A report on

INFO-H-415: Advanced Databases. Project.

By

Aleksei Karetnikov (000455065)

Content

Introduction

Nowadays, we are interacting with variably represented information: plain texts, tables, trees and others. Sometimes, it is impossible to imagine how it is possible to represent the information, which is exists as a tree in real world, in relative database. It is clear that one can implement lots of many-to-many connection and different attributes in such system but it takes more time to apply lots of joins in queries. The graph database can be a solution for such problems. It consists no opportunities to join different datasets and problems with performance – it is an anther NoSQL approach, which provides direct connections with data.

There are lots of different databases which deal with structured data: neo4j, Oracle, Postgres, ArangoDB, OrientDB and other. All of them provides some common and some unique features: query languages, store engine, system requirements, pricing, overall performant, etc. It is clear that NoSQL solution also requires a special language for interaction (e.g. AQL, Cypher, SPARQL and other languages).

For this research project, the ArangoDB was selected as a graph database. It looks a little bit different than the rivals: it is really young technology – it was developed in 2011 by German company ArangoDB GmbH, but it already has its customers. According to the developers' release, ArangoDB has the best performance in comparison with competitors with impressive opportunities in data interaction. Moreover, support of this technology is really simple.

Furthermore, any technology must have a commercial advantage. In other way, it is ineffective. According to the vendors' information, ArangoDB is the best solution because it is free for everyone and not so expensive when we need some additional features and support.

So, the main aim of this research project is to explain the graph database features, advantages and disadvantages in ArangoDB and its commercial opportunities.

Graph database

What is it?

Graph Database is a database type which stores data in vertices and edges of the graph. It based on the mathematical graph theory. In addition, each element of the graph database can consist any number of additional properties (additional key-value pairs). The main feature of such type of storage is highly connected data independently of the volume of the dataset. It means that all "join-like" operations are processing by using persistent connections between nodes, so it takes constant-time [2].

Nowadays this approach is becoming more and more popular. It is clear that we are living in the world of connected data, so such type of data structures significantly better represents the real information in the database.

The common place of data in such type of databases is nodes (vertex) of the graph. It can include different properties (usually, schema-less), label of the node and some additional metadata (such as index or constrains). In other type of data structures nodes are represented as records in relational databases or documents in document storage.

All the relations in graph databases (which are represented by edges) are named for semantic connection between node-entities and have the start and the end node. So, all the relations are directed. Normally, properties of the edges include information about distance between nodes, weight, cost, etc. On the one hand, it improves interaction with all the directly connected data but on the other hand simple deletion of connected node follows full deletion of the relationship [1].

There are many different approaches of storage mechanisms in graph database. Some DBMS store data in tables, others in document or key-value storages, which provides better compatibility with NoSQL approach.

The most famous graph databases are: ArangoDB, Neo4j, Oracle, OrientDB, SAP HANA, Teradata, SQL Server, GraphDB.

Differences between other technologies

The main aim of graph databases is storing of connected data to improve interactions between all the connected nodes. So, the main differences of such approach in comparison with other types of databases correspond to different aims of the databases and so, different data structures which are used to store it. In the table below the data types which could be recognized like the same in the most popular type of the databases are presented [1].

Type of the database	Name of the element
Relational (also temporal, key-value and spatial)	Row/record or tuple
Document	Document
Graph	Node
Object	Object

One of the main features of the graph databases in comparison with other type is better performance of interaction with connected data. Each of mentioned database type is the most relevant for particular case. For example, relational databases are the best solution for storing table values and its aggregation, document store for storing of the whole documents, object databases for file storing. Graph databases are simultaneously oriented for specific problems. In its case, it is finding of the shortest path, connected data storing, etc. So, the most relevant cases for using this technology are:

- real-time recommendations;
- decision making engines;
- social networks;
- access management;
- route planning;
- data and knowledge management;
- fraud detection.

It is clear that the vast majority of other NoSQL databases are aggregate oriented but frequently such operations are very expensive because of large volumes of data. In this case graph databases are the best solution for such problems because it is not necessary to spend resources for joining of different records, which are connected by default.

Query languages

One of the features of Graph database is a new problem of the query language selection. Unlike traditional relational databases which use SQL as a query languages, there are lots of different query languages for this type of databases, which are aimed for resolving of certain problems. The most famous query languages for graph databases are [2]:

- SQL (it still can be used);
- AQL (ArangoDB Query Language);
- Cypher (the most popular declarative language for Neo4j);
- GraphQL (the language which created by Facebook);
- Gremlin (language for Apache TinkerPop™ project. Can be used natively in Java, Scala, JS, Groovy, Clojure and Python);
- SPARQL (SQL-like language for RDF graphs).

SQL

SQL for graph databases does not contain some specific operators so it is not necessary to review it in details.

AQL

The most similar for SQL language is AQL, which was created especially for ArangoDB. In comparison with SQL, "AQL does not support data definition operations, such as creating and dropping databases, collection and indexes" [1]. Sometimes it overlaps SQL keywords, for example FILTER clause, which is equivalent to WHERE clause in SQL, but when all the queries in AQL are executed from left to right, so position of this clause in the query determines the precedence.

For example, the AQL code:

```
"FOR user IN users FILTER user.active == true RETURN user"
is the following SQL query:
"SELECT * FROM users WHERE active=true".
```

Cypher

Cypher is also SQL-inspired query language which was created for Neo4j database. The main advantage of this language is opportunity to describe the result which is necessary to obtain without describing of the necessary procedures, aimed to obtain the result.

For example, the code

```
"MATCH (s:Sales {items:'Beer'}) RETURN s.items, s.price"
corresponds to the following SQL query:
"SELECT s.items,s.price FROM sales AS s WHERE s.items='Beer').
```

GraphQL

GraphQL is a special query language for Facebook API. So, it is not directly specified for graph databases. The query of this languages specifies only object's structure, for example:

```
"{
product(quantity>100) {
      productName
      productPrice
      }
}".
```

The main users of this language are Facebook, Pinterest, Dailymotion, GitHub, Coursera, Intuit and Shopify.

Gremlin

The main feature of this language is native support of the most famous programming languages. Each query consists a sequence of atomic operations (steps) of the data stream. For example, the next code takes vertex with name “Austria”, moves to countries around and selects their names.

```
g.V().has(“name”,”Austria”).out(“border”).values(“name”)
```

SPARQL

This language was created by W3C consortium for RDF (Resource Description Framework, technology for data serialization and knowledge management, which is originally designed as a metadata model) graphs. The following code represents selection of the title in the given RDF triple:

```
SELECT ?title WHERE { <http://example.com/book/book1>
<http://purl.org/dc/elements /1.1/title> ?title . }
```

ArangoDB

Features

ArangoDB has lots of different interesting and important features:

- The most important one is multi-model (graph, key/value and document models) implementation which provides an opportunity to interact with different models in single query. Such approach helps to improve the performance of the database.
- Reduced operational complexity, which improves the selection of the most appropriate model for given data.
- Support of ACID (Atomicity, Consistency, Isolation, Durability) transactions.
- Modular architecture which provides fault tolerance.
- Low cost of ownership.
- Integrated framework ArangoDB Foxx, which provides native access to in-memory data by using JavaScript.
- Horizontal and vertical scalability.
- WEB UI for graph visualization and AQL queries.

