite perceber se todo o processo que leva à obtenção destes dados está a funcionar corretamente, pois estes dados sofrem verificações no Storm e também no *Web service* quando é feito o pedido.

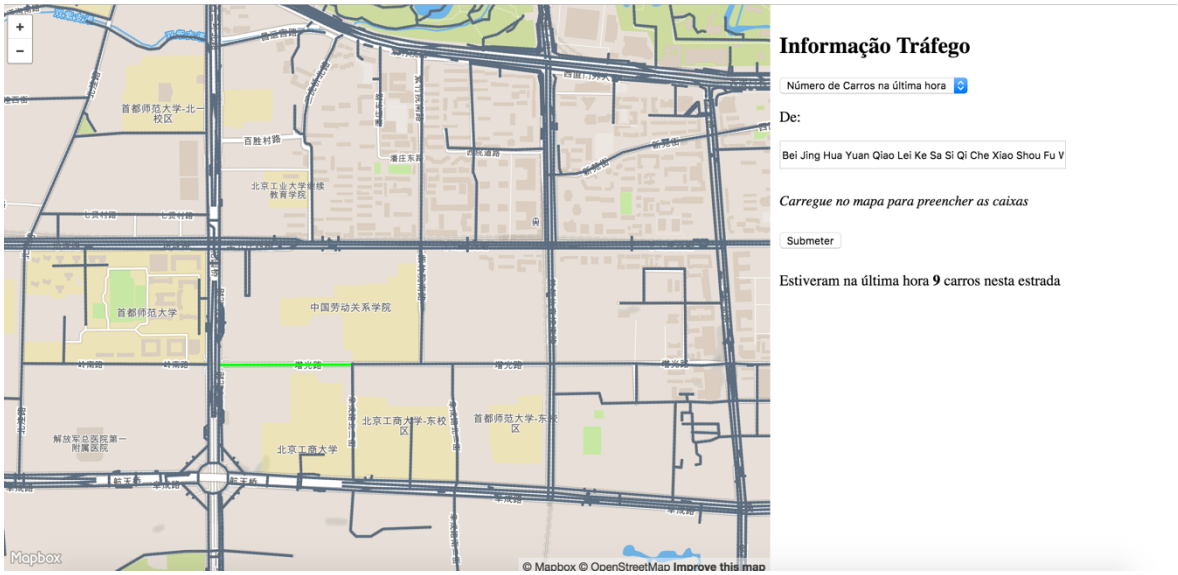


Figura 17 - Número de veículos na última hora

Como é possível ver na Figura 17, para aquele segmento específico que se encontra colorido a verde, circularam na última hora nove veículos.

Após este teste foi efetuado outro, sendo que neste caso se aguardou sensivelmente 60 minutos de forma a perceber se o segmento mantinha o mesmo número, verificando-se neste segundo momento que o número tinha aumentado ligeiramente para doze, havendo assim uma diferença de três veículos na média de veículos que circularam neste segmento na primeira hora em que foi efetuado o teste para o segundo teste efetuado uma hora depois.

O tempo de diferença entre os testes foi de 60 minutos para dar tempo ao Storm de atualizar toda a lista. Como já referido, o Storm atualiza a lista dos veículos que passaram na última hora e sempre que a data de um registo de veículo tiver uma diferença de uma hora com o registo que vai ser adicionado, o primeiro é removido. Com esta diferença, os veículos que se encontravam na lista no primeiro teste, já tinham sido removidos.

5.4 Estado dos segmentos comparativamente a registos anteriores

Este teste vai comparar o número de veículos que se encontram no segmento e nos seus adjacentes com os dados do conhecimento já adquirido (dados *offline*) que se encontram na base de dados. Neste caso o sistema irá recorrer à tabela que contém a média de veículos existentes nos segmentos em determinada hora.

Para possibilitar alguma interação com o utilizador e conforme explicado na secção anterior desta dissertação, o mapa foi pintado com cores diferentes para que fosse possível entender o estado dos segmentos, como pode ser visualizado na imagem seguinte.

