

Large Knowledge Bases

Storage and Distribution

PAIVA LIMA DA SILVA Bruno

Université Montpellier 2 (EPI GraphIK - INRIA - CNRS)

February 28, 2011

- 1 Introduction
- 2 Existing work
- 3 Own work
- 4 Short demo
- 5 Future work
- 6 Questions

Table of Contents

- 1 Introduction
- 2 Existing work
- 3 Own work
- 4 Short demo
- 5 Future work
- 6 Questions

Problem description

Beginning of the PhD in October 2010 within the GraphIK team.

Problem:

- Conjunctive query answering over large knowledge bases.
- Emergence of many, and larger and larger knowledge bases all around. (Social networks, ontologies, Linked Data, etc.)

New paradigm:

- How does the fact that the size of KBs is increasing very fast affect our problem?

Possible approaches

Three different approaches to address the deduction problem when dealing with large KBs:

- Algorithmic approach: Filtering, reducing exploration path, etc. (Team and other teams previous work)
- Approximative approach: “Pseudo-deduction”, probabilistic approach. (See Nicolas and others work)
- Storage methods: To check (and to find) one or more (the best) storage method that would enable us to perform conjunctive query answering with good efficiency. (Context of this thesis)

Well-known storage methods

Well known storage methods:

Graphs in main memory storage.

- Known by having solid efficiency for most basic operations needed for reasoning.

Problem:

- Large does not fit in main memory.

Relational databases

- Known for supporting very well large knowledge bases, stored in main or secondary memory.
- Deduction may be made by using a SQL interface. (also translatable in BackTrack algorithms)

Problems:

- SQL: Bad with semi-structured data, joining tables become highly “expensive” very quickly.
- Backtrack: Uses SQL for every basic operation, loss of efficiency in almost every basic operation call.

RDBs May work very well in some cases (structured data), but may also very poorly in other situations.

Objectives

Thesis goals:

- To identify different storage systems adapted to our problem.
- To have tools that enable us to compare several storage system between themselves.
- To answer: Which storage works better than the other? For which kind of data?

Storage paradigms:

- To compare them when testing real data vs. unbiased generated data.
- To find ways to improve current algorithms for these systems based on our results.

Table of Contents

- 1 Introduction
- 2 Existing work**
- 3 Own work
- 4 Short demo
- 5 Future work
- 6 Questions

Existing Work

- Description of the NoSQL “movement” and some of its features.
- Quick overview of some NoSQL popular projects and their particular characteristics.
- Currently available Graph databases.
- Description of Blueprints: a tool that uses a software-graph vision for the unification of differently implemented graph databases.

NoSQL: Short history

- 1998: Carlo Strozzi started the "NoSQL" project, a relational database without SQL interface.
- The project failed. He said that the problem was not on SQL, but on relational model.
- "NoRel" ?
- NoSQL became then "Not Only SQL".
- 1999: First NoSQL Conference, called no:sql(east)
- Worst tagline ever: "select fun, profit from real_world where relational=false"

NoSQL: What is it?

- "Not Only SQL" may also be easily seen as "Non-relational".
- That said, the NoSQL community around the world may also be easily seen as the union of different databases management systems that differ from the classic relational database model from Codd.
- Some of those systems may be put together into groups: Key-Value store, Object databases, XML databases, Graph databases, Document store. (Document as in JSON)

CAP Theorem

History:

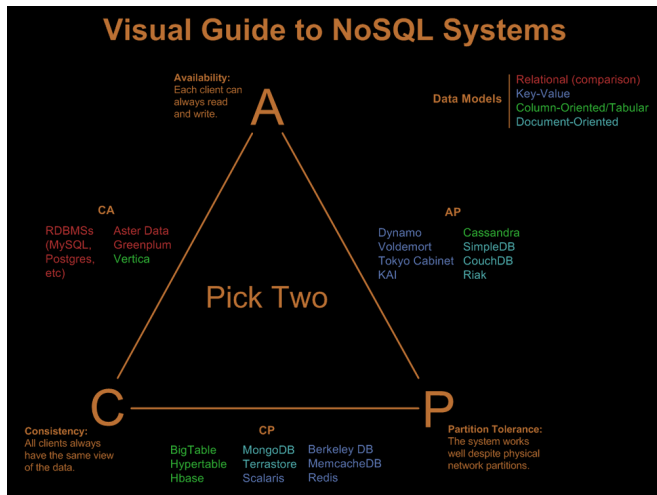
- 2000: Conjecture made by Eric Brewer.
- 2002: Formally proved by MIT scientists.

