Searching heterogeneous data-sources: Master thesis problem statement

Vidmantas Zemleris, October 9, 2012

1 Introduction

At large scientific collaborations like the CMS Experiment at CERN's LHC that includes more than 3000 collaborators data usually resides on a fair number of autonomous and heterogeneous proprietary systems each serving it's own purpose [1]. As data stored on one system may be related to data residing on other systems[2], users are in need of a centralized and easy-to-use solution for locating and combining data from all these multiple services.

Using a highly structured language like SQL is problematic because users need to know not only the language but also where to find the information and also lots of technical details like schema. A data integration system based on simple structured queries is already in place. Still various improvements including support for less restricted keyword queries and improvements to system's usability and performance still have to be researched.

2 Case study: the CMS Experiment at CERN

Users' information need may vary greatly depending on their role, however most of the time they are interested in locating a *full set* of entities matching some selection criteria, e.g.:

- find all *files* from *dataset(s)* matching wild-card query each containing some of the 'interesting' *runs* from a list provided (Release validation teams)
- find (all) *datasets* related some specific physics phenomena[3] together with conditions describing how this data was recorded by detector or simulated which are present in separate autonomous system than the datasets (Physicists)
- find all *datasets* matching some pattern stored at a given *site* (filtering on entities stored on separate services)

For more use-cases of data retrieval at CMS Experiment see [DGK+08].

The Data Aggregation System

The Data Aggregation System (DAS)[KEM10, BKEM11] was created which allows integrated access to a number of proprietary data-sources by processing user's queries on demand - it queries the data-sources, merges the results, and caches them for subsequent use. DAS uses *Boolean retrieval model* as users are often interested in retrieving ALL the items matching their query.

Currently the queries specify what entity the user is interested in (dataset, file, etc) and provide selection criteria (attribute=value, name BETWEEN [v1, v2]) operators. The combined query results could be later 'piped' for further filtering and aggregation (min, avg, etc), e.g.:

```
dataset=*RelVal* | grep dataset.nevents >1000 | avg(dataset.size), median(dataset.size)
```

The query above would return average and median datasets sizes of ones containing *RelVal* in their name having more than 1000 events.

Queries could be run either from web browser or through command line interface where the results could be fed into another application (e.g. program doing physics analysis or automatic software release validation).

[1] For instance, at CERN, due to many reasons (e.g. research and need of freedom, politics of institutes involved) software projects usually evolve in independent fashion resulting in fair number of proprietary systems[KPGM00]. Further high turnover makes it harder extending these systems

[2] For example, datasets containing physics events are registered at DBS, while the physical location of files is tracked by Phedex which also takes care of their transfers within the worldwide grid storage

[3] In case of dataset this data is present in filename or run

3 Problem statement

DAS is based on *Virtual Integration* where data is left at the sources allowing data to be volatile (e.g. new records gathered or existing records updated; service descriptions may be changed; new services added), while some subsets of the data could be fairly static (e.g. datasets stored at DBS do not change often). Further as the total data that is being queried could be fairly large (with growth of order of 1TB per year) and the data-services may have certain constraints (e.g. only certain queries allowed) as their main goal is supporting production (data taking from the CMS detector).

Therefore, it is important to balance between returning locally cached query results quickly and between issuing queries to slow data-services to get up to date results. It is also not less important to provide users with easy to use interface having a fairly flat learning curve, while at the same time not loosing the functionality of fairly complex queries. **Another issue is about handling distributed queries, where either selection criteria or selection values could come from distributed sources – the biggest problem could be in the APIs they [DO NOT] provide**.

3.1 Ease of use: Query Language over the mediated schema

Even a very simple structured query language that also contain entity names over the mediated schema may seem hard to learn, especially in the beginning. On the other hand, at CERN the names in the mediated schema are referring to real-world entities that are fairly consistently named (even though there may exist slight differences in their naming on different data-sources). So some sort of guiding helping user to build the query shall be useful.

Supporting non-structured keyword queries is also worth investigation as many users reporting they are missing this Google-like search experience. Because DAS uses Boolean IR we could only rank specific structured queries.