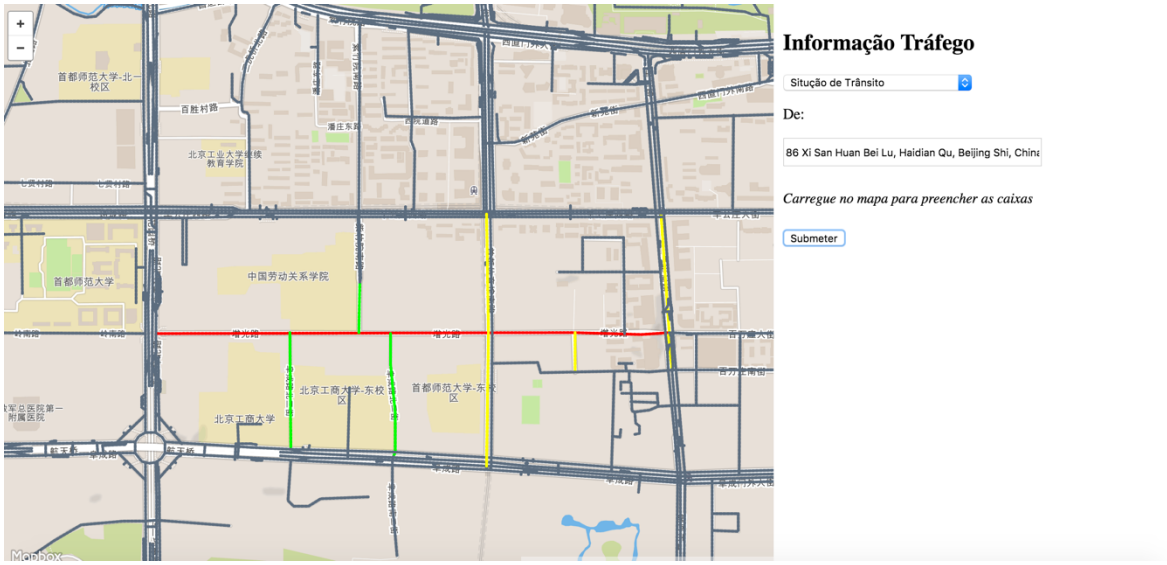


Figura 18 - Estado dos segmentos com recurso a cores no mapa

Como é possível ver na Figura 18 os segmentos foram coloridos consoante a resposta recebida. Neste caso o segmento a vermelho é o segmento principal e os outros são os adjacentes. Pela resposta dada e as cores no mapa é possível perceber que o segmento principal está com um valor superior à média estando a vermelho, enquanto os adjacentes alguns estão com valores inferiores e encontram-se a cor verde e outros num valor intermédio ou sem informação que têm a cor amarela.

5.5 Caminhos do segmento A para o segmento B

Os testes apresentados acima utilizam apenas a capacidade do grafo relativa ao conhecimento dos segmentos adjacentes. É necessário também perceber se a sua construção foi conseguida de forma correta.

Para tal, foi realizado este teste que usa todo o potencial do grafo, desde conhecer os adjacentes até percorrer o grafo de forma a encontrar os caminhos que unem o segmento A ao segmento B.

