Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Performance Analysis of Graph Queries

Master's Thesis

Author:

Zsolt Kővári

Advisor:

Gábor Szárnyas
Dr. István Ráth

2015.

Contents

Kivonat

A modell központú rendszerek tervezése során kulcsfontosságú a modellen kiértékelt lekérdezések optimális teljesítménye, ennek elérése érdekében pedig a megfelelő végrehajtó eszközök megválasztása. Az elmúlt években különböző NoSQL adatbázis-kezelő rendszerek váltak népszerűvé, amelyek célja a gyors lekérdezés kiértékelés és skálázhatóság biztosítása.

A lekérdezés kiértékelésének teljesítményét nagy mértékben befolyásolja a modell topológiája és a lekérdezés komplexitása. Célunk az, hogy bizonyos, a modellt és a lekérdezést leíró metrika felhasználásával, kapcsolatot találjunk a metrikák és a teljesítmény között, úgy, hogy az adott metrikákat ismerve, a teljesítmény megjósolható legyen.

A metrikák alapján tervezési szintű döntéseket hozhatunk az optimális teljesítményre törekedve, továbbá, ez a tudás lehetőséget teremt a valósidejű lekérdezés optimalizálás területén is arra, hogy a modellt és a lekérdezést jellemző metrikák alapján döntsünk optimalizálást érintő kérdésekben.

Különböző NoSQL adatbázis-kezelő rendszereket vizsgálva, regressziós analízis segítségével olyan metrikák után kutatunk, amelyek képesek az adott rendszer futási teljesítményét jellemezni. A vizsgálandó adathalmazok előállításához, egy valós modellből kiindulva, különböző eloszlású és topológiájú gráfokat generálunk.

További célunk az, hogy eldöntsük, vajon egy tetszőleges struktúrájú adatmodellre képesek vagyunk-e a regressziós analízisünk alapján a teljesítményre becslést adni, illetve a modellhez a megfelelő, készletünkben lévő adatbázis-kezelő rendszert társítani, a legjobb teljesítményre koncentrálva.

Végezetül, a valós idejű lekérdezések optimalizálásától motiválva, választ keresünk arra, hogy egy tetszőleges adatmodellre, költséghatékony metrikaszámítással a teljesítmény előre megbecsülhető-e.

Abstract

Achieving the optimal performance of query evaluations plays an important goal in model-based systems. In the last few years, different NoSQL database systems were introduced to provide a better performance and solve the problem of scalability.

The performance of query evaluations depends on the topology of the model, and the complexity of the particular query. Our primary goal is that — by defining model and query-related quantitative metrics — to find a considerable connection between metrics and the performance, and thus, be able to predict the query evaluation time.

Based on the metrics and their effect to the performance, we are able to make decisions in design to achieve on the optimal performance. Furthermore, this knowledge can be utilized in the area of real-time query optimization engines as well.

We investigate various NoSQL database systems via regression analysis in order to find different model-related metrics that are suited to characterize their performance appropriately. Based on a real model, we generate graphs with various topologies and distributions to find metrics from different aspects.

Furthermore, we explore that whether an arbitrarily structured model's performance is predictable via our regression analysis, and also predict which database system from our scope can be associated to the model in order to achieve an optimal performance.

Finally, being motivated by the real-time optimization engines, we search answers whether by reducing the cost of metric calculations, an arbitrary model's performance is still predictable.

Introduction

Chapter 1

Background

1.1 The Trainbenchmark Framework

In the following section, we introduce Train Benchmark, a framework on which our search is based to find empirical connections between different models and the related evaluated queries in order to estimate query evaluation time. The next few sections explain the main goal of the framework and represent the most important components in the system on which we will concentrate in the later chapters. Apart from these, we also allude some obsolete parts of the framework and emphasize their disadvantages in order to make understandable the main motivation of design decisions that resulted in some considerable modifications in the system.

1.1.1 Main Concepts

The basic idea and implementation of the Train Benchmark framework was introduced by the Fault Tolerant Systems Research Group in the Budapest University of Technology and Economics. The implementation is written in Java programming language. Basically, Train Benchmark focuses on the performance of model validations in which the goal is to find those elements from a model that violate the well-formedness constraints. [1] The model validations are accomplished by different databases systems, where the validation appears as a query evaluation, and the measured performance by the system is the evaluation time of the particular query. For the sake of clarity, Figure 1 represents the main steps of the framework's workflow.