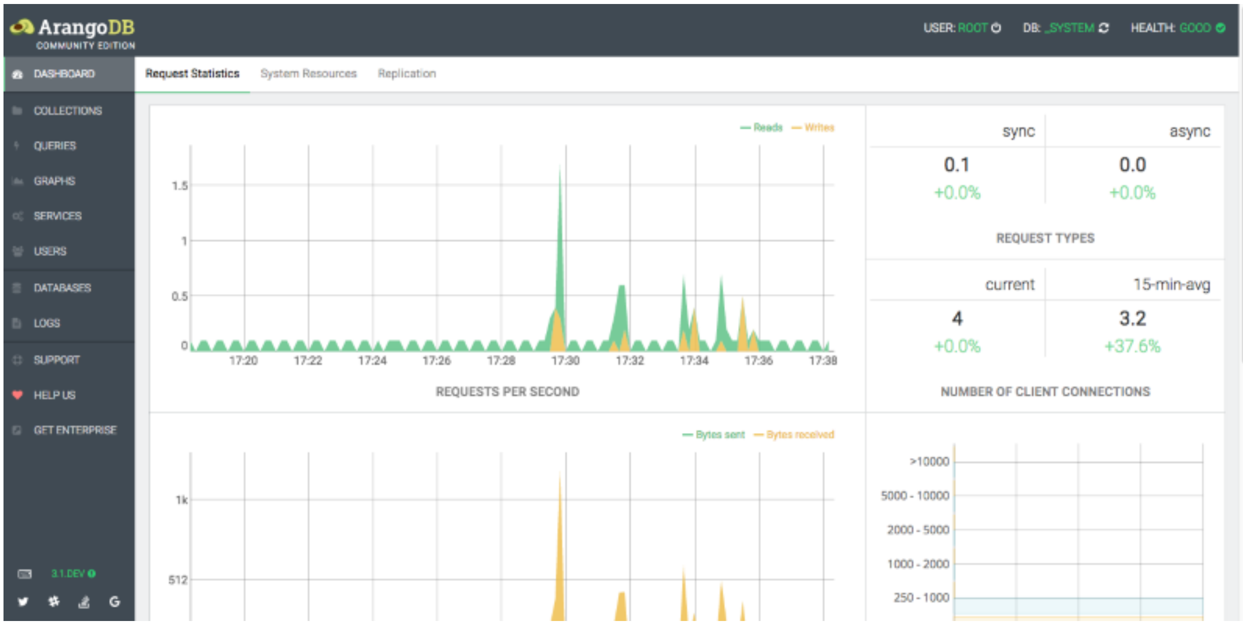

Figure 1. Example of Arango web-interface. Statistics.

