Journal of Computer and System Sciences 82 (2016) 45–68

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Querying a graph database – language selection and performance considerations

Florian Holzschuher [1], René Peinl *,[2]

iisys, Hof University, Alfons-Goppel-Platz 1, 95028 Hof, Germany

ARTICLE INFO

Article history:
Received 26 October 2014
Received in revised form 9 April 2015
Accepted 25 June 2015
Available online 7 July 2015

Keywords:
Graph databases
Neo4j
Graph query processing
Benchmarks
Performance optimization
WebSocket

ABSTRACT

NoSQL and especially graph databases are constantly gaining popularity among developers as they promise to deliver superior performance when handling highly interconnected data compared to relational databases. Apache Shindig is the reference implementation for OpenSocial with a highly interconnected data model. However, it had a relational database as back-end. In this paper we describe our experiences with the graph database Neo4j as back-end and compare Cypher, Gremlin and Java as alternatives for querying data with MySQL. We consider performance as well as usability from a developer's perspective. Our results show that Cypher is a good query language in terms of code readability and has a moderate overhead for most queries (20–200%). However, it has to be supplemented with "stored procedures" to make up for some performance deficits in pattern matching queries (>1000%). The RESTful API is unusable slow, whereas our WebSocket connection performs significantly better (>650%).

1. Introduction

Relational databases have been the means of choice for storing large amounts of structured data for decades due to their high performance and ACID capabilities (atomicity, consistency, isolation and durability). With requirements changing due to transformation of the IT world caused by the social Web and cloud services, several types of NoSQL databases, often dubbed "Not only SQL", have emerged and are gaining popularity [1,2]. They differ from traditional databases by using a non-relational internal data structure in order to achieve higher performance and scalability in certain scenarios. However, to achieve this they often sacrifice data consistency, availability or partition tolerance (CAP theorem) [1,2]. Among those, graph databases are especially interesting since they often offer a proper query language, which most key-value stores as well as document-oriented databases are currently missing. They perform especially well in domains like chemistry, biology and social networking [3]. Particularly Web 2.0 data in social networks is highly interconnected, e.g., networks of people, comments, ratings, tags and activities. Modeling such graphs in a relational database causes a high number of many-to-many relations. Complex join operations are needed to retrieve such data. Graph databases on the other hand are specifically designed to store such data and to deliver a high performance traversing them. To assess the suitability and performance of a graph database-based solution, we chose to implement different back-ends for *Apache Shindig*,[3] the *OpenSocial*[4] reference

* Corresponding author.
E-mail addresses: florian.holzschuher@iisys.de (F. Holzschuher), rene.peinl@iisys.de (R. Peinl).
[1] Fax: +49 9281 409 6214.
[2] Fax: +49 9281 409 4820.
[3] https://shindig.apache.org/.
[4] http://opensocial.org/.

http://dx.doi.org/10.1016/j.jcss.2015.06.006
0022-0000/

implementation, and measure their performance with realistic sets of data, as well as judge their practicability. *Shindig* is a *Java* web application supplying and managing social network data via a RESTful API as well as offering rendering of *JavaScript*-based, embeddable *OpenSocial* gadgets. Our use case considers an enterprise social network, where user data is accessed in user profile pages, short messages can be sent and people's activities are shown in an *activity stream*. For our effort, we chose *Neo4j* [3], a *Java*-based open source graph database that offers persistence, high performance, scalability, an active community and good documentation. Furthermore, two different query languages can be used to access data in *Neo4j*, *Cypher*,[5] which is declarative and a bit similar to SQL, as well as the low level graph traversal language *Gremlin*.[6] We compare both regarding performance and usability against native traversal in Java as well as SQL with the Java Persistence API (JPA) and MySQL. JPA is used for accessing relational databases, offering object-relational mapping, query generation and offers generic query language. Our hypotheses were, that Cypher is the best choice as a query language (H1) and Neo4j is a faster database back-end for Apache Shindig than JPA and MySQL (H2).

RESTful HTTP (REpresentational State Transfer) is a common connection choice for a NoSQL database. AllegroGraph, CouchDB, Neo4j and Riak are examples of NoSQL databases using REST as their primary interface (see nosql-database.org). Based on the high performance achieved with an embedded Neo4j instance and a low using the RESTful HTTP, our next hypothesis was that a WebSockets-based database connection will perform better than a RESTful HTTP one (H3). We assumed that Neo4j's core was already highly optimized and fast, but remote access was unable to fully harness this performance.

Since computer engineers put considerable efforts into connection pooling and similar optimization strategies for JDBC and other database connectivity technology [4–6], we further assumed that using multi-threading and compression should increase throughput of the server (H4). Considering a software as a service (SaaS) use case, our next hypothesis was, that our WebSocket solution is able to scale well horizontally (H5) and that it makes more sense to horizontally scale the database instead of the application (H6).

Our goal was to analyze both performance of the query language as well as that of the data connection and transmission.

The remainder of the paper is structured as follows. We first discuss related work in the areas of query languages and especially other benchmarking approaches for graph databases. Then we present the benchmark setup, before comparing the different query languages from a developer's perspective. The subsequent presentation of benchmark results starts with a comparison of Neo4j's performance with that of Shindig's JPA back-end. We continue with a comparison of Gremlin and Cypher within Neo4j and finally Cypher with native access. After that, we focus on the WebSocket approach and compare it to embedded Neo4j and Cypher over RESTful HTTP. We continue presenting the impact of compression, multi-threading and multiple WebSocket connections on performance. Finally, we perform scaling tests in different setups before analyzing the results. We finish with discussing limitations, future work and a conclusion.

2. Related work

Our paper builds upon previous work in two main areas, which are (graph) query languages, as well as benchmarks, especially those comparing graph databases and relational ones. We further analyzed test data generation and query selection.

2.1. Query languages

Many NoSQL databases do not help the developer in querying the data stored in the database management system (DBMS). Stonebraker and Cattell [7] state that this is can be avoided and advise clients to choose a DBMS that offers a high level language which can still offer high performance. Query languages have always been a key to success of database systems. The prevalence of relational database systems in the last decades is tightly coupled with the success of SQL, the structured query language [8,9] and extensions of SQL like SQL/XML [10] helped RDBMS to remain the common choice for most data. Soon after a new type of database system arose, a query language was developed to query data in the respective format, e.g., XPath and XQuery for XML databases [11], OQL for object-oriented databases [12] or MDX for multi-dimensional databases [13]. In case of querying RDF data, there even was a serious contest between multiple query languages like RQL, RDQL and SeRQL that competed to become the W3C standard [14], before SPARQL [15] finally arose as the winner of this battle.

Besides those RDF graph query languages, which are implemented in RDF triple stores like Jena or Allegro GraphDB, there are also proposals for other graph query languages that can be used for general purpose graph databases. Graphgrep for example, is an application independent query language based on regular expressions [16]. GraphQL was proposed in 2008 [17] and introduced its own algebra for graphs that is relationally complete and contained in Datalog. It was implemented in the *sones GraphDB* [18] which achieved some attention in Germany before the company behind it went bankrupt in 2012. Angles [18] also mentions Cypher, the main query language of Neo4j, which is a declarative language similar to SQL. Barceló Baeza [19] analyzes a number of graph query languages from a theoretical perspective in order to compute the complexity. They conclude that none of the languages used in practice would have a clear semantic and syntax and it would be difficult to compute their expressiveness and computational costs.

[5] http://neo4j.com/developer/cypher-query-language/.
[6] https://github.com/tinkerpop/gremlin/wiki.