The first step is the model generation and the definition of the well-formedness constraints. The Train Benchmark framework uses artificially generated graph-based models with the same domain, it and describes different constraints on them. Since the main goal is to assess model validations, a perfectly valid model generation is not advisable, thus, some

[1] A well-formedness constraint is defined on the model and it is considered as a definition of a particular rule belonging to the elements in the model. For example, a constraint can order the cardinality of the elements, or it can define maximum bounds for attributes, etc.

fault injections are required during the generation phase, when a subset of the elements (typically 2-5%) violates the well-formedness constraints.

The second step is loading the model to the database systems. Subsequently, the generated model is already accessible from the database's repository, and stored in their own format. The next phase is the model validation itself, when a validation — defined in the particular database's query language — is evaluated and its result set includes those erroneous elements from the model that violate the well-formedness constraints. Thus, a model validation occurs and the required time of the query evaluation is measured as well. The validation time will play the most important indicator in the performance comparison among different databases. To summarize, the Train Benchmark framework uses artificially generated graph-based models to investigate the model validation times and thus, the performance of different database systems. In the following, we introduce the domain of the model and its typical characteristics, furthermore, we also define the scope of the framework, as represent the used databases.

1.1.2 The Metamodel

As we referred in 1.1, there were some components in the Train Benchmark framework that needed to reconstruct in order to obtain precise statistical analysis of model and query metric relationships. One of them was the original domain of the framework, the metamodel. However, (i) to understand the motivations of changing the domain entirely, (ii) to see the contrast between the two of them, and (iii) also recognize the possible virtues that we obtained by introducing a now domain, it is important to represent the original domain in this section.

At first, let define the basic related concepts.

Metamodel and Instance Model A metamodel is considered as an abstract model of another model, where the prior — the metamodel — defines the available elements and their cardinalities, additionally, it order rules, constraints on the elements which are used in the latter. Then, a concrete instance of a metamodel is called instance model which can use those elements that are defined in the metamodel, and it must also obey the specified rules and constraints, otherwise, it is considered as an invalid model which violates the constraints in the metamodel.

The domain, or the metamodel[2] of Train Benchmark is depicted in Figure 2, and it is related to a railway network (here comes the name of the benchmark framework). A train route is defined by a sequence of sensors. Sensors are associated with track elements which are either segments (with a specific length) or switches. A route follows certain switch positions which describe the required state of a switch belonging to the route.

[2]We use the concepts of domains and metamodels as synonyms in this paper.

Different route definitions can specify different states for a specific switch. Each route has a semaphore on its entry and exit [10].

The instance models are not related to any real life data, since the instances are generated artificially, and each cardinality of the elements originally was chosen arbitrarily. As a consequence, we cannot associate our models to real life topologies, and thus, we are not able to draw conclusions after our benchmark results in real life situations. Figure 3 shows the cardinalities of the elements, which leads to the fact that segments represent the majority of the elements.

Later, we will query the inflexibility of this metamodel, referring to some heterogeneity problems, and decide to change the domain entirely that is already appropriate for our purpose.

1.1.3 Supported Formats

The statement that the Train Benchmark framework assesses the performance of different database systems is not entirely true. Actually, the scope of the framework is extended, since we also measure various application programming interfaces (API) which are able to store the data in memory, and behave as embedded databases, furthermore, we investigate the performance of different query engines and business management tools. In order to avoid the misunderstandings and use a common expression for these systems, from now on, we reference them as tools.

The currently supported tools can be categorized into four different formats. These categorizations do not only mean conceptual differences, but these appear in the framework's architecture as well. The formats are the following:

- SQL: This category includes the Relational Database Management Systems (RDBMS) that are based on relational models. The format was given its name after the query language of these tools that is formulated in Structured Query Language (SQL).
- EMF: The EMF abbreviation denotes the Eclipse Modeling Framework that facilitates the usage of model-driven development, as it focuses on the creation and applicability of domain models [5]. Previously, in Section 1.1.2, the metamodel in Figure 1 was created with EMF.
- Graph: That tools are involved in this format which operate as graph databases and typically use the GraphML format for their primary data storage structure [9].
- RDF: The majority of the tools are related to the Resource Description Framework (RDF). Our search highly concentrates on these systems, therefore, Section 1.4 introduces the main concepts of RDF in details.