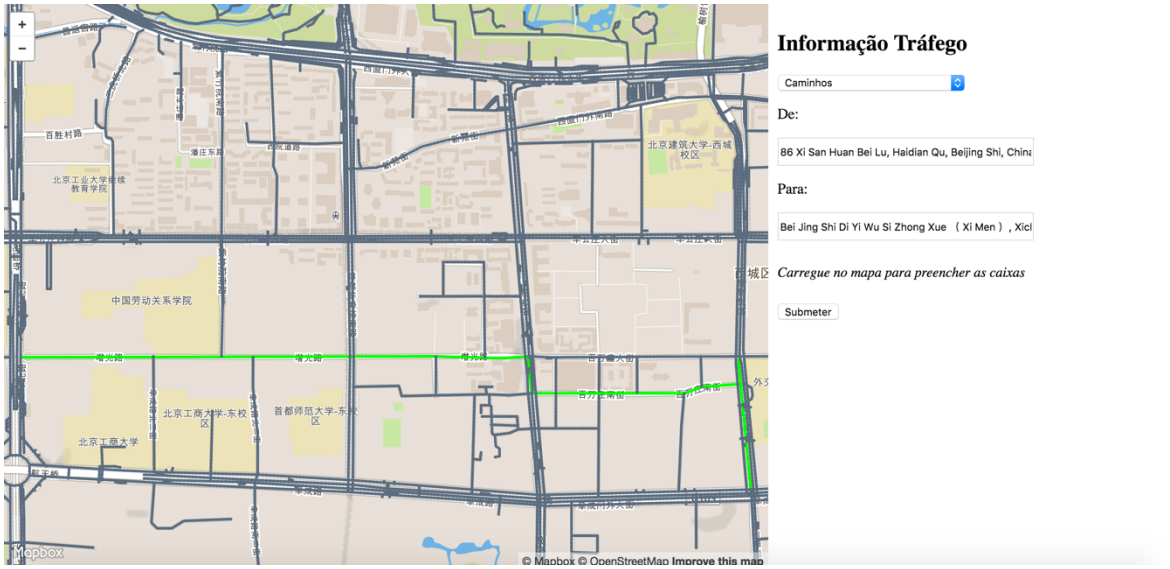


Figura 19 - Caminho entre o ponto A e o ponto B

Como é possível ver na Figura 19, o método para encontrar caminhos entre o segmento A e o segmento B deu como uma resposta um caminho possível (segmentos assinalados a verde) entre os dois locais. Com ajuda do Google Maps foi possível entender que este não é o único caminho que existe entre aqueles dois segmentos, prendendo-se esta questão com a forma como foi efetuada a construção do grafo. É necessário recordar que o grafo é construído segundo a matriz de contadores e só são adicionados os segmentos onde existe contagem de veículos, podendo haver mais que um caminho entre segmentos, contudo no caso de teste só existia um.

Um dos problemas que pode ocorrer e que justifica esta situação prende-se com o intervalo de tempo entre as amostragens de veículos. Por exemplo o veículo estava em A e depois em B, mas como a diferença de amostragens ronda os 30 segundos a cinco minutos, o veículo passou por um ou mais segmentos intermédios (entre A e B) sem que esses registos ficassem guardados. Este problema será, por isso, alvo de análise no teste seguinte.

5.6 Previsão do número de veículos

Com este último teste pretende-se que dado um segmento e usando os algoritmos estatísticos do *Web service* seja dada uma previsão de quantos veículos aí vão estar ao fim de alguns minutos. Sendo assim, é possível perguntar quantos táxis vão estar no segmento X dentro de 4 a 7 minutos tendo em conta o tráfego nos segmentos adjacentes ao segmento X e as matrizes de transição.

Para que fosse possível retirar algumas conclusões, este teste foi efetuado em duas zonas distintas do mapa de Pequim, com base nas tabelas criadas e explicadas no ponto 5.1.

Não havendo dados sobre o tempo médio que um veículo demoraria a percorrer um segmento foi necessário analisar nas duas zonas os segmentos de estradas que estivessem ligados entre si, de forma a entender quanto tempo em média demoraria um veículo a percorrer os segmentos até àquele que seria inicialmente testado. Nessa análise foi possível concluir que em média, três níveis de segmentos adjacentes equivalem temporalmente a aproximadamente sete minutos. Com este tempo é possível então correr os testes e esperar aproximadamente sete minutos para comparar os resultados da previsão com os reais.

No teste anterior foi possível perceber que poderia haver alguns problemas com a diferença de tempo entre as amostragens dos veículos, algo que ficou ainda mais comprovado com este teste, uma vez que após terem sido efetuadas as primeiras comparações foi possível entender que havia uma enorme discrepância entre os resultados obtidos pelo algoritmo e o número de veículos que se encontravam no segmento sete minutos após os testes. Este problema acontecia nas duas zonas tidas em consideração.

A Figura 20 representa parte do grafo onde é possível ver o porquê dos resultados se encontrarem díspares.