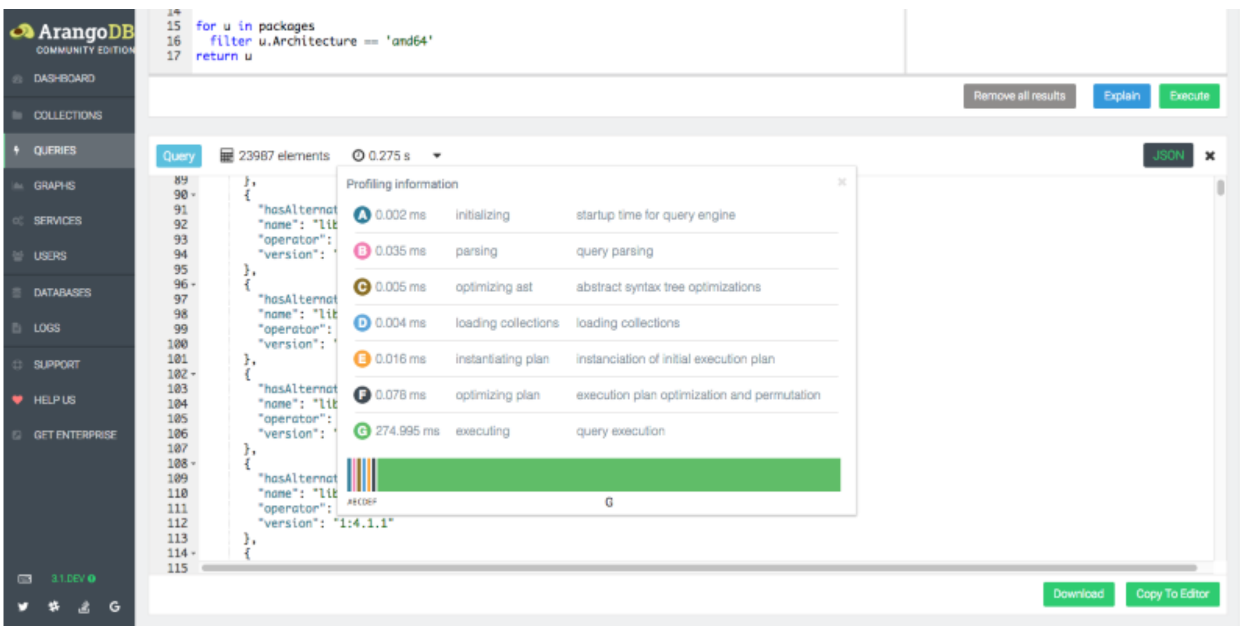

Figure 2. Example of query view in web-interface. Source: www.arangodb.com

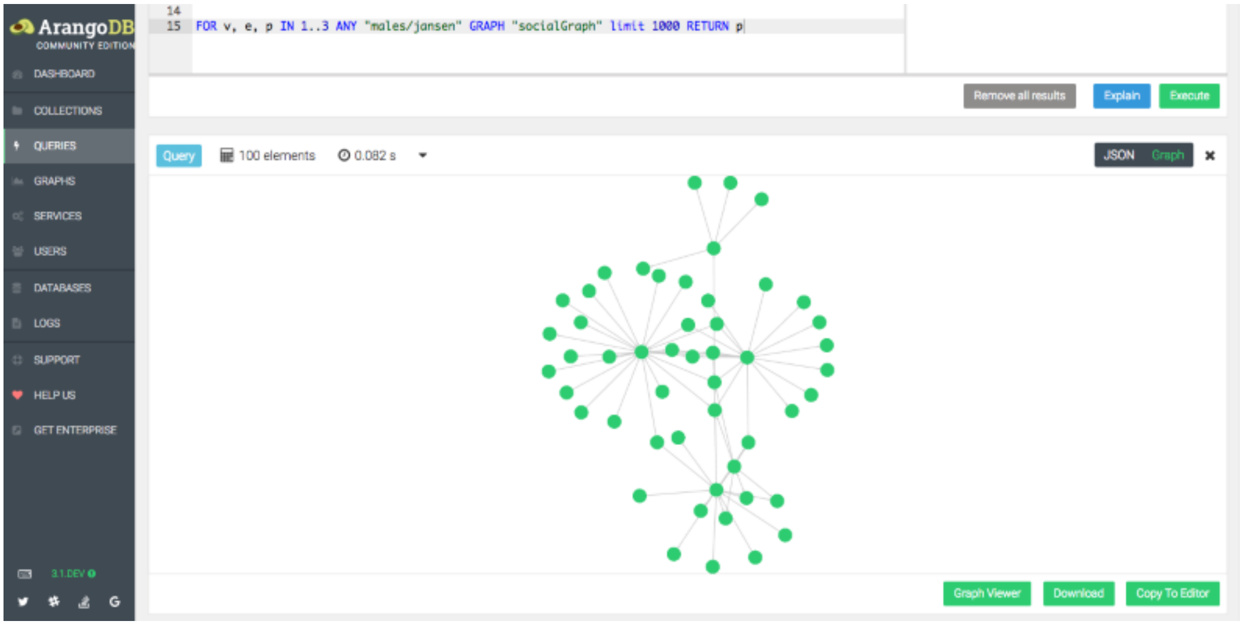

Figure 3. Example of graph view in web-interface. Source: www.arangodb.com

Benefits with same DB

There are lots of different graph databases which provide almost the same range of functions. ArangoDB can be compared with Neo4j and OrientDB but it offers some supplemental opportunities (according to the research and information, provided by DB developers, vschart.com and db-engines.com websites).

The most valuable advantages of ArangoDB are [1]:

1. It is multi-model DBMS when Neo4j is single-model graph database. If the project requires a different storage type, it is necessary to use another database to store it. At the same time, ArangoDB provides graph, document and key/value models.
2. ArangoDB provides the best performance in interactions with data from different data model queries.
3. ArangoDB offers impressive scalability opportunities.
4. In comparison with Neo4j, all the transactions can be encrypted by TSL or SSL. Moreover, in consist role-based control (via Foxx framework) and auditing in Enterprise version.
5. it compatible with larger number of NoSQL query languages than OrientDB, so it provides better opportunities for data interactions.

DB architecture

A cluster of ArangoDB consist of a number of database instances which can play 4 different roles: agents, coordinators, primary and secondary servers.

Agents can form the Agency in the cluster, which is the main part of the cluster configuration. It provides synchronization services for the whole cluster.

Coordinators are working with queries and Foxx services.

Primary servers are the places where data is hosted.

Secondary servers are asynchronous replicas of primary ones. There are one or more secondary servers for each primary. They are suitable for backup actions.

Storage engines

ArangoDB provides two different storage engines: traditional memory-mapped files and RocksDB based engine. The selection of the engine depends on the real case.

The traditional memory-mapped files engine has significantly longer startup time but it makes impressive performance with in-memory data queries. So, this engine is suited for datasets which can be represented in the main memory.

The new “rocksdb” engine was developed for large datasets which can not be kept in the main memory. In comparison with the traditional approach, this engine saves all the indexes on the disk that makes very fast startup.

Unfortunately, it is not possible to combine different engines in the one installation of the database. So, it must be selected for the whole server or cluster at the time of installation.

Hardware/Software requirements

ArangoDB runs on Linux, OS X and Microsoft Windows both 32bit (according to the architecture limits, it can use no more than 2-3GB of RAM) and 64bit systems.

There are no critical requirements for this DBMS. For example, according to the official manual, it is possible to run it on Raspberry Pi.

Furthemore, ArangoDB can be installed as a Docker container, in DC/OS, on Windows Azure and Amazon cloud services.

Installation

ArangoDB can be installed in different operation systems. So, this process is various and depends on the environment.

Mac OS X

There are two ways to install ArangoDB in OS X:

1. Graphical application. In this case the .app container should be downloaded and moved to the Application folder.

2. Installation through Homebrew. It is possible to install the latest version of ArangoDB by package manager Homebrew with the command: “`brew install arangodb`”. Then it can be started from the default installation folder `/USR/LOCAL/OPT/ARANGODB/SBIN/` by using command “arangod”. Consequently, we need to initialize the database by executing application “arango-secure-installation” and set a root password.

Linux

There are few different precompiled packages for Red hat, Fedora, Debian, Ubuntu and Suse Linux distributives. Normally, it is necessary to download the package for the necessary Linux version.

Another way of installation – using of package managers. The installation code for apt-get, yum and zypper package managers is presented below.

Apt-get

```
curl -O
https://www.arangodb.com/9c169fe900ff79790395784287bfa82f0dc0059375a34
a2881b9b745c8efd42e/arangodb32/Debian_9.0/Release.key
sudo apt-key add - < Release.key
ECHO 'DEB
HTTPS://WWW.ARANGODB.COM/9C169FE900FF79790395784287BFA82F0DC0059375A34A2881B
9B745C8EFD42E/ARANGODB32/DEBIAN_9.0/ /' | SUDO TEE
/ETC/APT/SOURCES.LIST.D/ARANGODB.LIST
SUDO APT-GET INSTALL APT-TRANSPORT-HTTPS
SUDO APT-GET UPDATE
SUDO APT-GET INSTALL ARANGODB3E=3.2.8
```

YUM

```
cd /etc/yum.repos.d/
curl -O
https://www.arangodb.com/9c169fe900ff79790395784287bfa82f0dc0059375a34
a2881b9b745c8efd42e/arangodb32/CentOS_7/arangodb.repo
yum -y install arangodb3e-3.2.8
```

Zypper

```
zypper --no-gpg-checks --gpg-auto-import-keys addrepo
https://www.arangodb.com/9c169fe900ff79790395784287bfa82f0dc0059375a34
a2881b9b745c8efd42e/arangodb32/openSUSE_13.2/arangodb.repo

zypper --no-gpg-checks --gpg-auto-import-keys refresh
zypper -n install arangodb3e=3.2.8
```

Compiling

Furthermore, if the precompiled package is not available, it is possible to compile and build ArangoDB from raw source. For example, for using it on RaspberryPi. ArangoDB was tested with GNU C/C++ and clang/clang++ compilers with C++11-enabled argument. This method is not very popular for normal using, so all the necessary information about compiling can be found on the ArangoDB documentation website: https://docs.arangodb.com/.

Windows

For Microsoft Windows the Windows Installer package is available. At the time of installation user can select the installation path, make single- or multiuser installation, keep a backup, create a desktop icon and other parameters which the Installation Wizard offers to the user.