In the further sections, we only focus on the subset of Graph- and RDF-based tools.

1.1.4 Performance Comparison and Uniformity

After introducing the main concepts, the metamodel, and the different formats belonging to Train Benchmark, it is necessary to explain the performance comparison in more details.

Initially, clarify the meaning of performance in the case of Train Benchmark. The key concept on which the framework concentrates the most is the model validation, and thus, under the concept of performance is meant the required time of model validations. The measured validations are executed by various tools, and thus, a concrete validation time characterizes the performance of the particular tool. To summarize, the Train Benchmark framework assesses various tools and their query evaluations performance via model validations.

One important approach of the performance investigation is the scalability of the tools. In order to asses scalability, the benchmark uses instance models of growing sizes, and each model contains twice as many model elements as the previous one.

As it was already emphasized, the benchmark framework measures various tools with different supported formats which naturally indicates the diversity of the instance models. In order to prevent dissimilarities among instance models, the Train Benchmark framework constructs abstract models during the generation phase, independently on formats, and then these models are persisted to the individual acceptable formats belonging to the tools. As a result, the model generation does not depend on the specific formats, and thus, a uniform model is mapped and used by the various implementations. The most important consequence from this fact is that the performance comparison becomes possible among the tools, since they all execute the same validations on the same structured models.

Besides uniformity, it is essential to guarantee reproducibility of the instance models. In order to achieve this, we use pseudo-random[3] generators which makes the model precisely reproducible.

1.1.5 The Foundations of the Framework: Tools

In terms of the implemented tools in the benchmark framework, it is already explained that they are categorized into four different formats. These tools are listed in Table 1.1. Unfortunately, we cannot afford in our research to investigate and analyze all of the currently integrated tools from the framework, hence, we chose a subset of the tools, and typically focus on the RDF-based systems. The analyzed tools on which we concentrate are marked in bold in the table.

[3]It is an algorithm for generating a sequence of random numbers, but actually it can be completely determined by a relatively small set of initial values, called the pseudorandom generator's seed.

Name	Format	Implementation	Query Language	Version
AllegroGraph	RDF	C, C++	SPARQL	5.0.0
Blazegraph	RDF	Java	SPARQL	1.5.2
Drools	EMF	Java	DRL	5.6.0, 6.3.0
EclipseOCL	EMF	Java	OCL	3.3.0
EMF-IncQuery	EMF	Java	IQPL	1.0.0
4store	RDF	C	SPARQL	1.1.5
Jena API	RDF	Java	SPARQL	2.13.0
MemSQL	SQL	C++	SQL	4.0
MySQL	SQL	C++	SQL	5.5.44
Neo4j	Graph	Java	Cypher, Core API	2.2
OrientDB	Graph	Java	Gremlin	2.0.8
Sesame API	RDF	Java	SPARQL	2.7.9
Virtuoso	RDF	C	SPARQL	6.1.6

Table 1.1. The implemented tools in Train Benchmark

1.1.6 The Framework's Architecture

It is important to emphasize that our goal in this paper is not entirely related to the main motivation and aim that belong to the Train Benchmark framework. We are not concerned with model validations (or transformations[4]) , instead, we pay more attention to the first query evaluations and the precise characterization of the model. The latter option is entirely missing from the framework, since we cannot calculate metrics connected to the instance models. However, the embedded model validation components — and the integrated tools themselves — are reusable for our purpose.

Figure 4 depicts the main components are found in the benchmark framework. Based on this, it is clarified which components can be reused, and which needs to be extended or reimplemented completely for our purpose. The first considerable alteration is the Generator unit, which is responsible for the graph-based model generation on an abstract layer, meaning that the component operates independently on the existing formats. Which components are responsible for parsing the model to the specific format, those are the FormatGenerator units. Only the RDF-based component needs to be extended. The generation related modifications are described later in details.

As far as the Validation unit is concerned, it can be reused, independently from the fact that we do not consider model validations in our search, however, the query execution units can be utilized.

The Publisher component provides the measurement results and serializes them to the disk in JSON formats.