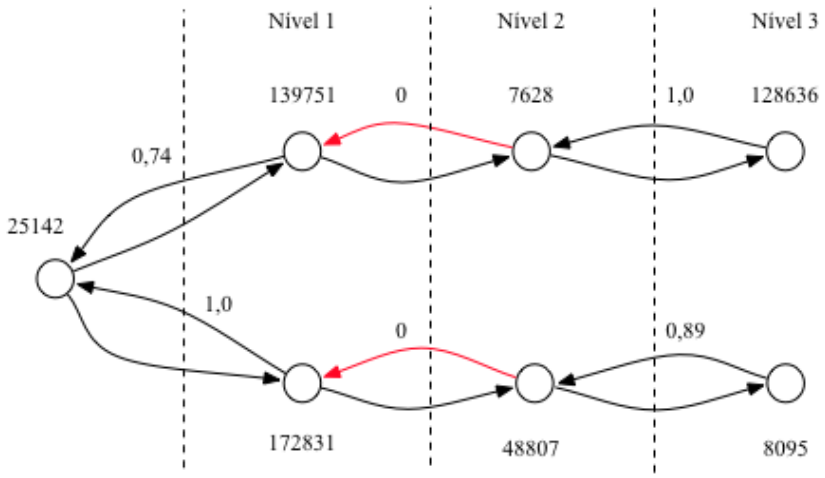


Figura 20 – Representação do erro em parte grafo de segmentos

O segmento principal do qual se pretende obter informação é o 25142, que contém probabilidades com os seus adjacentes, sendo uma delas de 1,0 o que significa que todos os veículos em 172831 têm de ir para 25142. Contudo como podemos ver no grafo que as transições de 7628 para 139751 e de 48807 para 172831 têm probabilidade zero, fazendo

com que os valores obtidos dos segmentos 128636 e 8095 não fossem tidos em conta porque para as matrizes de transição não havia ligação com os segmentos seguintes.

Isto podia ser considerado normal porque por exemplo 7628 é adjacente de 139751 porque a matriz de contadores demonstra que existiram veículos que foram de 139751 para 7628, contudo o contrário não acontece, podendo isto significar que 139751 era um segmento de sentido único.

Procedeu-se então a uma análise aos dados *offline* de movimento dos veículos com apoio do mapa criado para a aplicação cliente, onde foi possível constatar que o problema se encontrava no intervalo de amostragem entre os registos. Verificou-se que havia registos de veículos que estavam no segmento 7628 e passavam para 25142 sendo a única ligação entre eles 139751, contudo como neste caso a diferença entre registos era de três a cinco minutos não havia registo da passagem em 139751. O mesmo acontecia com os segmentos 48807 e 172831.

Para corrigir esta situação torna-se necessário fazer uma análise aos dados mais aprofundada, e perceber quantos veículos deviam ter transitado de 7628 para 139751 e de 48807 para 172831. Foram colocados os dados na matriz de contadores de forma a que quando fosse preenchida a matriz de probabilidades, esta já tivesse dados e não desse o valor zero. Esta correção foi realizada, mas não havendo uma solução automatizada para a conseguir acabou por ser uma tarefa demorada a nível de execução tendo sido apenas corrigida para os dados acima ilustrados.

Após a correção das matrizes foi possível obter os dados apresentados de seguida na Tabela 1:

	Número Previsto	Número Real	Hora	Diferença
1	17.84	13	10:07	4.84
2	19.01	16	15:36	3.01
3	20.91	18	20:44	2.91
4	24.64	19	03:03	5.64
5	22.64	21	11:15	1.64
6	25.69	23	16:10	2. 69
7	29.90	26	21:19	3. 90
8	32.18	28	05:55	4. 18
9	32.80	28	13:15	4. 80
10	35.82	31	18:24	4. 82
11	33.53	30	00:41	3. 53
12	34.13	32	12:22	2. 13

Tabela 1 - Comparação dos dados reais com os obtidos no método "traverse"

A primeira coluna da tabela contém a ordem porque foi executado o método "traverse", a coluna com o nome – Número Previsto – contém o resultado obtido pelo "traverse", a coluna com o nome – Número Real – é a coluna que contém o número de veículos que se encontravam nesse momento no segmento, a coluna seguinte com o nome – Hora – é a hora do dia em que se obteve os resultados anteriores e por fim a última coluna representa a diferença entre a coluna Número Previsto e a coluna Número Real.

Estes dados retirados correspondem todos ao mesmo segmento num período em termos de dias da semana compreendido entre domingo e quarta-feira e em diferentes horas de cada dia.