Pricing schema

ArangoDB is licensed under Apache 2.0 public license. So, from the end user's point of view (excluding development, distributing, etc.), it means that ArangoDB is fully free for non-commercial and commercial use.

Moreover, there are 2 commercial subscriptions: Basic and Enterprise.

Features	Community	Basic	Enterprise
Community Edition Features	✓	✓	✓
SatelliteCollections			✓
SmartGraphs			✓
Encryption at Rest			✓
Enhanced User Management with LDAP			✓
Training			
Free, online education	✓	✓	✓
Private, on-demand training			✓
Support			
SLA	none	9×5	24×7
Response Time			
critical issues	no guarantee	12 hours*	2 hours
level 2 issues	no guarantee	16 hours*	5 hours
level 3 issues	no guarantee	40 hours*	16 hours
Number of issues		10 per month	unlimited
Support contacts	google-group only	1 email, web	4 email, web, phone
Technical alerts		✓	✓
Hotfixes	general release-cycle	general release-cycle	✓
critical issues	no guarantee	12 hours*	2 hours
level 2 issues	no guarantee	16 hours*	5 hours
License			
Type	Apache V2	Apache V2	Commercial

In order to estimate the possible expenditures for commercial subscription to the ArangoDB, the Sales department of ArangoDB GmbH was contacted. According to the ArangoDB GmbH politics, price model is variable for different projects and the average figures can not be provided.

Interactions

ArangoDB uses AQL language as a main query tool. The query can be executed from Command Line and Web UI. Furthermore, there is an impressive number of drivers for different languages:

- NodeJS;
- PHP;
- Java;
- JavaScript;
- .NET;
- Go;
- Python;
- Scala;
- Railo;
- D;
- Dart;
- Vert-X;
- Gremlin;
- Ruby on Rails;
- Clojure;
- Elixir;
- C++.

All the native drivers are presented on the project's GitHub https://github.com/arangodb/ with all the necessary documentation.

Benchmark

When it comes to the performance of the database, we can use the special benchmark for different NoSQL databases (https://github.com/weinberger/nosql-tests) which is based on NodeJS. The test collection consists 100 000 elements to read, write, find the neighbor, make aggregations and find the shortest way. We can simply install it on the UNIX based operation system by simple steps:

```
git clone https://github.com/weinberger/nosql-tests.git
npm install .
npm run data
```

To start all available tests, we need to start the server with additional parameters:

```
./bin/arangod  /mnt/data/arangodb/data-2.7 --server.threads 16 --wal.sync-interval
1000  --config etc/relative/arangod.conf --javascript.v8-contexts 17
```

As soon as ArangoDB was updated to version 3, scheduler.threads parameter is now obsolete and hava to be excluded.
We have to type the following command to start all the possible tests.

```
node benchmark arangodb -a 127.0.0.0 -t all
```

It is recommended to start the server
where 127.0.0.1 – IP address of the installed and run ArangoDB.

Unfortunately, the available benchmark was developed 2 years ago for ArangoDB2, when the current version is 3. So, it was necessary to fix it but it unfortunately it still not possible to test the performance

According to the executed test, we have received the results for the whole collection of 100.000 elements. The results of this test in comparison with the results from ArangoDB website are presented below:

Test	Single Read	Single Write	Aggregation	Shortest path	Neighbors (1+2 deg)	Neighbors (with profile data)
Result (sec.)	12.672	20.194	-	-	-	-
Vendor's results	16.962	20.530	1.250	0.061	0.464	4.327

As a result, we can see that at least single read operation became quicker in new version of ArangoDB. It is clear that the results (which are available) are almost the same that the DB vendor provides. So, we can use their results for further comparison, which are presented in the table below.

	single read	single write	single write sync	aggregation (ad-hoc query)	shortest path	neighbors* distinct 1st+2nd deg	neighbors with profile data	memory usage
ArangoDB	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %
2.7.0 RC2	16.962 sec.	20.530 sec.	91.816 sec.	1.250 sec.	0.061 sec.	0.464 sec.	4.327 sec.	13.142 GB
MongoDB	86.76 %	143.11 %	352.83 %	246.08 %		242.26 %	146.32 %	49.20 %
3.0.6 - WiredTiger	14.716	29.380	323.953	3.077		1.124	6.331	6.466 GB
Neo4j	522.31 %		173.02 %	225.37 %	694.41 %	511.43 %	152.59 %	68.47 %
2.3 M3	88.593		158.860	2.818	0.422	2.372	6.602	8.998 GB
OrientDB	168.55 %	116.96 %		1860.12 %	2188.82 %	1119.4 %	905.72 %	102.06 %
2.2 alpha	28.590	24.011		23.259	1.331	5.192	39.187	13.413 GB
PostgreSQL (json)	111.07 %	106.92 %	75.12 %	1401.22 %		461.91 %	66.21 %	85.21 %
9.4.4 - json	18.840	21.951	68.972	17.521		2.142	2.865	11.199 GB
PostgreSQL (tab.)	267.93 %	153.27 %	76.87 %	48.77 %		425.08 %	43.34 %	77.40 %
9.4.4 - tabular	45.445	31.465	70.581	0.61		1.972	1.875	10.172 GB

*) distance 1 and the distance 2 neighbors, each of them once.

Weinberger 2015-10-13 (r207)

Such results show us that ArangoDB has better performance than the vast majority of other Graph databases.

Use case

Problem

Nowadays problem of the load of public transport is becoming more and more important. Governments are interested to move all the people from private to public transport to improve road situation in cities, at the same time people are interested in maximal convenience of the trip: distance from the real destination from the closest bus/tram-stop, non-overcrowded vehicles and optimal time of the trip. To resolve the first problems, we have to increase number of vehicles but two other we can resolve by data research. Moreover, it is clear that the vast majority of current route-planners consider only nominal trip-time but in real life road traffic can significantly effect on the trip time. So, the quickest route can become the longest because of such problems.

As a result, we have a colossal number of different objects with lots of different 1:1 or 1:n relations. We can store it in habitual relational database which lots of different bridges. So, to build the route we need to consider lots of connections between them by using lots of join operations. It is clear that building of the route with some changes can become really difficult and hard to compute. Fortunately, we can use graph database to simplify this problem and make it more optimized because, it Is clear, that normal transport network is a graph. So, we can store information about our object (the network) in its native form – array of vertices and edges with some attributes.

Let’s look at the maps of Vienna’s tram (Straßenbahn) and metro (U-Bahn) maps (). We can see that both images are the graphs. So, we can simply store it in the ArangoDB graph. As a result, we can receive a united graph of tram and metro stations with information about possible traffic problems for trams (sometimes, tram rails are separated from the road network and we do not have to recognize road traffic for such path segments, it is mentioned on the image below).

Conclusively, our task is to store our data and build the quickest way by path finding and recognizing of transport situation on the road where it is applicable with the best performance by using ArangoDB.