In summary, the framework's architecture provides an appropriate base to investigate the relationship among the precise characterization of a model, the composition of a particular query, and the evaluation itself.

[4]Some instance model transformations are also occurred in the workflow of Train Benchmark after each validation. The goal is to investigate how the implemented tools perform after certain model modifications so that they execute the same validations and transformations again, iteratively. Since currently this area is out of our scope, we do not describe it in details.

1.2 Basic Statistics

In this section we present the fundamental concepts of the field of statistics. It is important to emphasize that we do not have the intention to cover most of the theorems in statistics, on the contrary, this section only introduces briefly the important concepts on which our search is based. The majority of the definitions can be found in [15].

1.2.1 Study and Experiment

In a **designed experiment** the engineer makes purposeful changes in the controllable variables of the system or process, observes the resulting system output data, and then makes an inference or decision about which variables are responsible for the observed changes in output performance.

In an **observational study**, the engineer observes the process, disturbing it as little as possible, and records the quantities of interest.

The main essential difference between them is that randomized experiments allow for strong claims about causality[5].

1.2.2 Population and Sample

A **population** is a large set of objects of a similar nature which is of interest as a whole. On the other hand, a **sample** is a subset of objects is drawn from a population [7]. Another approach to define the difference between the concepts is that a population includes all of the elements from a set of data, however, a sample consists of one or more observations from the population [8].

For example, the population can be human beings, and the sample is a finite subset of humankind. To be more constructive, in our case the population contains every possible measurement, and a sample includes a several of them, restricted for a particular tool's measurement results. A population can also be imaginary.

The sample size is the number of observations in a sample and commonly denoted by n.

In the following chapters, we will refer to samples, as we build different ones and analyze them, respectively.

1.2.3 Random Variables and Distributions

Random Variables

In statistics, a variable is as an attribute belonging to an entity, and its value can vary from one entity to another.

[5] Causation indicates that one event is the result of the occurrence of another event, which means, there is a causal relationship between the two of them.

When a variable exhibits variability in measurements — as some deviation can be observed between the variable's values — it is considered as a random variable. Assume that X represents a measurement, then the random variable's formula is

$$X = \mu + \epsilon \tag{1.1}$$

where μ represents a constant and ϵ is a random disturbance.

Probability Distributions

The probability distribution of a random variable X is a description of the probabilities associated with the possible values of X. In other words, a probability distribution links each possible value that a random variable can assume with its probability of occurrence.

One differentiates the discrete and continuous distributions, when a random variable is a discrete variable, its probability distribution is called a discrete probability distribution, similarly to the continuous one.[6]

As we will introduce in details later, the base of our model generation is strongly connected to different discrete probability distributions. The ones that will be taken into account are introduced briefly in the following paragraphs [6].

Uniform Distribution A random variable X has a discrete uniform distribution if each of the n values in its range, say, x_1, x_2, ..., x_n, has equal probability. Then,

$$f(x_i) = \frac{1}{n} \tag{1.2}$$

where $f(x_i)$ denotes the probability distribution function.

Pareto Distribution If X is a random variable with a Pareto distribution[7], then the probability that X is greater than some number σ, is given by

$$\overline{F}(x) = Pr(X > \sigma) = \left(\frac{x}{\sigma}\right)^{-\alpha} \tag{1.3}$$

where σ is the scale, and α represents the shape parameter($x > \sigma$, $\sigma > 0$).

A similar approach is the power-law distribution which describes the probability of the random variable X that — on the contrary of the Pareto distribution — is not higher, but equals to x. This leads to

$$f(x) = c \cdot x^{-\gamma} \tag{1.4}$$

[6]From now on, we only use discrete probability distributions, and in every context, we refer to discrete automatically.

[7]More types of Pareto distributions are distinguished, the commonly known Type I distribution is introduced.

where c is a constant and γ is called as the *exponent* or *scale factor*. Later, we will increasingly pay attention to the power-law distributions.

Poisson Distribution The Poisson distribution is characterized by the following elementary probabilities:

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda} \tag{1.5}$$

where $\lambda > 0$ is the shape parameter and $k \in \mathbb{N}$.

1.2.4 Related Measurements

It is essential to introduce some measurements related to random variables and their probability distributions on which we will refer later.