Further a minimum set of search predicates is imposed by APIs (mainly because of performance reasons) and user has to be at least informed what is he expected to provide.

Also to consider: input ambiguity and typos, some queries are more common than others.

3.2 Performance

- Query prioritization: maybe we could have prioritization in general, the heavier your query is the more you have to wait in favour of the light queries: (we could have some sort of query cost evaluation based on history or manually predefined scores per API)
- Explore more intelligent caching
- Could displaying Partial Results improve performance?
  - given current APIs to do aggregation over data sources, DAS has to fetch ALL records matching the query instead of only the first page.
  - e.g. intermediary aggregation results while still calculating
- scale testing - if we are storing lots of historical info. MongoDB is not so performant if DB cant fit in memory. One of well known solutions is installing SSD.

3.3 Handling distributed search efficiently

4 Proposed solutions

4.1 Ease of use

- Interactive guidance on how to compose queries to ease the learning (could also include some examples of popular queries):
    1. What entity are you searching for?
    2. How could you identify it (i.e. what do you know)?
    3. After seeing first results: What do you want to do with the results? (list items; select only specific columns; do aggregation; use in command line/another program)
- (?) JavaScript-based query interpreter to ease query writing: could suggest search attribute and entity attribute names. also taking into account some possible ambiguous namings (auto completion)
- Keyword query –¿ ranked list of matching structured queries:
    – map keywords to entity name and selection parameters (quite resembling 'Query forms' and 'Keyword cleaning' approaches)
    – Inverted index could be built with help of some good full-text search engine (e.g. Xapian[4]) and used to generate mapping to structured queries. If no match found (e.g. new entry not yet in our cache) some pattern matching could be used to try to corresponding guess search attribute
    – ranking could be based on: how closely keywords are matching some or all required parameters, popularity of certain query and users feedback
    – –¿ evaluate different approaches (HMM, etc); could some natural language processing help?
    – special attention to **wilcard** queries

4.2 Performance

4.2.1 Continuous view maintenance at Data providers with large DBs

Use either *materialized refresh fast views with query rewriting* (Oracle; completely transparent for proprietary apps)[ea11] or some other continuous view maintenance tool (e.g. DBToaster[5]) to improve performance of heavy queries containing joins and/or aggregations.

4.2.2 More intelligent caching

Some of the data entities instances changing very rarely, for example in DBS system old datasets would never change, while new ones are constantly added (still some of their attributes may change, in this example validity).

Cached copy may be shown before hand while up to date results could be retrieved on user's request.

An automatic change rate prediction could be useful to efficiently balance between caching and retrieving results.

Also consider:

- given an entity received from provider, determining if it is useful to cache for long-term
- could cache even *expired* data, but warning user
- could also have different validity dates for certain fields. if no volatile fields are not explicitly requested, even a very old cache could be used.
- Can we automatically figure which fields are static and which are changing?
- pre-fetching common (sub-)queries: determining manually and/or automatically
- **Deciding if to cache or not...**

[4]http://xapian.org/
[5]http://www.dbtoaster.org that is being developed at EPFL

4.3 Integrating distributed information efficiently

Bloom-join (which could be quite transparent and implemented even on DB side [pure SQL is possible for MySQL, to check for Oracle]- take a query and bit-vector as parameter) , lazy pagination (and order required for aggregation) - this is not yet supported by any of the data-service APIs

5 Preliminary Literature review

TODO: Overview; evaluate performance

5.1 Searching 'Deep Web' and Heterogeneous web services

In 1990s there were was much research on search based service integration systems, for instance *Information Manifold* following Local-as-View approach and *TSIMMIS* developed at Stanford following Global-as-View approach. The *Information Manifold* is quite resembling to DAS , so it could be useful checking it's papers. TODO: finish

Deep web search at Google

[multiple papers from Google] discusses various ways for implementing data integration in terms of large-scale search engine (Google) Virtual integration vs. surfacing. They also present ways for integrating systems without human intervention through use of statistical 'mediator'.

