SN Computer Science (2020) 1:31
https://doi.org/10.1007/s42979-019-0031-y

ORIGINAL RESEARCH

# Performance Tests of Smart City IoT Data Repositories for Universal Linear Infrastructure Data and Graph Databases

**Michał Gorawski**[1] · **Krzysztof Grochla**[1]

Received: 22 July 2019 / Accepted: 2 September 2019 / Published online: 13 October 2019


## Abstract

The infrastructure managed by Smart City systems includes mainly linear structures, such as pipelines, electric power lines, railway lines, etc.. In this paper, we propose the transcription of linear infrastructure to graph representation and storing needed meta-data in a graph database, merged with the traditional relational database used for device management. Such an approach enables fast acquisition of data about infrastructure's properties and allows merging information about the structure of the managed infrastructure with information devices' properties and statistics. We develop a graph generator that generates virtual network basing on representative examples of linear infrastructure, incorporation of a data from existing metering systems with the generated structure, which automates the deployment and allows to evaluate the performance of Smart City monitoring and management system. We present results of performance tests that show and the creation time of graph database structures in Neo4j and the difference between the performance of the Microsoft SQL Server database and the Neo4j database in a few Smart City use cases. The results show that the use of the graph database to execute queries related to linear infrastructure allows decreasing the response time up to 9 times.

**Keywords** Smart City · Smart devices · Graph databases

## Introduction

The idea of Smart City became firmly rooted in the consciousness of the world population, and in the last years, many cities began to incorporate more and more smart technologies in their infrastructure [1, 2]. In the Smart City IoT architecture, which deals with data from sensor smart devices, such as gas, heat, or electric energy meters, one of the important information is a layout of an existing infrastructure in which smart devices are implemented. The lack of information about the actual infrastructure severely decreases the usefulness of Smart City systems and delivers only raw data about measurements from smart devices without any additional context.

Smart City IoT systems collect data from smart devices, e.g., the smart media meters, actuators, controllers, etc. In this paper, we focus on smart media meters that measure utility consumption and can be placed on the clients' devices such as heaters, water pipes, or gas pipes; moreover, such devices can also be placed on the infrastructure components that are outside of clients' apartment, but are crucial in utility delivery, such as ground pipes, manholes, tanks, etc. These devices can give the utility supplier crucial information about utility consumption but also about the state of the whole infrastructure and possible emergencies—given we have knowledge about the whole infrastructure and how it is connected. Most of the infrastructures serviced by Smart City IoT systems that we focus on are so-called man-made linear infrastructures in this paper referred to as LI. As the construction law defines, the LI is a construction object, whose characteristic parameter is its length, e.g., railways, waterworks, canal, gas pipeline, heat pipelines, pipeline, electric power lines, cable ducting or roads. A good representation of such infrastructure is very helpful in locating, e.g., leakage in these systems, especially leaks in gas and

This article is part of the topical collection "Modelling methods in Computer Systems, Networks and Bioinformatics" guest edited by Erol Gelenbe.

✉ Krzysztof Grochla
kgrochla@iitis.pl

Michał Gorawski
mgorawski@iitis.pl

[1] Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Batycka 5, 44-100 Gliwice, Poland

fuel lines are very dangerous and need to be quickly localized. As the inconsistencies can exist from nay different sources such as malfunctioning sensor or miscalculation of equipment [3], it is important to supply effective algorithms that identify real leaks without false positive alerts [4]. As all the above-mentioned infrastructures are defined as an LI, their characteristics can be easy translated into graphs. The usage of graphs and graph theory in representation of LI is not a new idea and it was used to represent the LI and detect faults in numerous publications for gas [5, 6], district heating [7, 8], water [9], and landscape planning [10] Quite interesting idea was presented by the authors of US patent [11] that gives an idea for generating A method and system to generate a network graph representation of a physically connected network.

