Searching heterogeneous data-sources: Master thesis problem statement

Vidmantas Zemleris, October 11, 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. Various improvements including support for less restricted keyword queries, 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 attributes from 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 determines data-sources are able to answer[4], queries them, 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, sorting or aggregation (min, max, avg, sum, count, median), 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

[4]This is done by a mapping between flat mediated schema ('DAS keys') into web-service methods and their arguments. Then system queries all services that could provide a result of expected type with given parameters

3 Problem statement

3.1 Ease of use

For an IR system with wide variety of users, it important to provide an easy to use interface with fairly flat learning curve, while at the same time not loosing support of fairly complex queries. Even a simple structured query language containing entity names over the mediated schema may seem hard to learn[5], especially in the beginning. Therefore, it shall be useful to guide new users interactively through the process of building of their query. Supporting non-structured keyword queries is also worth investigation as quite many users reported they are missing this Google-like search experience.

3.2 Performance

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.

- Low data providers performance (e.g. on DBS some queries requires joining many large tables)
- At the moment all queries are put into one pool and has to wait until some threads are available. If a couple of heavy queries were submitted before hand, even very light queries would have to wait long. Explore more advanced Query prioritization (e.g. we could have some query cost model based on history or predefined scores per API)
- Currently result items are cached for a fairly short period of time (5min-1h) and then completely discarded, however many entities are not changing that often - 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. displaying intermediary results of aggregation while still calculating with statistical bounds – however statistical bounds are much meaningful only if items are well snuffled/randomized – people could stop the query if they see a non-sense

3.3 Other

Handling distributed search efficiently

People may be filtering data items where selection criteria could be present on two separate autonomous sources, meaning with the current APIs available the results would have to be fetched separately from each source and ones not contained in both filtered out only afterwards that is suboptimal.

Generic connector access relational databases

Generic connector able to access relational databases with minimal manual integration could be useful for integrating proprietary systems that were not yet integrated and there's no resources available to build APIs. A possible use-case could be the *Prep* database, but this still has to be discussed.

Depending on needs it could be either of exploratory or API/Query Forms based approach, where fake APIs are defined in terms of SQL queries (or even better in a simplified form that would generate SQL by taking schema into account)

[5] 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).

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. What do you want to do with these results (after seeing the results)? Possible options: list items; select only specific columns; do aggregation; use in command line/another program
- Given a keyword query, suggest the best matching structured queries (as a ranked list)[6]:
    - map keywords to entity names and selection attributes (quite resembling 'Query forms' and 'Keyword cleaning' approaches)
    - build an inverted index of attribute values with some good full-text search toolkit[7]. If no match found (e.g. new entry not yet in our cache) pattern matching could be used to make a best guess.
    - ranking could be based on: how closely keywords are matching (some or all) required API's input parameters, popularity of certain queries and users feedback
    - evaluate different user feedback approaches (e.g. HMM)
    - multi-wildcard queries (e.g. dataset=*Zmm*special*RECO*) are common at CMS that will need special attention[8].
    - making use of direct access to database (if available; in theory)
        * one could do incremental indexing of a target database [9], then try to map database table columns into API parameters (manually or even automatically by applying some of Data Integration approaches)
        * then this index + mapping could be used to improve the keyword search (better mapping from keywords to API inputs) or even structured search (e.g. query cleaning/term auto-completion)
        * problem: the owners of services may not want to provide direct access to DB because of no trust, security or performance issues
        * may need to exclude very large tables
- (?) 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)

4.2 Performance

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 by default 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.

[6]As DAS uses Boolean IR model we could only rank specific structured queries, but not the results itself.

[7]e.g. Xapian (http://xapian.org/) implemented in C++ which that is known for its flexibility, or Sphinx (http://sphinxsearch.com/)

[8]Matching to structured query will be harder and guessing even worse. Also for performance reasons some web-services will introduce limitations on the format of wilcard queries, while DAS wants still to be flexible at least for datasets

[9]E.g. an inverted index of values in Oracle/MySQL DB tables can be built to keep the list of possible terms. For instance, Sphinx full text search engine can access DBs directly and also supports incremental indexing but it needs a bit of manual configuration http://sphinxsearch.com/docs/current.html#delta-updates

On the other hand some result items may not make sense to be cached, as there may be too many of valid input parameter combinations (e.g. now there are 900M of *(run, file)* combinations).

Also consider:

- 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

Continuous view maintenance at Data providers with large DBs

In the cases when new records are coming not the existing ones are not changing much, continuous view maintenance that computes only differences from earlier results could be a good solution to improve the performance.

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[10]) to improve performance of heavy queries containing joins and/or aggregations.

For example currently in DBS (80GB + 280GB indexes) to process a query 'find files where run in [r1, r2, r3] and dataset=X' one needs to join all the tables below.

```
Dataset (164K rows) -> Block (2M) -> Files (31M) -> FileRunLumi (902M) <- Runs (65K)
```

However having a materialized view of all these tables joined together would allow answering such queries much quicker. Given low change rates (in comparison to data already present), maintaining the view should be also comparatively cheap with only expense of just couple of times of space.