There two approaches to web scale search for deep-web (Google mostly cares about web forms):

Runtime query reformulation - 'leaves data at the sources and routes queries to appropriate services'[**?**, p. 1]

Deep-web surfacing - tries to add content from the deep-web into search index. There are algorithms which allow to iteratively choose input parameters to forms to surface most of the 'hidden' data without loosing much (i.e. if choosing parameters in not smart way the web form could yield as many results as a cross product of all input combinations).

PayGo approach - there is NO single mediated schema over which users pose queries. Queries are routed to to the relevant sources. Statistical methods used to model uncertainty at all levels: queries, mappings and underlying data.

Then there is no access to index data terms

TODO: Describe Keymantic[BDG+10]. This is a very suboptimal solution as indexing terms improves results (mention evaluation from SODA paper). It works only then keywords map entity names.

some hybrid approaches: index if exists regexp some string similarity measure based on historical data (e.g. even edit-distance would work for many items, like site)

5.2 Keyword search over DBs

The problem of Keyword search over Relational Databases (or also semi-structured sources like XML) has received a significant attention by the research community over the last decade.

The basic approach would first build an inverted index on database tables (usually only text columns). Then after finding all occurrences of the keywords, would try to construct join paths (based on Foreign keys) that would unite tuples containing the keywords.

A number of problem variations exist: returning only the ranked Top-k results vs. returning ranked list of possible queries, while some systems would even allow generating more complex queries including aggregations, etc (SQAK, SODA[BJK+12]).

Ranking Query Templates based on keyword query

A simple way to access relational database could be through a set of predefined named query templates (SQL with selection parameters or operators still to be specified) exposed to a user as a Form that the user has to fill in.

[CBC+09] proposes alternative approach for processing keyword queries over relational databases: given a keyword query, instead of returning database tuples one could rank query forms that best matches the query for user to choose the right one (if they are properly named this is fairly easy). The ranking is based on checking matching of keywords to table names in templates and to column values (could be implemented with inverted index).**TODO: more detailed and our limitations (after reading keyword cleaning)**.

An interesting feature of this approach is that a Query Template is functionally similar to any autonomous web service (which given the parameters would in turn execute that query on its database). In case of the Data Aggregation System, a user after entering a keyword query could be provided with a ranked list of structured queries (attribute=value) that could be processed given data source constraints (e.g. parameters required) and if needed he could refine his search (e.g. provide more parameters).

Keyword query cleaning

Keyword queries are often ambiguous, may contain misspellings or multiple keywords that refer to the same attribute value, therefore [PY08] suggested to perform query cleaning before proceeding to subsequent more computationally expensive steps (e.g. exploring all the possible join paths).

Further employing some machine learning method like HMM[Pu09] would allow to incorporate user's feedback (even the fact that user has chosen n-th result as a query to be executed is a good clue).

Meta-data approach

With a goal to bridge the increasing gap between high-level (conceptual, business) and low level (physical) representations of data, researchers from *ETHZ* have been investigating Generation of SQL for Business users at *Credit Suisse*. For converting natural language queries into SQL statements, in addition to what used by earlier approaches they used meta-data describing the schema (at multiple representation levels) and the domain (ontologies) and some natural language processing.

Even on a large data-warehouse of 220GB data with a complex schema of 400+ tables they reported that if good meta-data is available, generating even quite complex SQL (n-way joins with aggregations, etc) is quite feasible for computer. That would making it 'much easier for business users to interactively explore highly-complex data warehouses' [BJK$^+$12, p.932].

6 Work status

TODO

```
- obtained DB copy of biggest data provider DBS (currently 80 GB + 200 GB indexes)
- preliminary literature review (to be continued deeper + waiting for book arrival: Principles of

--- some analysis of DAS logs


Upcomming Work items:
- couple of fairly simple prototypes of UI/access patters for simpler DAS querying
- check on performance improvements after creation of materialized view(s)
- interview (more) DAS users
```

References

[BDG+10] Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Mirko Orsini, Raquel Trillo Lado, and Yannis Velegrakis. Keymantic: semantic keyword-based searching in data integration systems. *Proc. VLDB Endow.*, 3(1-2):1637–1640, September 2010.