The Smart City systems must merge existing data about smart meters state, readings, and parameters with information about the structure of the infrastructure, such as the topology of the network. The knowledge about the exact infrastructure and the connections between LI nodes gives many new possibilities of utilizing the system to monitor many parameters on a certain path between the media source and receiving clients. Knowledge of the whole infrastructure with online access to the state of all smart devices in every point of LI gives a good level of control and helps to pinpoint possible malfunctions of infrastructure components. An important problem in the development of a management or decision support system for linear infrastructure is the lack of data of measurements from representative systems. We use data from a measurement system that has large areas monitored using the smart devices, and collects a large amount of data about the smart devices measurements, properties, monitored equipment information, localization information, etc. However, the data lack information about how the localizations are connected and via which medium. To create a pilot proof of concept system, we created the LI infrastructure generator in the form of a multi-connection graph. The virtual infrastructure is then populated by real data from the Network Management System (NMS) database and can be used as a base for developing an application that potential clients require

The Smart City systems must merge existing data about smart meters state, readings, and parameters with information about the structure of the infrastructure, such as the topology of the network. The knowledge about the exact infrastructure and the connections between LI nodes gives many new possibilities of utilizing the system to monitor many parameters on a certain path between the media source and receiving clients. Knowledge of the whole infrastructure with online access to the state of all smart devices in every point of LI gives a good level of control and helps to pinpoint possible malfunctions of infrastructure components. An important problem in the development of a management or decision support system for linear infrastructure is the lack of data of measurements from representative systems. We use data from a measurement system that has large areas monitored using the smart devices, and collects a large amount of data about the smart devices measurements, properties, monitored equipment information, localization information, etc. However, the data lack information about how the localizations are connected and via which medium. To create a pilot proof of concept system, we created the LI infrastructure generator in the form of a multi-connection graph. The virtual infrastructure is then populated by real data from the Network Management System (NMS) database and can be used as a base for developing an application that potential clients require.

## The Linear Infrastructure Representation

All of the above-mentioned infrastructures can be naturally presented as a directed or undirected multigraph, where edges have their own identity. The main problem is how to represent the LI infrastructure in the graph model to include the various possible transportation methods that can exist in the Smart City infrastructure. Every one of possible architectures has their own characteristics, e.g., waterworks are generally connected with single pipe transporting fresh water (with sewage system considered as a separate architecture), in case of electric power, we have cables, and in case of a district heating, we deal with two separate pipes—one transporting hot water that powers the system (power) and second pipe with cool water that is returning to a heating plant (return). Our first idea was to model the real infrastructure, where connections will be represented using graph edges with certain properties. However, after a deeper analysis, we decided to create transport nodes to include the possibility of several devices that can be mounted on transporting objects (e.g., several sensors inside the pipe). The main idea of our graph structure can be described as a hierarchical graph structure divided into layers, and the full idea is presented in [12], where we focus on a district heating system. A similar solution was presented in [11]; however, our solution includes much more focus on the description of monitored objects (meters) and monitoring devices (devices).

The idea is to divide the schema into three layers:

1. *Infrastructure layer* this layer represents the real connections between physical objects (e.g., heating plant, client location), points (“crossroads”), and transport medium (transport nodes representing, e.g., pipes).
2. *Meter layer* this layer represents devices on which sensors are mounted (e.g., tanks, heaters, etc.).
3. *Device layer* this layer represents physical measuring platforms that can include one or several sensors.

Each Infrastructure nodes can have several devices, and each device can have several platforms. Such an approach is quite flexible to include different kinds of LI which clear connections and dependencies between LI nodes, devices, and measuring equipment.

## Metadata Repository

The graph representation of LI presented in [12] forces us to store graph data in some repository. There are several methods of storing the graph data in a relational database, such as the Microsoft SQL Server; however, this causes some problems. First of all, we need separate tables for nodes and edges, with a series of additional tables storing objects data. The example of such a simplified structure we created is presented in Fig. 4. The straightforward query about the path from one node to other is quite easy in this structure, however getting, for example, the shortest path requires implementing additional algorithms, and the response times will be longer than from, e.g., graph databases that are designed to store such data and have already implemented algorithms for optimizing graph search. We made research on available graph database systems and decided to utilize the graph database environment.