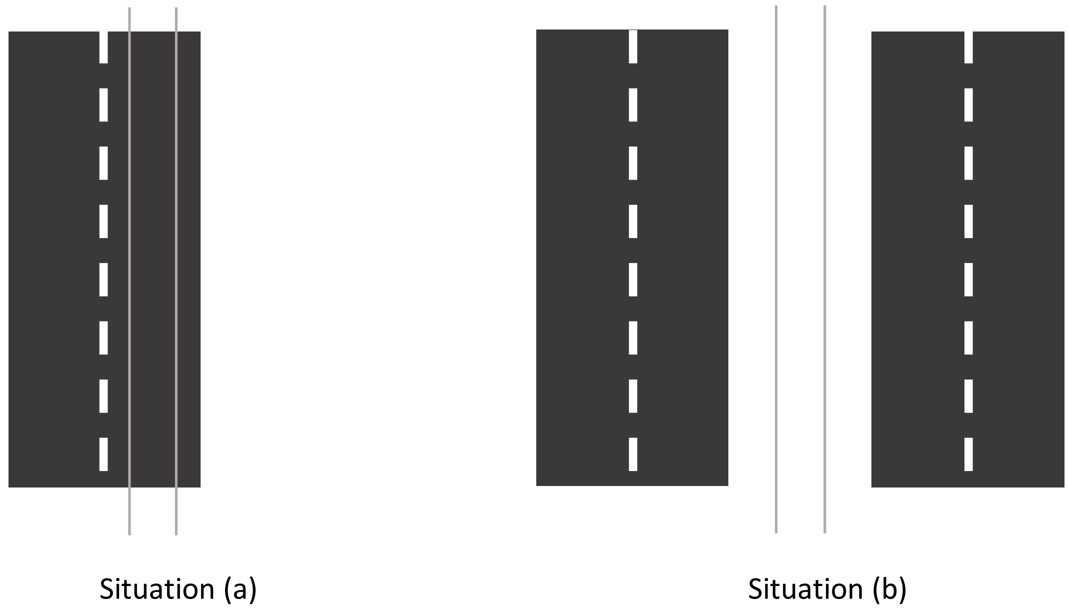

Figure 4. Differents situations with tram rails

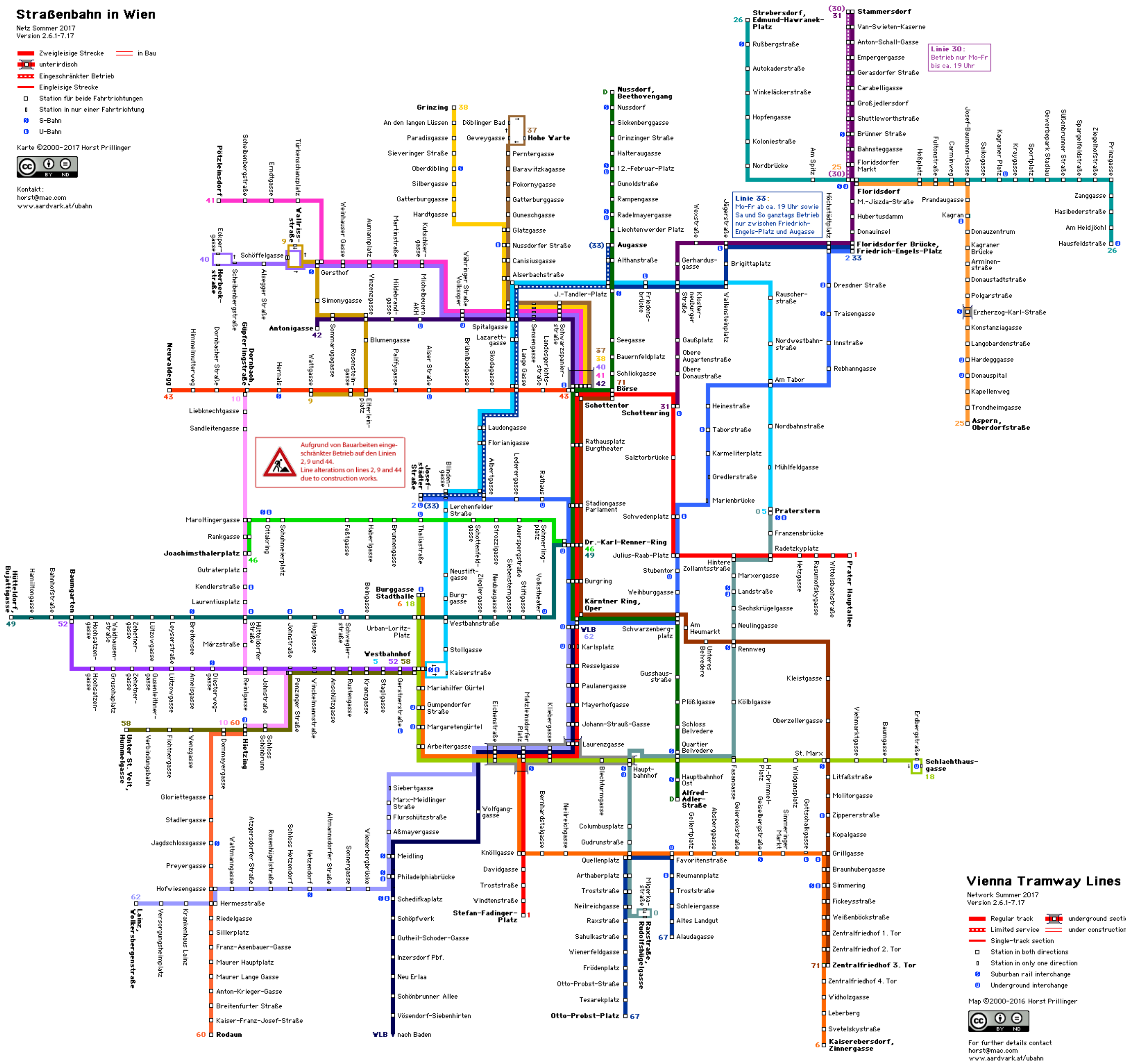

Figure 5. Scheme of Vienna's tram. Information from http://www.stadt-wien.at

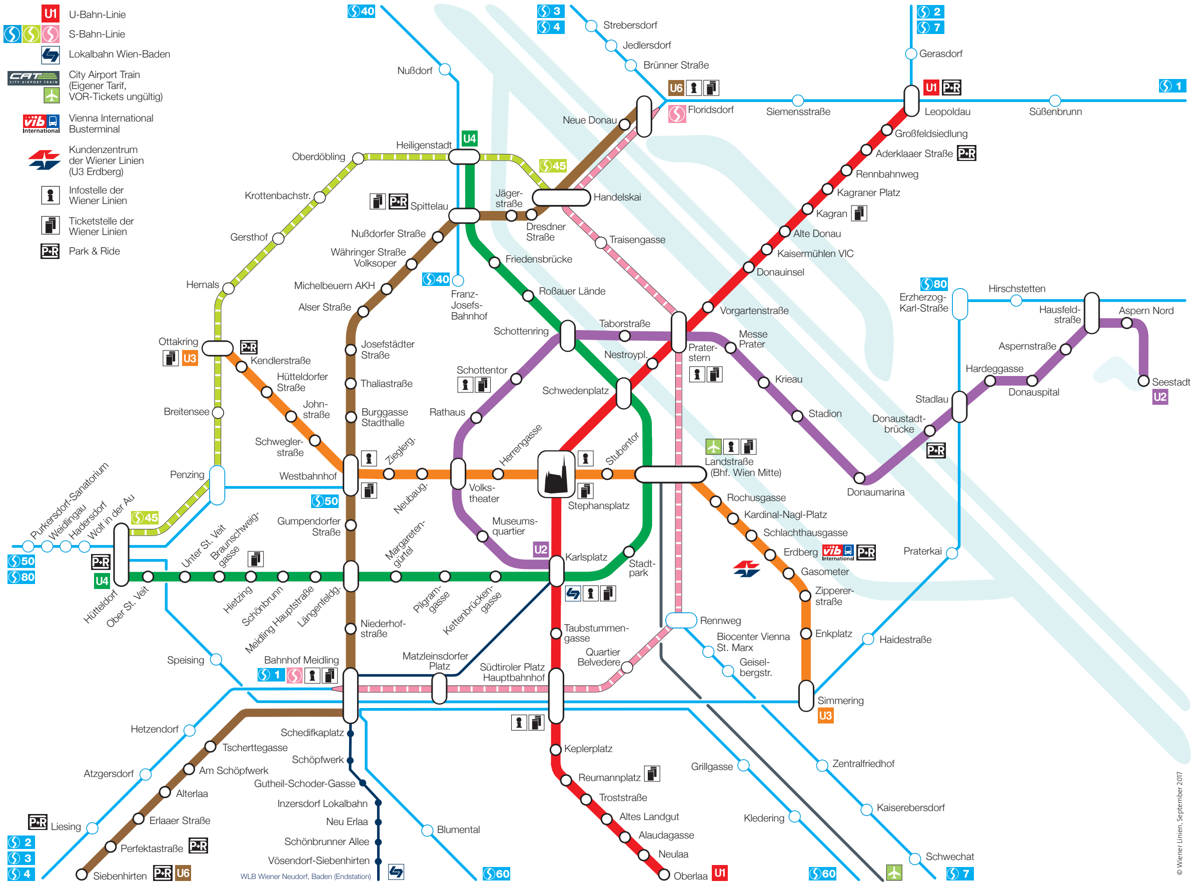

Figure 6. Map of Vienna's metro. Information from http://www.stadt-wien.at

Dataset

The problem touches public transport in Vienna, so we will use information from the web-portal "Austrian Open Data" https://www.data.gv.at/. Unfortunately, all the necessary data sets are represented in geographically appropriate format and we have to use few datasets to build the necessary one, which consists stops and connections between them.

We also need to store some additional information which can't be represented as a graph (or with lots of difficulties).

We will use the following sets:

- U-Bahnnetz Bestand Wien
- Öffentliches Verkehrsnetz Haltestellen Wien

All the data are presented in WFS, so it is necessary to make a preprocessing procedure to make all the necessary vertexes and edges of the final graph. For this reason, a small converter, based on Java, was developed. To import data to ArangoDB, one can use json or csv format of data. It is possible to import data by three ways:

- Web-interface, which is available on the local webpage http://127.0.0.1:8529/. We have to login to this interface and select a necessary database. To import the dataset it is necessary to create a new collection of data, open it and import the file (by clicking Import->Select file->Import JSON). Unfortunately, it is impossible to import CSV file by this way;
- By command-line request through arangosh app. To import data we need to access it, enter our password. To import file we can use the following line:

```
arangoimp --file data.json --collection commits --create-collection
true
```

To import csv file one should add -- type csv parameter. In case of necessity we can select the database, host and other parameters. One can read the full description of all possible options in the integrated man support article by typing “arangoimp –help”.

- The third opportunity is direct interaction between database and application through the appropriate driver.

FID	OBJECTID	SHAPE	HTXT	HTXTK	HLINIEN	DIVA_ID	SE_ANNO_	AD_DATA
OEFFHALTES	1	POINT (16.53594056321	Invalidensiedlung	Invalidensiedlung	89A	579		