Mean

The *mean* or *expected value* of a discrete random variable X, denoted as μ or $E(X)$, is

$$\mu = E(X) = \sum_x x f(x) \tag{1.6}$$

where x represents the values of the random variable X, and $f(x)$ is the probability distribution function. A mean is a measure of the center of the probability distribution.

Variance

The *variance* of X, denoted as σ^2 or $V(X)$, equals to the following formula:

$$\sigma^2 = V(X) = E(X - \mu)^2 = \sum_x (x - \mu)^2 f(x) \tag{1.7}$$

The variance is a measure of the dispersion, or variability in the distribution, as it represents the average of the squared differences from the mean. For example, a variance of zero indicates that all the values are identical.

Standard Deviation

The *standard deviation* of the random variable X is $\sigma = \sqrt{\sigma^2}$, thus, it is the square root of the variance.

1.2.5 Covariance and Correlation

Covariance

Covariance is a measure of linear relationship between the random variables. The covariance value between two random variables X and Y can be expressed by

$$cov(X, Y) = E\big[(X - \mu_X)(Y - \mu_Y)\big] \tag{1.8}$$

where if $cov(X, Y) > 0$, then Y tends to increase as X increases, and if $cov(X, Y) < 0$, then Y tends to decrease as X increases [2]. A few examples are depicted in Figure 1 in different scenarios where — in the terms of Figure (a) and (b) — a covariance is observable between X and Y, however in Figure (c) and (d), the covariance equals to zero.

Correlation

Similarly to covariance, a *correlation* describes the strength of the relationship between variables. A type of correlation, the Pearson product-moment correlation coefficient is formulated as

$$\rho_{XY} = \frac{cov(X, Y)}{\sigma_X \sigma_Y} \tag{1.9}$$

where $-1 \leq \rho_{XY} \leq 1$, and 1 indicates a total positive linear relationship between X and Y, -1 means negative linearity, finally, 0 is interpreted as a correlation does not exist between the variables.

1.2.6 Regression Analysis

Regression analysis is a statistical technique for exploring the relationship between two or more variables. A regression model can be considered as an equation that relates a random variable Y to a function of a variable x, and a constant β. In formally, a regression model is defined as

$$Y = \beta_0 + \beta_1 x + \epsilon \tag{1.10}$$

where Y is the dependent or response variable, x is called as an independent variable or predictor, and β_0, β_1 are the regression coefficients, the intercept and the slope, respectively. Finally, ϵ is a random error. More precisely, Equation 1.10 is called a *linear regression model*, since it uses one independent variable to predict the outcome of Y.

A multiple linear regression model considers k independent variables, and the equation is extended as the following

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \epsilon \tag{1.11}$$

1.3 Graph Theory

During the model generation later, we will construct different graph topologies, and also introduce metrics that are able to characterize these models precisely. As a consequence, it is important to present the main related concepts in graph theory with the expectation that the theorems of directed graph, connectivity, complete graph, adjacent nodes, and degrees of vertices are already clarified.

1.3.1 Degree Distribution

The spread in the node degrees is characterized by a distribution function $P(k)$, which gives the probability that a randomly selected node's degree equals to k.

1.3.2 Network Topologies

1.4 Resource Description Framework: RDF

Chapter 2

Related Works

The precise characterization of different graph topologies often appears in the literature of graph theory. Similarly, numerous publications can be referred that investigate the performance of various NoSQL databases. However, a precise performance analysis of NoSQL systems that exhibits a connection between the model's structure and the performance of query evaluations, is considered as a new approach.

In this section, we represent some examples that have a specific connection with graph analysis or performance comparison of NoSQL databases.

2.1 Graph Analysis

2.1.1 Studies of Barabási and Albert

Barabási and Albert study the topologies of different real-life networks, and also inspect the natures of various well-known artificial graph models in [11]. For example, they generate Erdős-Rényi random graphs, scale-free networks, and small-world graphs of Watts-Strogatz models. As a main result, they discover that a number of real-life networks follow a power-law degree distribution, and thus, considered as scale-free models. Furthermore, they experience that there are significant differences between the networks, regarding specific graph metrics, both in the real-life and artificial models as well. They show a considerable spread in the *clustering coefficient* and the *average path length* per different networks.