Graph databases are popular tools while dealing which personal and social data, quite commonly used in social media platforms, Human Resources, banking (fraud detection) and in many other scenarios. Graph databases are characterized by:

1. Based on graph theory—it consists of nodes, edges, and properties.
2. Properties can be ascribed to both nodes and edges.
3. The graph databases store the relations between records directly (no need to join multiple tables as in RDB).

Several popular solutions were considered : InfniteGraph, Sparksee (DEX), Neo4j, OrientDB, Titan (JanusGraph), TinkerPop, Ontotext (GraphDB), and ArangoDB. In the implementation of a Graph Generator/Import Wizard, we decided on Neo4j [13], mainly because of its well-established position on the market (according to [14] in 2018 and 2019 by far most popular solution), support for SPARQL [15], flexible, and easy to learn CYPHER language [16].

## Graph Generator

The graph generator application creates a list of nodes with different types (Source, Middle, Transport, Client, Device, Meter, and Sensor) the complexity of the generated graph can be changed using the configuration file in JSON format:

1. Maximal number of generated source nodes.
2. Maximal number of inputs for a node.
3. Maximal number of outputs for a node.
4. Number of generated nodes.
5. Maximal number of observed meters for a node.
6. Maximal numbers of DEVICES for METERS.
7. Maximal number of levels DEVICE.
8. Number of devices on every level.
9. probability of getting additional level for DEVICE (0–1).

The application was implemented in Java as a Maven project and works in sync with Microsoft SQL Server and Neo4j databases. The LI layer graph structure is generated based on the number of MIDDLE nodes (crossroads) and the connections are based on a chosen number of possible inputs and outputs for each node. The more inputs and outputs the more dense graph will become. In addition, the possible number of SOURCE and CLIENT nodes can be defined.

After the Infrastructure layer is set, for every node, meters and devices are added according to values defined in the config file. In addition, the additional levels of devices can be configured to be more than 1 level deep

### Integration with representative data from NMS database

To fuel the generated structure in which data, we get the real data about locations, devices, and meters from the NMS database and integrate them into a created graph structure. For the transport nodes, we generate data (parameters) according to the configuration file. The integration schema is presented in Fig. 1 and the created database is currently being used as a metadata source for the demo of ontology service application (Fig. 2).

### Import Wizard

As for now, we generate graph data, however, in the representative environment, the system will get the data about localisations and connections from external sources (e.g., a database or CSV file). The application is ready to receive such data, and all of the mechanisms of creating graph nodes and connections are implemented; however, to actually implement the loading process, we need to have the source data, similar to Extraction Transformation and Load (ETL) process in data warehouses, we need to implement it separately for a data source, because for each client, it will look differently, the data will be stored in different formats and repositories, so this part is actually to be fully implemented during pilot implementation of the system. The import of data is partially implemented with the NMS database; the