Como é possível verificar pela tabela anteriormente apresentada, depois de corrigido o problema nas matrizes o algoritmo aproxima-se muito dos números reais não existindo assim a discrepância inicial que demonstrava valores muito abaixo daqueles que se verificam na realidade.

Caso a frequência de amostragem fosse mais elevada e não tivesse existido o problema explicado no início desta secção, o ideal seria realizar um número de testes significativo – mil ou mais – para se poder tirar algumas conclusões quanto ao comportamento do tráfego em várias zonas. Infelizmente a correção deste problema é algo demorado que só foi realizado para este segmento em específico, onde apesar de tudo é possível perceber que o algoritmo consegue chegar até a profundidade pretendida e devolver resultados aproximados da realidade.

Este algoritmo está preparado para percorrer, em termos de profundidade até onde for necessário, contudo depois de ser encontrado o problema nos dados quanto mais longe o algoritmo fosse, mais segmentos iriam ter problemas e mais demorada seria a sua correção.

Ao contrário dos outros testes este foi maioritariamente feito ao nível do *Web service*, de forma a automatizar todo o processo, uma vez que era necessário de sete em sete minutos obter dados para fazer a comparação. Por seu turno, na parte do cliente para realizar uma tarefa deste género seria necessário utilizar o método "delay" ao nível do jQuery, método este que aguardaria sete minutos, não sendo possível realizar mais nenhuma operação enquanto se aguardava. Todavia, a aplicação cliente está preparada para chamar o algoritmo de previsão e mostrar a sua resposta.

Como conclusão deste capítulo é importante fazer uma análise aos tempos de resposta do *Web service* sempre que a aplicação cliente faz um pedido. Como é possível ver na Tabela 2, os tempos de resposta foram rápidos, mesmo nos testes em que era necessário aceder à base de dados ou em que a resposta não era direta e existia um algoritmo. É de realçar que todos estes testes foram feitos com toda a arquitetura a funcionar, estando o *Web service* a receber dados do Storm e a responder aos pedidos do cliente.

A primeira coluna da Tabela 2 representa o subcapítulo e a ordem dos testes efetuados nesta secção e a segunda coluna o tempo que demorou a resposta a chegar à aplicação cliente depois de feito o pedido.

Teste	Tempo de resposta em milissegundos
5.2 – Primeira vez	70
5.2 – Segunda vez	55
5.3	32
5.4	82
5.5	102

Tabela 2 - Tempos resultantes dos testes efetuados

6 Conclusão e trabalho futuro

Nesta dissertação foi proposto um sistema capaz de processar fluxos de dados de tráfego em tempo quase real, sendo que durante esse processamento se demonstrou possível fazer também uma análise sobre os dados.

Para a fase de processamento foi escolhido o Storm como plataforma, depois de uma análise sobre outras também disponíveis, tendo sido tomados em conta alguns aspetos importantes como a modularidade e a escalabilidade consoante o fluxo de dados assim o obrigue. Após testado e implementado este sistema, é possível afirmar que o Storm suporta e tem um bom comportamento nos aspetos acima citados. A facilidade com que é possível, utilizando os Spouts e Bolts, construir uma estrutura que se adequa ao objetivo pretendido no projeto a partir dos dados que são processados é uma das suas grandes mais-valias. A possibilidade de haver um encadeamento entre Bolts que seja capaz de subdividir o processamento, confere benefícios ao sistema tanto em projetos mais simples como em projetos com estruturas mais complexas que passem por vários passos.

É possível concluir que no cenário apresentado nesta dissertação, a estrutura de encadeamento usado – Spout-Bolts – apesar de simples, satisfez as necessidades não havendo perdas de informação e sendo rápido o fluxo dos dados apesar de haver um tratamento de dados nos Bolts. No futuro, caso haja necessidade de obter outras respostas pode ser possível encadear esta estrutura de outra forma para a análise sobre os dados ser feita de forma faseada.

A sua escalabilidade é outro ponto forte sendo possível a coexistência de vários Bolts a processar a informação, permitindo dividir a informação em grupos consoante seja necessário, podendo a divisão ocorrer de uma forma aleatória ou originando agrupamentos por variáveis.