Theorem:

“A distributed storage system can satisfy any two of the following guarantees at the same time, but not all three:”

- Consistency.
- Availability.
- Partition tolerance.

CAP Theorem



NoSQL: Popular NoSQL projects

Some NoSQL have become popular for being supported by big IT companies:

- Google BigTable (HBase implementation for Hadoop)
Storage: Distributed multidimensional maps, Querying: GQL.
- Project Voldemort (Linkedin)
Storage: Distributed Key-Value, Querying: Hashtable semantics.
- Apache Cassandra (Facebook, Digg)
Storage: Distributed Key-Value, Querying: SQL-like.
- MongoDB (foursquare)
Storage: BSON (Binary JSON) Documents, Querying: SQL-like.

NoSQL: Current graph DBs

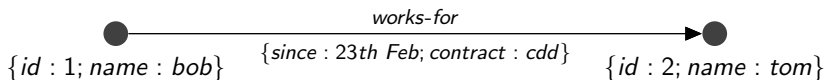
There are today several different implementations of Graph databases available for commercial and personal use:

- Neo4J,
- InfiniteGraph,
- sones,
- DEX,
- OrientDB,
- HyperGraphDB,
- and others...

Some can however be regrouped since they do implement a “common” graph model:

Property Graph Model

[Source: UIM-GDB “Connections to the real world” presentation]
The Property Graph model:



Characteristics of the model:

- directed: Each edge has a source and destination vertex.
- attributed: Vertices and edges carry key-value pairs.
- edge-labeled: The label denotes the type of the relationship.
- multigraph: Multiple edges between any two vertices allowed.

Blueprints project



- Blueprints is a project from TinkerPop team. (± 7 developers)
- Collections of interfaces and implementations for the Property Graph Model.
- Brings interoperability between Graph DBs by allowing developers to plug-and-play several Graph DB backends.
- Also unifies the indexing and transactions methods for those backends.

TinkerPop Team

Other projects from the team that has created Blueprints:

- Gremlin: A graph traversal language based on Blueprints implementations.
- Rexster: Web service providing basic REST (GET, POST...) methods for stored Blueprints graphs.
- Graphdb-Bench: A Graph DB benchmarking tool that allows one to define and run benchmarks against different Graph DB implementations.



Table of Contents

- 1 Introduction
- 2 Existing work
- 3 Own work**
- 4 Short demo
- 5 Future work
- 6 Questions

Details of the work

Detailed below, the first steps of my work on the subject:

- To find all the potential storage methods and implementations that could be interesting to test against the classic methods.
- To design an architecture with a logic based vision that regroups different storage systems and enable us to perform operations over each one of them using an unique language.
- To write code clean enough to enable our architecture to scale up horizontally and vertically without having to restart all over again.
- To have an abstract platform solid enough that could regroup previous and future works of GraphIK team within an only software architecture.

Problems ahead

- Not every NoSQL, and principally Graph DB implementation available is free or open source. (not easily testable then)
- Some Graph DBs may be easily plugged because of the Blueprints project, others, less easily. There is not actually a global notion of interoperability between those DBs yet.
- Reapparition of an old dilemma in software engineering: More general code vs. Specific and efficient code.