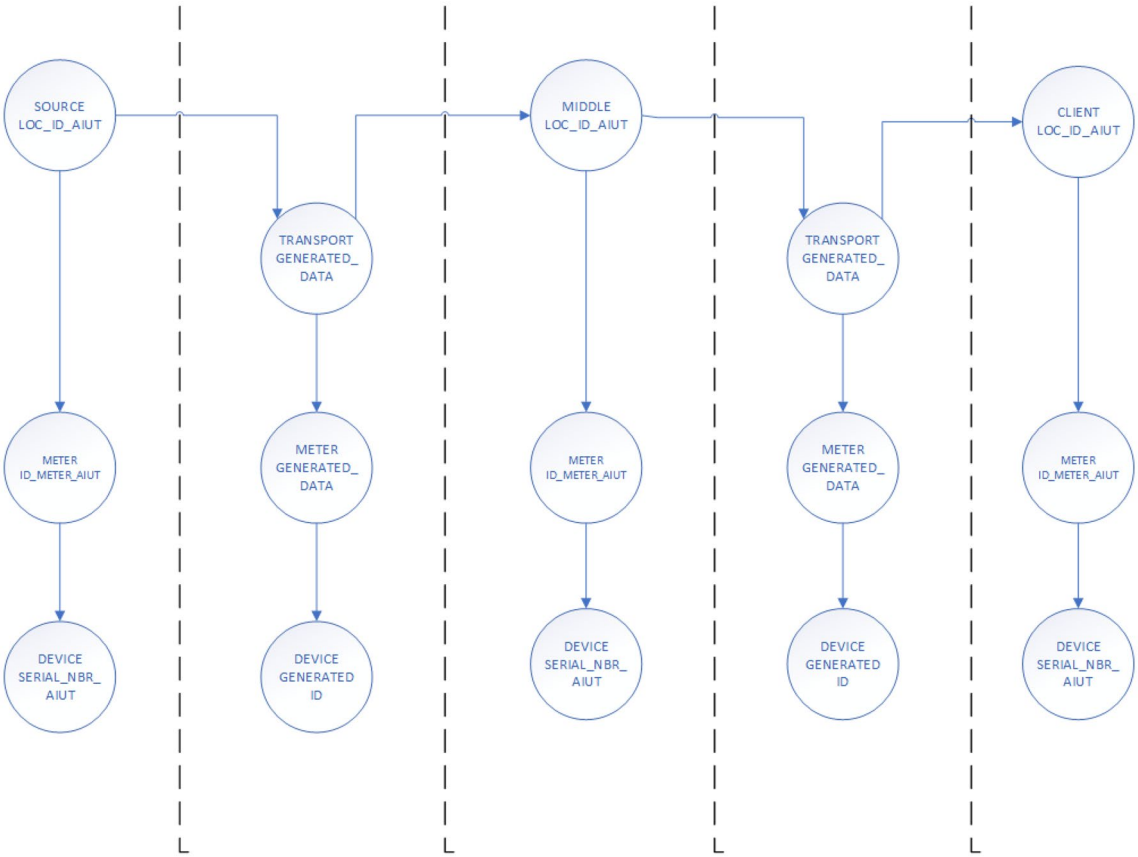


**Fig. 1** Integrating data from NMS into graph generator

application fully supports ETL of NMS data and already is used for generated infrastructure (Fig. 3).

## Graph Generator Tests

We performed tests of graph generator, where we were importing data from the NMS-to-Neo4j structure. The tests are thoroughly described in [12], however, the main conclusion was that along with the increase of the number of middle nodes, the building time grows exponentially, and however, the fact that it is a one-time process the building time around 4–5h is expected and acceptable.

The performance tests we present in this paper show the ability to get the path from one node to another. This is quite important in case of Smart City systems, where queries about the state of devices on a path from source to the client are quite important—checking the exact path of delivering gas, energy, or water is needed to, e.g., find alternative connection possibilities or checking measurement values while looking at possible leak situation. As a current installment of the NMS database is not able to process such queries, we introduced (as a comparison) Microsoft SQL Server database with schema presented on Fig. 4, and it stores the same data (generated + extracted from NMS) as a Neo4j structure, as presented in Fig. 3.

We made 2 series of tests with 1000 middle nodes and 3000 middle nodes, and we performed tests with 70 consecutive queries to the Microsoft SQL Server system and Neo4j system and compared the results. The Neo4j system was searching for the shortest path from source to a client device; in the Microsoft SQL Server system, we were looking for any path, not the shortest one, so the Microsoft SQL Server system had an easier task than Neo4j.

For 1000 nodes (Fig. 5) for Neo4j, average query response time was 13,3913 ms and the total time for all queries was 924. For Microsoft SQL Server, it was average 2199,4928 ms and the total time was 13,765 ms.

For 3000 nodes (Fig. 6) for Neo4j, average query response time was 28,3913ms and the total time for all queries was 1959. For Microsoft SQL Server, it was average 251,7681 ms and the total time was 17,372 ms.