A plataforma escolhida é ainda uma tecnologia em crescimento, encontrando-se apenas na versão 1.0.2, mas que tem todo o potencial para se tornar numa opção de confiança para o processamento de dados. Um dos fatores que revela estas vantagens é a documentação e exemplos de aplicação que atualmente se podem encontrar sobre a referida plataforma. Ao longo do desenvolvimento desta dissertação sentiu-se um grande crescimento da comunidade que usa o Storm nas suas aplicações, sendo que quando se iniciou esta dissertação o Storm se encontrava na versão 0.8, bem como um surgimento considerável de documentação, algo que no início desta dissertação era muito escasso. Assim, é atualmente possível obter muita informação sobre o Storm e até exemplos práticos, que impulsionam o seu uso enquanto ferramenta de processamento em aplicações.

Em termos de análise de dados, foi criado um *Web service* para funcionar como fornecedor de dados às aplicações cliente. Este *Web service* está dividido em duas partes, sendo que a primeira parte carrega informação já adquirida para a memória e a segunda recebe a informação do Storm. A segunda fase poderia ser pensada de outra forma, existindo um ponto intermédio entre o *Web service* e o Storm, por exemplo, uma base de dados. Esta base de dados guardaria toda a informação e o *Web service* teria apenas a função de ir buscar informação e processá-la retirando as respostas necessárias. Contudo, para o âmbito desta dissertação a opção de guardar em memória a informação em estruturas de dados próprias mostrou ser uma opção viável e talvez melhor que outro cenário.

Desta forma não foi perdido tempo nem capacidade de processamento em milhares de chamadas à base de dados, garantindo-se também que o tempo de resposta não seria muito elevado, sendo este último uma das grandes preocupações quando se está a tentar dar informação em tempo quase real.

É também necessário perceber que a quantidade de informação não cresce indefinidamente. Possivelmente, terá um grande crescimento inicial, mas uma vez existindo informação sobre todos os veículos ou todos os segmentos, a quantidade de informação a processar tenderá a estabilizar.

A aplicação cliente nesta dissertação serviu como forma de testar todo o ciclo do sistema, sendo contudo recomendável que para um trabalho futuro esta aplicação apresente versões *mobile* bem como deve conferir mais opções ao utilizador. Em todos os testes realizados na aplicação cliente, as respostas foram sempre dadas de forma muito célere, mesmo quando

havia algoritmos recursivos, estando sempre todo o sistema em funcionamento. O *Web service* apesar de estar constantemente a receber dados do Storm nunca respondeu de forma lenta ao cliente, mas no futuro é importante perceber se este comportamento se mantém com pedidos de múltiplos utilizadores.

Em termos dos dados utilizados, é de salientar a sua importância para a simulação de veículos em circulação, pois de outra forma seria necessário usar dados simulados. A utilização de dados reais também foi importante na fase de pré-processamento e a aquisição de informação sobre o tráfego, para perceber a complexidade dos algoritmos que era necessária para obter esta informação e colocá-la em estruturas adequadas, e finalmente utilizá-la nos algoritmos de análise e previsão de tráfego.

Foi no último ponto indicado que se conseguiu perceber a existência de alguns problemas nos dados. Os tempos de amostragem acabaram por se demonstrar inapropriados para a situação criando um problema para o qual nesta fase foi possível arranjar uma solução de forma a efetuar os primeiros testes. Contudo, como trabalho futuro será necessário arranjar uma solução mais viável e que possibilite a resolução do problema no geral.

Podem existir várias soluções para este problema, desde arranjar dados com um tempo entre amostragens mais pequeno ou então utilizar uma solução como a apresentada em [14] onde foram utilizados grafos baseados em segmentos de referência que conseguiam ultrapassar este problema ignorando, de certa forma, alguns segmentos que tinham menos quantidade de veículos nas amostragens.

Os testes revelaram-se positivos e demonstraram que é possível alcançar o objetivo apresentado na descrição deste problema, designadamente na questão do processamento de um fluxo de dados obtidos em tempo real e dar informação sobre o tráfego no momento e no futuro próximo.