OEFFHALTES	2	POINT (16.53629490645	Speierlinggasse	Speierlinggasse	89A	1282		
OEFFHALTES	3	POINT (16.53615264914	Speierlinggasse	Speierlinggasse	89A	1282		
OEFFHALTES	4	POINT (16.53433994625	Thujagasse	Thujagasse	89A	1380		
OEFFHALTES	5	POINT (16.53395272159	Thujagasse	Thujagasse	89A	1380		
OEFFHALTES	6	POINT (16.53258739079	Neuessling	Neuessling	89A	924		
OEFFHALTES	7	POINT (16.53160503501	Algenweg	Algenweg	89A	22		
OEFFHALTES	8	POINT (16.53187793888	Huflattichweg	Huflattichweg	89A	555		
OEFFHALTES	9	POINT (16.53108796111	Huflattichweg	Huflattichweg	89A	555		
OEFFHALTES	10	POINT (16.53039875815	Daphneweg	Daphneweg	89A	223		
OEFFHALTES	11	POINT (16.53119109560	Daphneweg	Daphneweg	89A	223		
OEFFHALTES	12	POINT (16.53057887664	Asparagusweg	Asparagusweg	89A	1093		
OEFFHALTES	13	POINT (16.52489407975	RÃ¶bbelingasse	RÃ¶bbelingasse	89A	9901		
OEFFHALTES	14	POINT (16.52164154186	Heringgasse	Heringgasse	89A	503		
OEFFHALTES	15	POINT (16.51992059293	Seiseneggergasse	Seiseneggergasse	89A	1221		
OEFFHALTES	16	POINT (16.50495799382	Aspern Nord	Aspern Nord	89A	910		
OEFFHALTES	17	POINT (16.48763988550	Stallarngasse	Stallarngasse	25A	1307		
OEFFHALTES	18	POINT (16.48425541467	Bhf. SÃ¼ÃŸenbrunn	Bhf. SÃ¼ÃŸenbrunn	25A	1357		
OEFFHALTES	19	POINT (16.48302511395	Gerasdorf	Gerasdorf	25A			
OEFFHALTES	20	POINT (16.46846347796	Gerasdorf, Postamt	Gerasdorf, Postamt	25A			
OEFFHALTES	21	POINT (16.47151051569	Gerasdorf, Hauptschule	Gerasdorf, Hauptschul	25A			
OEFFHALTES	22	POINT (16.47840740202	Josef-BÃ¶ck-Gasse	Josef-BÃ¶ck-Gasse	25A			
OEFFHALTES	23	POINT (16.49216577437	BettelheimstraÃŸe	BettelheimstraÃŸe	25A	133		
OEFFHALTES	24	POINT (16.49276291199	BettelheimstraÃŸe	BettelheimstraÃŸe	25A	133		
OEFFHALTES	25	POINT (16.49704975351	Weingartenallee	Weingartenallee	25A	1688		
OEFFHALTES	26	POINT (16.50027944530	Sportpark SÃ¼ÃŸenbrunn	Sportpark SÃ¼ÃŸenbru	25A	1689		
OEFFHALTES	27	POINT (16.49567928134	SÃ¼ÃŸenbrunner Platz	SÃ¼ÃŸenbrunner Platz	25A	1359		
OEFFHALTES	28	POINT (16.47624462103	Wagramer StraÃŸe/Ostba	Wagramer StraÃŸe/Os	25A	979		
OEFFHALTES	29	POINT (16.48182545273	Campingplatzweg	Campingplatzweg	25A	201		
OEFFHALTES	30	POINT (16.48792314982	Badeteich SÃ¼ÃŸenbrunn	Badeteich SÃ¼ÃŸenbru	25A	105		
OEFFHALTES	31	POINT (16.46487631207	PercostraÃŸe/Wagramer S	PercostraÃŸe/Wagram	25A	1254		
OEFFHALTES	32	[illegible]	[illegible]	[illegible]	25A	[illegible]		