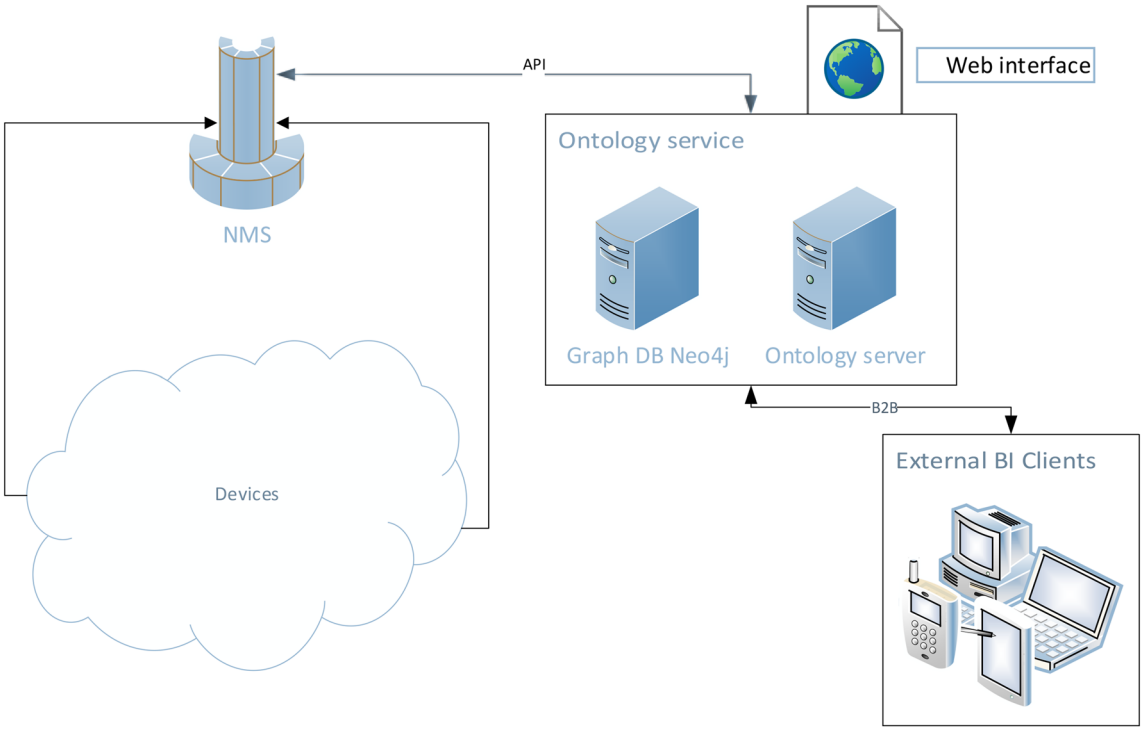


**Fig. 2** Ontology services schema

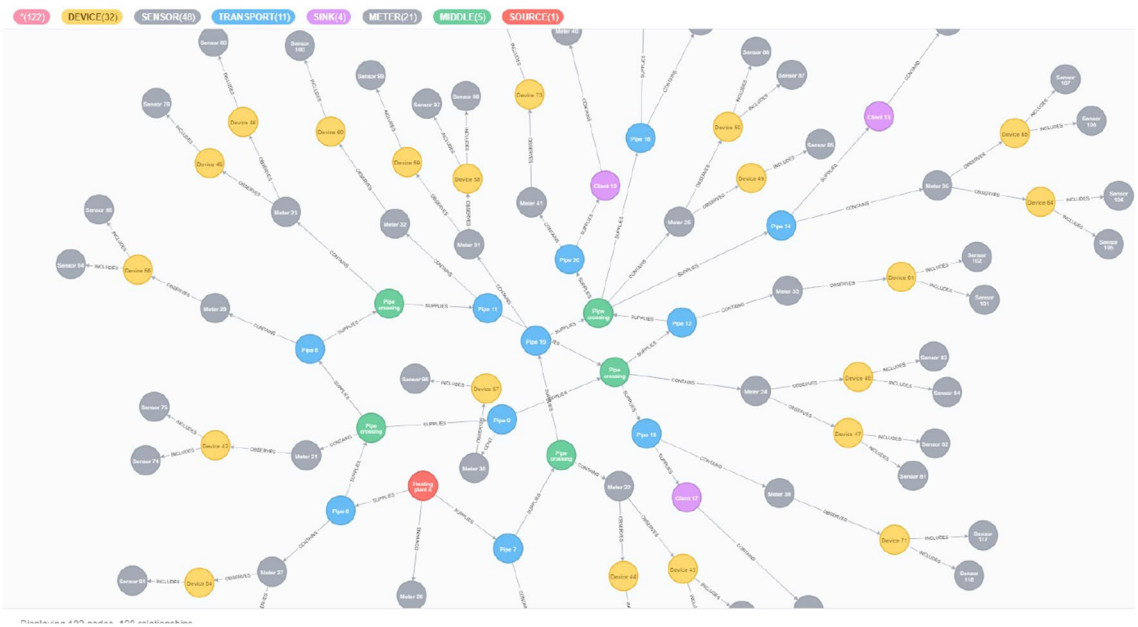


**Fig. 3** Example of populated Neo4j database

In both figures, we can see a considerable overhead on a first query, which is probably the Neo4j and Microsoft SQL Server overhead and it can be neglected, as it is not present in the following queries. The clear difference between response time that increases with the number of nodes and already implemented algorithms optimizing graph searches, proves that Neo4j will be a better choice for the Smart City systems.