Desta forma para trabalho futuro, é sugerido aplicar estes métodos no sentido de encontrar o caminho mais rápido entre dois locais e juntar mais informação na fase do *Web service*, como sejam o estado meteorológico e outras fontes de informação que possam revelar algumas condicionantes nos segmentos (por exemplo, obras na estrada), de forma a melhorar e a tornar cada vez mais corretas as previsões feitas. Esta melhoria, aplicada ao sistema aqui apresentado poderia ser um processo paralelo ao Storm que usaria os dados para calcular estes dados já enumerados ou outros dados considerados importantes.

Bibliografia

[1] C. Chede, “Você realmente sabe o que é Big Data? (Software, Open Source, SOA, Innovation, Open Standards, Trends),” 2012. [Online]. Available: https://www.ibm.com/developerworks/community/blogs/ctaurion/entry/voce_realmente_sabe_o_que_e_big_data?lang=en. [Accessed: 02-Jul-2017].

[2] S. Sicular, “Gartner’s Big Data Definition Consists of Three Parts, Not to Be Confused with Three ‘V.’” [Online]. Available: https://www.forbes.com/sites/gartnergroup/2013/03/27/gartners-big-data-definition-consists-of-three-parts-not-to-be-confused-with-three-vs/#5e16ce8d42f6. [Accessed: 20-May-2017].

[3] H. M. Klein Dominik, Tran-Gia Phuoc, “Big Data - GI - Gesellschaft für Informatik e.V.” [Online]. Available: https://www.gi.de/service/informatiklexikon/detailansicht/article/big-data.html. [Accessed: 24-Jun-2017].

[4] compare.com, “3 Best Traffic Apps for Android & iOS | WAZE, Google Maps & INRIX.” [Online]. Available: https://www.compare.com/auto-insurance/news/top-3-best-traffic-apps. [Accessed: 24-Jun-2017].

[5] L. Techlicious, “5 Best Navigation Apps - Techlicious,” 2016. [Online]. Available: https://www.techlicious.com/tip/best-navigation-apps/. [Accessed: 24-Jun-2017].

[6] Waze, “Aplicação gratuita de mapas, trânsito e navegação.” [Online]. Available: https://www.waze.com/pt-PT/. [Accessed: 24-Jun-2017].

[7] R. King, “Google Maps now covers 75% of global pop, 26 million miles of directions | ZDNet.” [Online]. Available: http://www.zdnet.com/article/google-maps-now-

covers-75-of-global-pop-26-million-miles-of-directions/. [Accessed: 20-May-2017].

[8] G. Developers, “Detalhes de cobertura de mapas | Google Maps APIs | Google Developers.” [Online]. Available: https://developers.google.com/maps/coverage. [Accessed: 20-May-2017].

[9] W. Gordon, “Google Maps Adds Incident Reports from Waze, Waze Gets Better Search,” 2013. [Online]. Available: http://lifehacker.com/google-maps-adds-incident-reports-from-waze-waze-gets-1171577008. [Accessed: 20-May-2017].

[10] W. Cunningham, “Autotelligent navigation service knows where you need to go - Roadshow,” 2016. [Online]. Available: https://www.cnet.com/roadshow/news/autotelligent-navigation-service-knows-where-you-need-to-go/. [Accessed: 20-May-2017].

[11] INRIX, “INRIX.” [Online]. Available: http://inrix.com/. [Accessed: 24-Jun-2017].

[12] “HERE WeGo Review — Gadget Syrup.” [Online]. Available: https://www.gadgetsyrup.com/home/2016/9/27/here-we-go-review. [Accessed: 20-May-2017].

[13] C. Barraclough, “Here WeGo app review: The best Android map app yet?,” 2016. [Online]. Available: https://recombu.com/mobile/article/here-wego-we-go-app-review-best-android-maps-transport#. [Accessed: 20-May-2017].

[14] J. Yuan, Y. Zheng, X. Xie, and G. Sun, “Driving with knowledge from the physical world,” in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’11*, 2011, p. 316.

[15] J. Yuan *et al.*, “T-drive,” in *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS ’10*, 2010, p. 99.

[16] L. X. Pang, S. Chawla, W. Liu, and Y. Zheng, “On detection of emerging anomalous traffic patterns using GPS data,” *Data Knowl. Eng.*, vol. 87, pp. 357–373, Sep. 2013.