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

[10]http://www.dbtoaster.org that is being developed at EPFL

5 Preliminary Literature review

5.1 Overview

The problem of keyword search over *relational and other structured databases* is widely covered, including even keyword search directly over distributed relational databases[AD12, ch.16] and some approaches for automatic schema matching has been discussed also. Such search is however very ambiguous and has quite large search space over the possible join paths, therefore most approaches rely on returning the top-k results.

Regarding the Boolean retrieval that we are the most interested, keyword query could be translated into list of best structured queries (e.g. SODA system proposing SQL over complex schemas, while Query Forms would propose mappings from keywords to parameters in SQL templates, Keymantic tries to achieve this without having access to Data itself).

Current research trends seem to fall on methods of automatic integration (e.g. statistical mediator), and pay-as-you-go integration. (TODO: see future chapter of [AD12])

5.2 Searching 'Deep Web' and web services

In 1990s there were was much research on search based service integration systems, for instance *Information Manifold* quite resembling DAS , so it could be useful checking it's papers..

No access to index data terms

[BDG+10, BDG+12] explores the case then there is no possibility to index the data terms, e.g. then a DB is behind a wrapper (e.g. accessible only through a *Web form* in Hidden Web or *a web-service*) then crawling is generally not possible.

In Keymantic[BDG+10] a 'keyword query is processed as follows: First, all keywords that correspond to metadata items (e.g., field names) are extracted. The remaining keywords are considered as possible input fields. Second, the likelihood of a remaining keyword to a metadata item is computed in order to rank different options to execute the keyword query on the Hidden Web database'[BJK+12, p.942].

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'[MJC+07, 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 a considerable part of the 'hidden' data without large overhead[11].

PayGo approach[MJC+07] - With this approach, 'a system starts with very few (or inaccurate) semantic mappings and these mappings are improved over time as deemed necessary'. there is NO single mediated schema over which users pose queries. Queries are routed to to the relevant sources with help statistical methods that are used to model uncertainty at all levels: queries, mappings and underlying data.

5.3 Keyword search over relational Database

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.

[11] i.e. if choosing parameters in not smart way, a web form with just a couple of free inputs (or even dropdowns that's easier case), could yield as many results as a cross product of all input combinations.

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).

SODA: 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 over a very complex data warehouse 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 both physical, conceptual and logical levels extended with DBpedia (for synonyms, etc) and domain ontologies (to capture business concepts like 'wealthy customer').

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 fairly complex SQL (e.g. n-way joins with aggregations) is quite feasible for a computer. That would make it 'much easier for business users to interactively explore highly-complex data warehouses' [BJK+12, p.932]. The users also reported system's potential a) for analysing the schema and learning patterns about it and b) as tool to help documenting legacy systems.

5.4 Keyword search: integration on demand

TODO?: [AD12, ch.16]

6 Work status

```
- obtained DB copy of biggest data provider DBS (currently 80 GB + 200 GB indexes)
- preliminary literature review (to be continued deeper)
- initial analysis of DAS logs

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

References

[AD12] Zachary Ives Anhai Doan, Alon Halevy. *Principles of data integration.* Number 9780124160446. Morgan Kaufmann, 2012. 497p.

[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.

[BDG$^+$12] Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Silvia Rota, Raquel Trillo Lado, and Yannis Velegrakis. Understanding the semantics of keyword queries on relational data without accessing the instance. In Roberto Virgilio, Francesco Guerra, Yannis Velegrakis, M. J. Carey, and S. Ceri, editors, *Semantic Search over the Web*, Data-Centric Systems and Applications, pages 131–158. Springer Berlin Heidelberg, 2012.

[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.

[MJC$^+$07] Jayant Madhavan, Shawn R. Jeffery, Shirley Cohen, Xin (luna Dong, David Ko, Cong Yu, Alon Halevy, and Google Inc. Web-scale data integration: You can only afford to pay as you go. 2007.

[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

Most common query patterns (from 2011-06-21 to 2012-10-01)

```
total valid queries: 569,408
not well formed queries (e.g. free text or with typing mistakes): 98,923

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-valid queries:
T!_CERN
*herwig*/AODSIM
--> keyword search
lumi file = (file dataset=/RelValProdTTbar/JobRobot-MC_42_V12_JobRobot-v1/GEN-SIM-RECO)
file,lumi dataset=/RelValProdTTbar/JobRobot-MC_42_V12_JobRobot-v1/GEN-SIM-RECO
--> Users may like more complex combined queries
file dataset=/MuEG/Run2011B-PromptReco-v1/AOD, file.size >1
file dataset=/MinimumBias/Run2010A-valskim-v6/RAW-RECO* | grep run between  [138923, 144086]
--> Users mixing up the post and pre filters
```

A Data providers statistics

```
(Some of the largest ones)

DBS:
    DB  size: 80GB + 200GB indexes, not many changes to existing  (old) records
    Largest tables:
      Dataset (164K rows) -> Block (2M) -> Files (31M) -> FileRunLumi (902M) <- Runs (65K)

Phedex: ~7GB, more often changes to existing (even old) records
    change rate: 2,359,934 file transfers last month (from site A to site B;
                 change rate on the DB to be found out)
```