[BJK+12] Lukas Blunschi, Claudio Jossen, Donald Kossmann, Magdalini Mori, and Kurt Stockinger. Soda: generating sql for business users. *Proc. VLDB Endow.*, 5(10):932–943, June 2012.

[BKEM11] G Ball, V Kuznetsov, D Evans, and S Metson. Data aggregation system - a system for information retrieval on demand over relational and non-relational distributed data sources. *Journal of Physics: Conference Series*, 331(4):042029, 2011.

[CBC+09] Eric Chu, Akanksha Baid, Xiaoyong Chai, AnHai Doan, and Jeffrey Naughton. Combining keyword search and forms for ad hoc querying of databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 349–360, New York, NY, USA, 2009. ACM.

[DGK+08] A Dolgert, L Gibbons, V Kuznetsov, C D Jones, and D Riley. A multi-dimensional view on information retrieval of cms data. *Journal of Physics: Conference Series*, 119(7):072013, 2008.

[ea11] P Lane et al. Oracle database data warehousing guide, 11g release 2 (11.2). chapter 9: Basic materialized views, September 2011.

[KEM10] Valentin Kuznetsov, Dave Evans, and Simon Metson. The cms data aggregation system. *Procedia Computer Science*, 1(1):1535 – 1543, 2010. ¡ce:title¿ICCS 2010¡/ce:title¿.

[KPGM00] Christoph Koch, Paolo Petta, Jean-Marie Le Goff, and Richard McCatchey. On information integration in large scientific collaborations, 2000.

[Pu09] Ken Q. Pu. Keyword query cleaning using hidden markov models. In *Proceedings of the First International Workshop on Keyword Search on Structured Data*, KEYS '09, pages 27–32, New York, NY, USA, 2009. ACM.

[PY08] Ken Q. Pu and Xiaohui Yu. Keyword query cleaning. *Proc. VLDB Endow.*, 1(1):909–920, August 2008.

A DAS Query logs

non parsible queries: 98923

Most common query patterns (Feb 2012-June 2012)

```
total valid queries: 569408

50.16% (285605) : dataset dataset.name=?
13.54% (77071)  : site dataset.name=?
8.96%  (51035)  : file dataset.name=?
5.88%  (33504)  : run dataset.name=?
2.59%  (14739)  : release dataset.name=?
2.11%  (12036)  : config dataset.name=?
1.65%  (9420)  : dataset run.run_number=?
1.34%  (7642)  : block dataset.name=?
1.24%  (7084)  : dataset site.name=?
1.15%  (6562)  : dataset dataset.name=? release.name=?
1.10%  (6287)  : parent dataset.name=?
0.96%  (5488)  : file file.name=?
0.92%  (5245)  : dataset dataset.name=? status.name=?
0.77%  (4363)  : dataset release.name=?
0.75%  (4257)  : file dataset.name=? run.run_number=?
0.68%  (3874)  : run run.run_number=?
0.53%  (2994)  : site site.name=?
0.45%  (2576)  : site file.name=?
0.45%  (2556)  : dataset file.name=?
0.43%  (2438)  : lumi file.name=?
0.40%  (2282)  : dataset dataset.name=? site.name=?
0.35%  (1999)  : file block.name=?
0.35%  (1970)  : dataset dataset.name=? run.run_number=?
0.29%  (1640)  : group dataset.name=?
0.29%  (1631)  : lumi run.run_number=?
0.25%  (1402)  : child dataset.name=?
0.22%  (1268)  : run file.name=?
0.21%  (1204)  : block block.name=?
0.19%  (1088)  : release release.name=?
0.16%  (936)  : site block.name=?
0.12%  (703)  : file dataset.name=? lumi.number=? run.run_number=?
0.10%  (594)  : parent file.name=?
0.08%  (455)  : dataset block.name=?
0.06%  (365)  : dataset dataset.name=? datatype.name=? release.name=?
0.06%  (360)  : file dataset.name=? site.name=?

Interesting non structured queries:
T!_CERN
*herwig*/AODSIM
```