2.1.2 Network Robustness and Metric Correlations

Similarly, [16] and [12] investigate the random graph of Erdős-Rényi, the small-world graph of Watts-Strogatz and the scale-free graph of Barabási-Albert, however, the publishers also inspect the connectivity and robustness of the networks. In their case, a network is said to be robust if its performance is not sensitive to the changes in topology. In [16] the algebraic connectivity metric is studied in relation to graph's robustness to node and link failures, however, they showed that the algebraic connectivity is not trivially correlated to the robustness of the network. The authors in [12] also drew the conclusion that there is no unique graph metric to satisfy both connectivity and robustness objectives while keeping a reasonable complexity, since each metric captures some attributes of the graph.

2.1.3 Chinese Network Analysis

A real-life Chinese railway network is studied in [17], and the authors exhibit that the network shows small-world characteristics and also follows power-law distributions. Besides topologies, they also investigate correlations between strength, degree, and clustering coefficients. However, the publishers do not regard graph queries and do not intend to find connections between metrics and performance, the correlations are exclusively found among the graph metrics.

2.1.4 Conclusions

As the examples represent, there is a particular well-know subset of graph topologies that are widely analyzed. The random graph of Erdős-Rényi, the small-world model of Watts-Strogatz, and finally, a scale-free graph of Barabási-Albert. Moreover, numerous real-life networks show scale-free characteristics. We also investigate these networks topology, by reason of that they follow unlike degree distributions and they also show a variety in graph metrics. As far as metrics are concerned, we also give the assumption that the clustering coefficients and average path length metrics can characterize the model precisely, and thus, they can be considered as main indicators in the estimation of the query performance. Furthermore, we extend the set of observable metrics with the betwenness centrality as well.

Interesting and very important question is whether a unique metric is able to characterize a network's topology, and thus, more interestingly, whether it is suited to predict the performance of graph queries. As it was empirically determined in2.1.2, each metric captures some attributes of the graph, and only one is not adequate for the goal to characterize the network precisely. Besides the empirical results, we conjecture the same that only more than one, different metrics are suited to become appropriate performance estimators.

2.2 Benchmarks of NoSQL Databases

A wealth of literature is available related to the performance comparison of different NoSQL databases over RDF data. Instead of introducing a part of these papers, we focus on the foundations of the measurements, the benchmark frameworks. In the following, each section represents a benchmark particularly connecting to RDF data models. Finally, Section 2.2.5 compares the frameworks, and by drawing conclusions based on the benchmarks, it also contains the main concepts on which we will rely in our search.

2.2.1 Yahoo! Cloud Serving Benchmark

The *Yahoo! Cloud Serving Benchmark* was elaborated in order to compare performance of the new generation of cloud data serving systems [14]. The framework proposes different workloads by assigning different distributions to them that determine the operation to perform — get or put — and the record from the data model to be read or written. In other words, the particular distributions specify the exact numbers of read or update queries in a workload, and also affect the choosing of records on which the workload operates. In order to demonstrate, Figure 1 represents the available distributions and number of choices per records.

2.2.2 Berlin SPARQL Benchmark

The Berlin SPARQL Benchmark (from now on BSBM) is settled in an e-commerce use case in which a set of products is offered by different vendors and consumers have posted reviews about these products on various review sites [13]. Taken scalability into consideration, BSBM proposes an arbitrary increased model size. Similarly to YCSB, BSBM concentrates on read and update operations as well, as it defines three different use cases and a suite of benchmark queries in each of them [1]. The queries were elaborated to simulate realistic, real-life workloads.

In contrast to YCSM, BSBM defines particular performance metrics that relate mainly to the query execution times from different perspectives. For example, the most important metrics are the following:

- Queries per Second: It equals to the number of queries that executed properly by the system within a second.
- Query Mixes per Hour: Denotes the number of *mixed* queries with different parameters that evaluated within an hour.
- Overall Runtime: The overall time that a certain amount of query mixes required.

2.2.3 DBpedia SPARQL Benchmark

DBpedia SPARQL Benchmark proposes datasets of various sizes derived from the DBpedia [3] knowledge base, and perform measurements on real queries that were issued against existing RDF data [18]. Similarly to BSBM, DBpedia also defines metrics to provide a more precise performance analysis. These are the same as in the case of BSBM: *Query Mixes per Hour*, and *Queries per Second*. Furthermore, DBpedia investigates the characterization of the models as well and calculates the average in- and outdegree, the number of nodes and distinct IRIs, however, it uses these measurements to judge and maintain the generated model to be similar to the original data set. As a consequence, DBpedia does not explore model and performance correlations neither.