Figure 7. Example of the raw dataset

_key	OBJECTID	location	DIVA_ID
Stephansplat	347	16.37172291	1320
Rennbahnwe	348	16.44966089	1860
Leopoldau	350	16.45236169	769
Kaisermühlei	353	16.41648782	641
Keplerplatz	355	16.37617475	666
Großfeldsied	360	16.44794106	1859
Vorgartenstr	361	16.40106760	1433
Aderklaaer S	365	16.45163882	1858
Kagran	382	16.43298497	627
Praterstern	385	16.39331893	1040
Oberlaa	386	16.40007286	731
Neulaa	387	16.385867489281246	48.14575040597
Alaudagasse	388	16.38236150	14
Altes Landgu	389	16.38325470	33
Troststraße	390	16.38016919	1391
Schwedenpla	391	16.37754212	1198
Karlsplatz	394	16.36919233	657
Reumannpla	404	16.37788286	1095
Taubstumme	411	16.37033221	1366
Alte Donau	414	16.42485680	31
Hauptbahnh	422	16.37360510	1349
Nestroyplatz	424	16.38559329	916
Donauinsel	426	16.41141723	234
Kagraner Pla	439	16.44338374	633

Figure 8. Example of the final dataset for metro line U1.

_key	_from	_to
350360	U1/Leopolda	U1/Großfeldsiedlung
360365	U1/Großfeld	U1/Aderklaaer Straße
365348	U1/Aderklaa	U1/Rennbahnweg
348439	U1/Rennbah	U1/Kagraner Platz
439382	U1/Kagraner	U1/Kagran
382414	U1/Kagran	U1/Alte Donau
414353	U1/Alte Don	U1/Kaisermühlen-VIC
353426	U1/Kaisermü	U1/Donauinsel
426361	U1/Donauins	U1/Vorgartenstraße
361385	U1/Vorgarte	U1/Praterstern
385424	U1/Praterste	U1/Nestroyplatz
424391	U1/Nestroyp	U1/Schwedenplatz
391347	U1/Schwede	U1/Stephansplatz
347394	U1/Stephans	U1/Karlsplatz
394411	U1/Karlsplat	U1/Taubstummengasse
411422	U1/Taubstun	U1/Hauptbahnhof
422355	U1/Hauptbal	U1/Keplerplatz
355404	U1/Keplerpla	U1/Reumannplatz
404390	U1/Reumanr	U1/Troststraße
390389	U1/Troststra	U1/Altes Landgut
389388	U1/Altes Lan	U1/Alaudagasse
388387	U1/Alaudaga	U1/Neulaa
387386	U1/Neulaa	U1/Oberlaa
386	U1/Oberlaa	

Figure 9. Example of edges dataset.

According to the ArangoDB architecture, all the vertices should consist "_key" attribute as well as edges "_from", "_to" and "_key" attributes. Sometimes there were some problems with data import, so it was necessary to convert data in csv format to json by webtool http://www.csvjson.com/csv2json.

Furthermore, some problems with charset also were met because original language of the dataset is German. So, it was necessary to fix some damaged data, like missing special symbols like ä, ö, ü, ë, ß. Moreover, key attributes must not contain such symbols and spaces.

Prototype

To make the prototype we will use Java 1.8 and driver for ArangoDB to have a connection between user interface and the application. After exporting of the dataset the following graph was received:

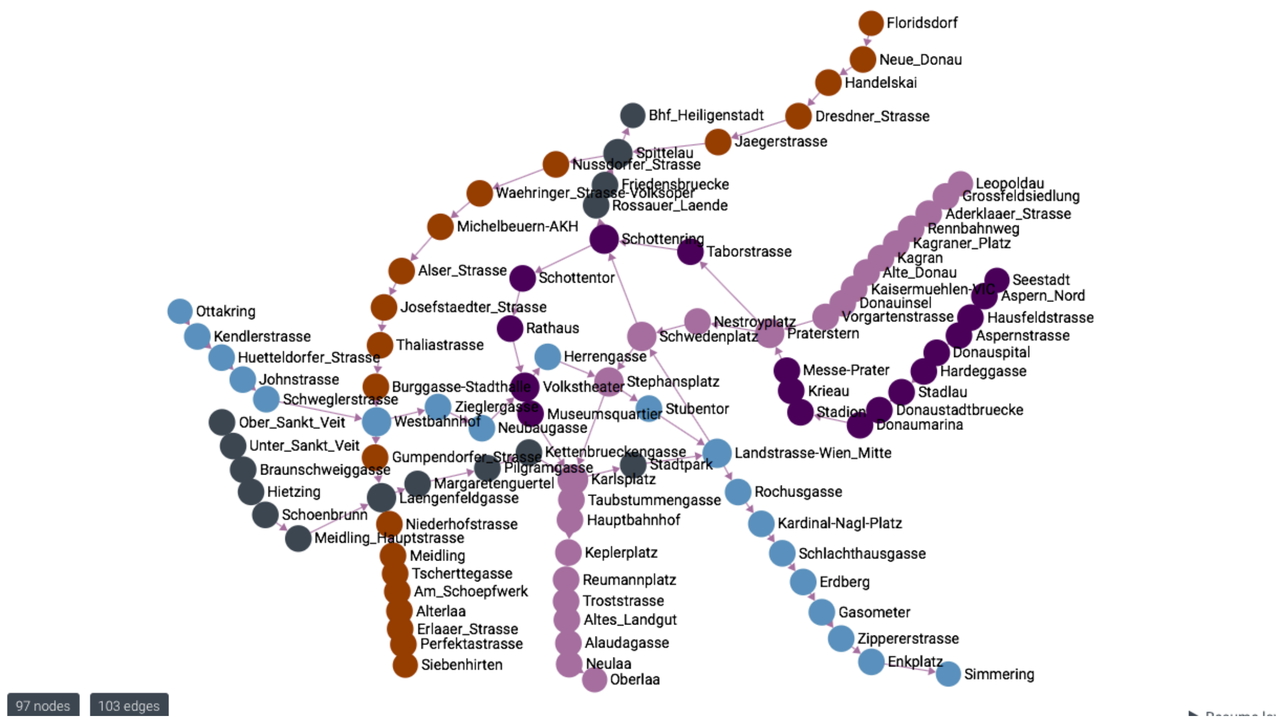

Figure 10. Graph of U-Bahn network in Wien

Current prototype shows the opportunity to make a graph search through the shortest path algorithm of the database. The AQL query is:

```
FOR h1, h2 IN OUTBOUND SHORTEST_PATH '"+st+"' TO '"+en+"' GRAPH
'U' RETURN [h1._key,h2._key]
```

Where st and en – variables of key for start and end stations. It is clear that it is only a prototype. In future, there is an opportunity to extend functionality: add graph visualization, additional parameters to compute and improve the GUI to make it more appropriate for mobile devices and user-experience.

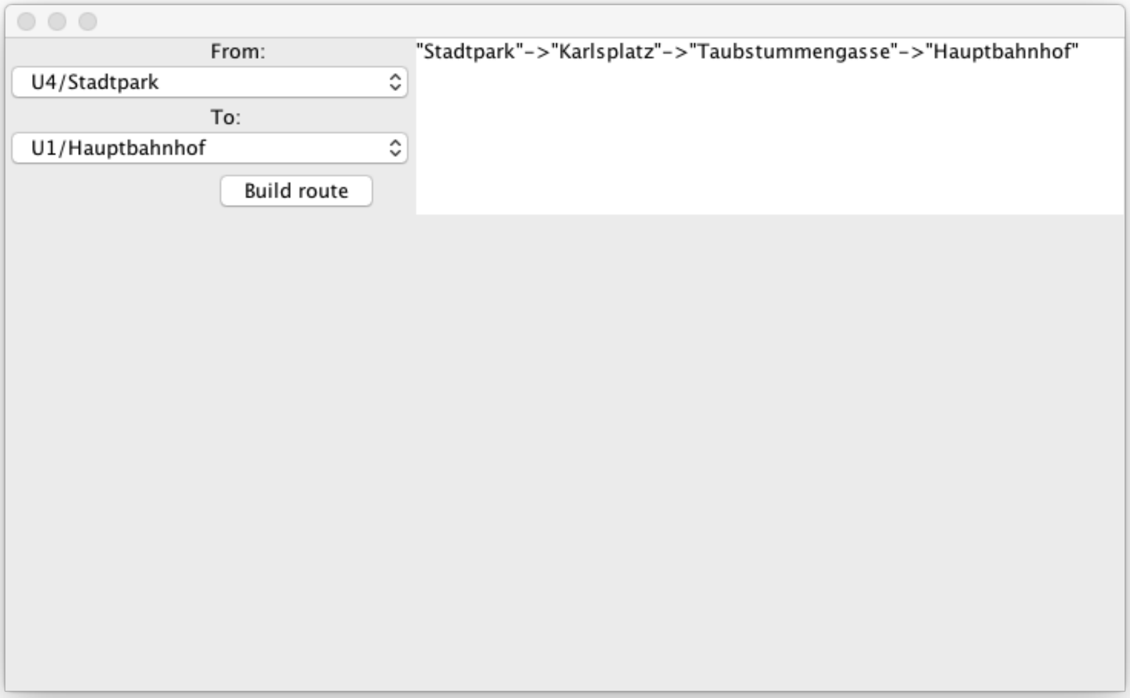

Figure 11. Prototype screenshot

Other use cases

The proposed idea is not one of the possible projects which could be implemented by using ArangoDB. As it was mentioned before, ArangoDB is a native graph database, so all the possible project which could be executed by using such technology, can be developed on its base. For example:

- Flights analysis;
- Social-network analysis (e.g. for internal networks like in KPMG and McKinsey companies);

So, according to the information from ArangoDB GmbH (https://www.arangodb.com/why-arangodb/case-studies/thomson-reuters-fast-secure-single-view-everything-arangodb/), the key users of the ArangoDB are:

- Thomson Reuters – analytical company which provides intelligence, technology and human analyses and expertise;
- FlightStats – company which deals with flight information: historical analysis, current monitoring and predictive services;
- and lots of other companies and academic projects by Oxford university, KIT.

Conclusion

Finally, graph database is a really important type of data storage which provides an opportunity to store and represent connected data, by making its storing more similar to real world. For example, relations between people, transport, financial operations, decision making systems, etc.

ArangoDB is one of the databases which provides an opportunity to store data in such format. The main advantage of this database in comparison with rivals – it is a native graph storage. Three types of store engines make the system very flexible for different situations. Furthermore, it is an excellent technology to deploy a graph-depended system like transport routing, social network and other net-oriented cases with impressive performance and low system requirements. Firstly, it was producted in 2011, so it has nothing in common with some outdated technologies, which could be reason for any compatibility and performance problems with more aged solutions. Fortunately, the licensing system provides the opportunity to use a full version of the database for free without any limitations.

The performance of ArangoDB looks really impressive when it deals with basic operations such as single read and write, when search of the shortest route sometimes takes a little bit more time than the same figure of its rivals (Postgres, Neo4j and MongoDB). But the most important advantage of this technology is opportunity to store data at the same time in graph and in document store. It is really interesting feature, because it significantly reduces time of the development for projects which store data in different storages.

Moreover, ArangoDB have a powerful clustering system that makes it really impressive in case of use as a distributed storage with different roles of participants.

After the research, some undocumented features and problems were discovered (for example, driver connection, performance, etc.), which were resolved by contacting the support or looking for the same problem on Stackoverflow website.

As a part of the research, the prototype of routing application of public transport in Vienna was developed. It uses searching opportunity of the database to find the shortest way between the bus-/tram-stops/metro-stations. The developer of the system claimed that native language AQL is very useful in case of interacting with graph data. It was proofed in this research.

References

1. https://www.arangodb.com/
2. http://neo4j.com
3. https://github.com/weinberger/nosql-tests
4. http://db-engines.com
5. http://stackoverflow.com
6. https://github.com/jgraph/jgraphx
7. http://open.wien.gv.at
8. http://wienerlinien.at
9. http://www.csvjson.com/csv2json