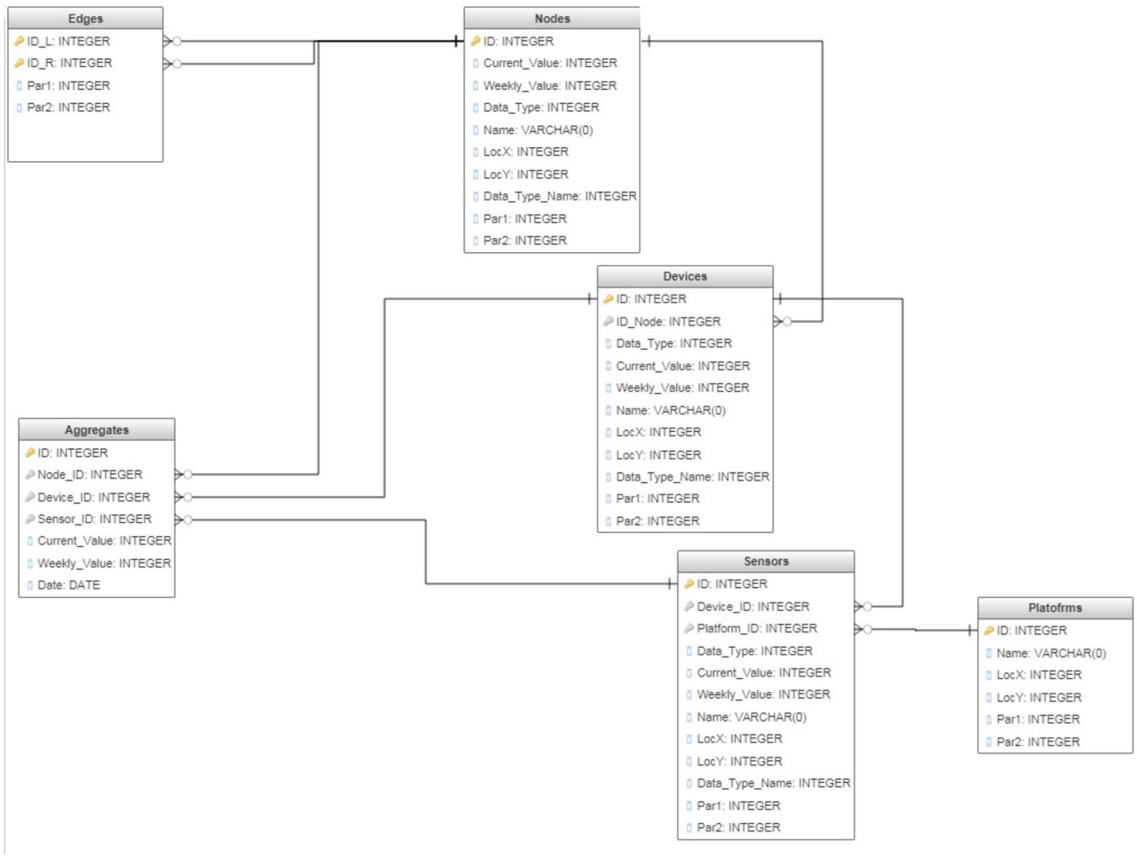


**Fig. 4** Microsoft SQL server RLDB graph model

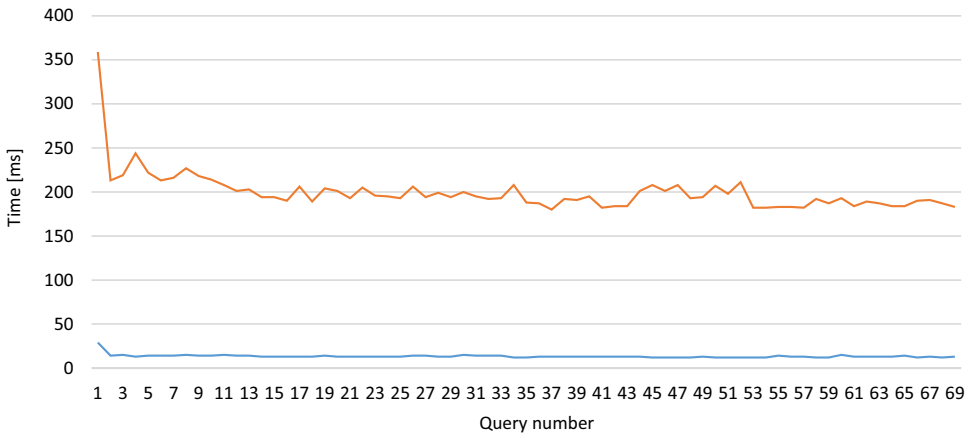


**Fig. 5** 70 queries for 1000 middle nodes

## Conclusions

In this paper, we propose the idea of representing the linear infrastructure of, e.g., railways, waterworks, canal, gas pipeline, heat pipelines, electric power lines, cable ducting or roads. The schema presented in [12] and explained in section "The Linear Infrastructure Representation" enables us to directly represent the real connections between LI objects and connect them with devices, meters, and sensors present in the system.

The graph generator creates a structure similar to physical infrastructure and combines virtual nodes with real data about real objects transferred from the NMS database. The tests show the response times for queries about certain paths in a graph structure and compare the results for the