2.2.4 SP^{2}Bench

The SPARQL Performance Benchmark (SP^{2}Bench) is designed to test the most common SPARQL constructs, operator constellations, and a broad range of RDF data access patterns [19]. Instead of defining a sequence of use case motivated queries, the framework proposes various queries that cover specific RDF data management approaches.

SP^{2}Bench is based on the *Digital Bibliography and Library Project* (commonly known as *dblp*) which provides an open bibliographic information on major computer science publications [4]. The benchmark queries are not explicitly evaluated over the *dblp* dataset, since SP^{2}Bench uses arbitrarily large, artificially generated models for the measurements, however, these models are constructed to follow real-life characteristics that were found in the original *dblp* dataset, such as a power-law distribution.

Similarly to BSBM, SP^{2}Bench also measures additional performance related metrics besides execution time, such as disk storage cost, memory consumption, data loading time, success rates, and every one them captures different aspects of evaluations.

2.2.5 Conclusions

One common attribute of the previously introduced benchmark frameworks is that they lack precise analysis of searching connection between model characteristic and evaluation performance. In these frameworks, it is not supported to assess a tool's performance on different topologies, still using the same domain. Generally, the frameworks propose the generation of variously large models, thus investigating scalability, however, they do not focus on modifying the internal structures of the models, and generate different topologies, even in the same size.

Acknowledgements

List of Figures

List of Tables

Bibliography

[1] Berlin SPARQL Benchmark (BSBM) Specification V3.1. http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/.

[2] Covariance. http://mathworld.wolfram.com/Covariance.html.

[3] DBpedia. http://wiki.dbpedia.org/.

[4] Digital Bibliography and Library Project. http://dblp.uni-trier.de/db/.

[5] Eclipse Modeling Framework. https://www.eclipse.org/modeling/emf/.

[6] A guide on probability distributions. http://dutangc.free.fr/pub/prob/probdistr-main.pdf.

[7] Population and Sample Definition. http://www.statistics.com/glossary&term_id=812.

[8] Populations and Samples. http://stattrek.com/sampling/populations-and-samples.aspx?Tutorial=AP.

[9] The GraphML File Format. http://graphml.graphdrawing.org/.

[10] The TTC 2015 Train Benchmark Case for Incremental Model Validation. https://www.sharelatex.com/github/repos/FTSRG/trainbenchmark-ttc-paper/.

[11] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.

[12] Alireza Bigdeli, Ali Tizghadam, and Alberto Leon-Garcia. Comparison of network criticality, algebraic connectivity, and other graph metrics. In *Proceedings of the 1st Annual Workshop on Simplifying Complex Network for Practitioners*, SIMPLEX '09, pages 4:1–4:6, New York, NY, USA, 2009. ACM.

[13] Christian Bizer and Andreas Schultz. The berlin sparql benchmark. *International Journal On Semantic Web and Information Systems*, 2009.

[14] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[15] George C. Runger Douglas C. Montgomery. *Applied Statistics and Probability for Engineers.*

[16] A. Jamakovic and S. Uhlig. On the relationship between the algebraic connectivity and graph's robustness to node and link failures. In *Next Generation Internet Networks, 3rd EuroNGI Conference on*, pages 96–102, May 2007.

[17] W. Li and X. Cai. Empirical analysis of a scale-free railway network in china. *Physica A: Statistical Mechanics and its Applications*, 382(2):693 – 703, 2007.

[18] Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. Dbpedia sparql benchmark: Performance assessment with real queries on real data. In *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, ISWC'11, pages 454–469, Berlin, Heidelberg, 2011. Springer-Verlag.

[19] Michael Schmidt, Thomas Hornung, Michael Meier, Christoph Pinkel, and Georg Lausen. Sp2bench: A sparql performance benchmark. In Roberto de Virgilio, Fausto Giunchiglia, and Letizia Tanca, editors, *Semantic Web Information Management*, pages 371–393. Springer Berlin Heidelberg, 2010.