[17] Y. Zheng and Yu, “Trajectory Data Mining,” *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 3, pp. 1–41, May 2015.

[18] D. Pfoser, N. Tryfona, and A. Voisard, “Dynamic Travel Time Maps - Enabling Efficient Navigation,” in *18th International Conference on Scientific and Statistical*

Database Management (SSDBM'06), 2006, pp. 369–378.

[19] D. Pfoser, S. Brakatsoulas, P. Brosch, M. Umlauft, N. Tryfona, and G. Tsironis, "Dynamic travel time provision for road networks," in *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems - GIS '08*, 2008, p. 1.

[20] E. Kanoulas, Y. Yang Du, T. Tian Xia, and D. Donghui Zhang, "Finding Fastest Paths on A Road Network with Speed Patterns," in *22nd International Conference on Data Engineering (ICDE'06)*, 2006, pp. 10–10.

[21] K. Sung, M. G. . Bell, M. Seong, and S. Park, "Shortest paths in a network with time-dependent flow speeds," *Eur. J. Oper. Res.*, vol. 121, no. 1, pp. 32–39, Feb. 2000.

[22] A. Storm, "Apache Storm." [Online]. Available: http://storm.apache.org/. [Accessed: 24-Jun-2017].

[23] A. Hadoop, "Welcome to Apache™ Hadoop®!" [Online]. Available: http://hadoop.apache.org/. [Accessed: 24-Jun-2017].

[24] A. Samza, "Samza." [Online]. Available: http://samza.apache.org/. [Accessed: 24-Jun-2017].

[25] A. Spark, "Apache Spark™ - Lightning-Fast Cluster Computing." [Online]. Available: https://spark.apache.org/. [Accessed: 24-Jun-2017].

[26] "Apache Storm Tutorial." [Online]. Available: https://www.tutorialspoint.com/apache_storm/index.htm. [Accessed: 25-Jun-2017].

[27] J. Semedo, "Sistema de recomendação de trajectos rodoviários orientado ao contexto" Universidade de Aveiro, 2015.

[28] Y. Zheng, "T-Drive trajectory data sample." 12-Aug-2011.

Anexos

Na página seguinte é apresentando o POM utilizado neste protótipo na etapa do Storm. É possível logo à partida verificar-se a presença da informação básica que são as primeiras propriedades desde a versão ao *groupid*, etc. Logo de seguida surge a configuração do *build* onde se verifica a existência de *plugins* com alguma informação sobre o mesmo desde a sua configuração até ao *groupdid* e *artifactid* e alguns diretórios onde é necessário colocar a informação. E por fim, surgem ainda as dependências, mais uma vez com alguma informação sobre as mesmas, onde é possível observar a existência da biblioteca do Storm e neste caso do JUnit caso fosse necessário a realização de alguns testes.

Poderia ser feita uma explicação mais extensa, ponto a ponto e fornecendo ainda mais exemplos sobre como pode ser configurado o POM. Contudo, para além de ser extensa toda esta explicação atendendo a todas as variáveis que podem formar o POM e todas as maneiras de que podem ser usadas, a configuração deste ficheiro deve ter em conta as necessidades do projeto.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" mlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>dissertacao.com.storm</groupId>
  <artifactId>dissertacao-maven</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>dissertacao-maven</name>
  <url>http://maven.apache.org</url>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.0.2</version>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <configuration>
          <archive>
            <manifest>
              <addClasspath>true</addClasspath>
              <classpathPrefix>lib/</classpathPrefix>
              <mainClass>com.storm.topology</mainClass>
            </manifest>
          </archive>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <executions>
          <execution>
            <id>copy</id>
            <phase>install</phase>
            <goals>
              <goal>copy-dependencies</goal>
            </goals>
            <configuration>
              <outputDirectory>${project.build.directory}/lib</outputDirectory>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
   <dependency>
      <groupId>storm</groupId>
     <artifactId>storm-lib</artifactId>
     <version>0.8.1</version>
    </dependency>
  </dependencies>
</project>
```

Figura 21 - Ficheiro POM do Maven na fase do Storm