API Architecture

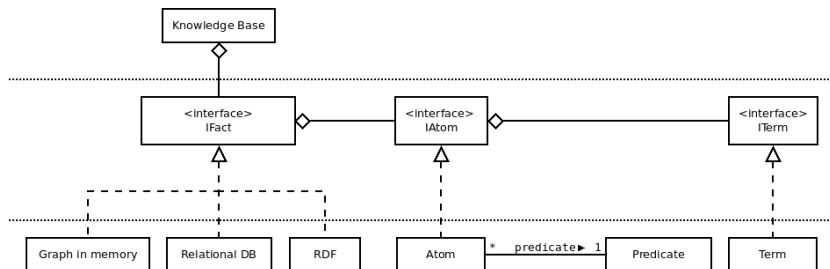


Figure: Current class diagram of our architecture.

Application description

- Implementation of an IFact interface that ensure that all the storage systems plugged will answer to the same methods.
- Implementation of Atom, Term and Predicate classes that allows all structures to communicate using a common type of data.
- In the case of our work, we have decided to use, by default, the fact that if two terms do have the same label, then they represent the same entity.

Technical details

- Due to the facility of plugging several different pieces of code together, we have decided to write our framework in JAVA.
- Pros: Almost everything can be easily plugged in.
- Cons: Loss in speed and efficiency.
- For now, our structure can handle a Relational DB (Sqlite) and adjacency lists stored graph. (in memory)
- We have then written a simple command-line interface in order to interact with the framework. (See demo next)
- The command-line interface instantiates an (almost) empty KB and has several commands for storing and querying our pieces of information.

Table of Contents

- 1 Introduction
- 2 Existing work
- 3 Own work
- 4 Short demo**
- 5 Future work
- 6 Questions

Screenshot of the application

```

bplsilva@graphik: ~
Fichier Edition Affichage Rechercher Terminal Aide
bplsilva@graphik:~$ java -jar apigraph.jar
[--- Command-line API Graph User Interface ---]
> help
API Graph help:
In this application...

Currently are available the storage systems listed below:
First Order Logic fact [fol], Adjacency List Graph [adjList], Sqlite RDB [sqlite]

Available commands:
- addConjunction [atomConjunction] [folFact] - Adds an atom conjunction to a logic fact.
- add [atom] [fact] - Adds an atom to a fact.
- checkValidity [atomConjunction] - Checks the validity of a FOL atom conjunction.
- copy [sourceAtom] [destAtom] - Copy all the atoms from a fact into another.
- getAtoms [fact] - Loads all the atoms of a fact into memory.
- hom [requestFact] [baseFact] - Checks if there is an homomorphism from requestFact into baseFact.
- list - Returns the list of all current facts.
- memory - Returns the list of all atoms currently stored in memory.
- new [factType] [factName] - Creates a new fact of a given type in the knowledge base.
- print [fact] - Prints the content of a fact.
- sql [sqlfact] [request] - Performs a SQL request on a RDB fact.
- writeAtoms [fact] - Inserts all atoms loaded in memory into a fact.

- exit - Terminates the program.

> new adjList g
SUCCESS: Adjacency List Graph "g" has been created and added to the base.
> add p(a,b) g
SUCCESS: The atom was successfully added to fact "g".
>

```

Figure: Application running on Ubuntu Linux.

Table of Contents

- 1 Introduction
- 2 Existing work
- 3 Own work
- 4 Short demo
- 5 Future work**
- 6 Questions

Near future

We can list some things that we are looking to do in the near future:

- To run all the tests needed to confirm or to discard our initial hypothesis.
- To add to our framework the possibility of managing RDF and 3-Store data and to compare them aswell.
- To start testing our framework structure with **larger and larger** datasets.
- To start identifying which storage method is more appropriated than other according to the kind or structure of the data we are dealing with.

Then after...

And also some that we will surely look in a not-so-near future:

- To implement a benchmarking tool that would help us to compare properly the storage systems we have when dealing with real large datasets against unbiased generated ones.
- To identify more operations/use cases for which we can compare our data structures.
- To develop/build a Graphic User Interface on top of our framework.

Table of Contents

- 1 Introduction
- 2 Existing work
- 3 Own work
- 4 Short demo
- 5 Future work
- 6 Questions**

(More) Questions & (more) comments...