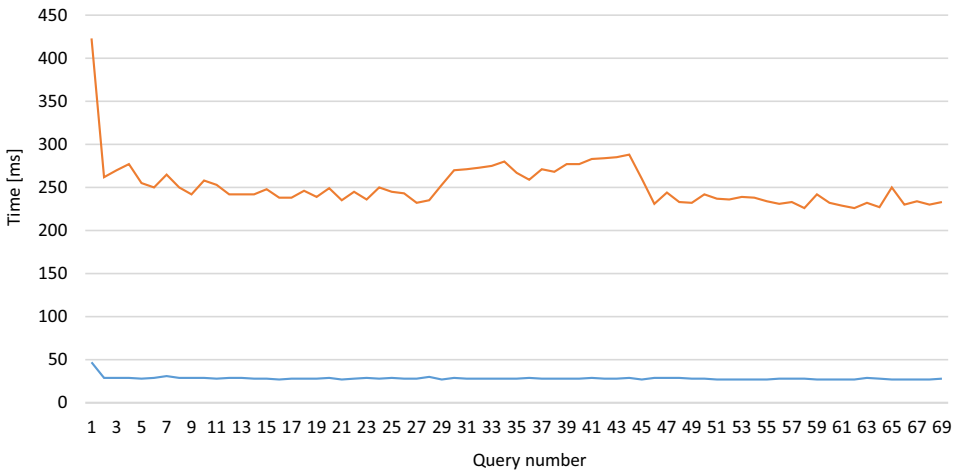


**Fig. 6** 70 queries for 3000 middle nodes

test Microsoft SQL Server database and the Neo4j graph database. The test clearly shows the superiority of the Neo4j database, thus confirming our choice of technology.

The bottleneck of this solution is quite a long database creation time for Neo4j; however, since, as in classical databases and data warehouses, this is a one time process, and the possible updates are much faster, long building time is an acceptable drawback in Smart City systems.

**Funding** This research was funded by Polish National Center for Research and Development Grant number POIR.04.01.04-00-0005/17.



## References

1. Weber M, Podnar arko IA. Regulatory view on smart city services. Sensors (Basel). 2019;19(2):415. https://doi.org/10.3390/s19020415 (PubMed PMID: 30669576; PubMed Central PMCID: PMC6358906).
2. Giffinger R, Fertner C, Kramar H, Kalasek R, Pichler-Milanovic N, Meijers E. Smart cities. In: Ranking of European Medium-Sized Cities. Centre of Regional Science, Vienna UT; Vienna, Austria: 2007. Final Report.
3. Gorawski M, Skrzewski M, Gorawski M, Gorawska A. Neural networks in petrol station objects calibration. In: International conference on algorithms and architectures for parallel processing. vol. 714–723, 2015.
4. Gorawski M, Gorawska A, Pasterak K. The TUBE algorithm: discovering trends in time series for the early detection of fuel leaks from underground storage tanks. Expert Syst Appl. 2017;90:356–73.
5. Praks P, Kopustinskas V, Masera M. Monte-Carlo-based reliability and vulnerability assessment of a natural gas transmission system due to random network component failures. Sustain Resil Infrastruct. 2017;2(3):97–107. https://doi.org/10.1080/23789689.2017.1294881.
6. Szoplik J. The gas transportation in a pipeline network. Adv Nat Gas Technol. 2012;. https://doi.org/10.5772/36902.
7. von Rhein J, Henze GP, Long N, Fu Y. Development of a topology analysis tool for fifth-generation district heating and cooling networks. Energy Convers Manag. 2019;196(15):705–16.
8. Blommaert M, Salenbien R, Baelmans M. An adjoint approach to thermal network topology optimization. In: Computational methods and simulations, IHTC-16; 2018. pp. 2081–9. https://doi.org/10.1615/IHTC16.cms.024074.
9. Herrera M, Edo A, Stoianov I. A graph-theoretic framework for assessing the resilience of sectorised water distribution networks. Water Resour Manag. 2016;. https://doi.org/10.1007/s11269-016-1245-6 (2081-2089, 2018, 10.1615/IHTC16.cms.024074).
10. Foltte J-C, Girardet X, Clauzel C. A methodological framework for the use of landscape graphs in land-use planning. Landsc Urban Plan. 2014;124:140–50. https://doi.org/10.1016/j.landurbplan.2013.12.012 (ISSN 0169-2046).
11. https://patents.google.com/patent/US9473368B1/en. Accessed 20 Jan 2019.
12. Gorawski M, Grochla K: Graph representation of linear infrastructure in Smart City IoT Systems. In: ICUMT 2019, to be published
13. https://neo4j.com/. Accessed 20 Jan 2019.
14. https://db-engines.com/en/ranking/graph+dbms. Accessed 20 Jan 2019.
15. https://www.w3.org/TR/sparql11-overview/. Accessed 20 Jan 2019.
16. Cypher Query Language. https://neo4j.com/developer/cypher-query-language/. Accessed 20 Jan 2019.