Searching heterogeneous data-sources: Master thesis problem statement

Vidmantas Zemleris, October 19, 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 others[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. An enterprise information integration system based on simple structured queries is already in place. Various improvements including: adding support for more complex queries and for less restricted keyword search, 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* matching some pattern stored at a given *site* (filtering attributes from separate services)
- find (all) *datasets* related to some specific physics phenomena[3] together with conditions describing how this data was recorded by detector or simulated which are present in another autonomous system than the datasets (Physicists)

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

The Data Aggregation System

The *CMS 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]). The combined query results could be later 'piped' for further filtering, sorting or aggregation (min, max, avg, sum, count, median), e.g.:

```
# returns average and median dataset sizes of ones containing RelVal in their name having more than 1000 events.
dataset=*RelVal* | grep dataset.nevents >1000 | avg(dataset.size), median(dataset.size)
```

Queries are executed either from web browser or through a 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-orientedness 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 to extend 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 grid storage worldwide

[3] In case of dataset, this data is present in the *filename* or in the *run* entity

[4] This is done by a mapping between flat mediated schema entities ('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 User Interface and ease of use

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

3.2 Supporting complex queries

Currently DAS can only process queries that could be directly mapped to data providers' APIs, but not the queries where results of one API need to be fed into another APIs [6]. Further, to be flexible DAS only keeps a list of 'mediated entities' and how they could be retrieved, but do not enforce any predefined schema (i.e. entity fields/attributes) nor exposes it[7] before querying for some particular entity(-ies). This makes it fairly hard to execute complex queries without a priori knowledge. User has to know what queries and how have to be combined (depends on the mediated schema and what APIs are available), and has either to combine the results by hand or to write a program doing that. To be researched:

- how should these queries be entered by users?
- could we use current source descriptions that only define API result type and API parameters as entities of mediated schema to select meaningful and sufficiently efficient composition of services?

3.3 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 ones updated, service's schema changed. Still large amounts of the data are fairly static. As APIs of some data providers' are comparatively slow[8] it is important to balance between quickly returning items from local cache and getting fresh results. Another approach we are taking is spotting the performance issues at the services (e.g. at DBS some queries requires joining many very large tables, while most of their existing data stays static). Issues to be addressed includes:

- 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 earlier, even light queries would have to wait long. Explore more advanced Query prioritization (e.g. we could have a query cost model)
- 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
- Since DAS does aggregation across multiple data providers, given their current APIs (with no *ordering* and *paging* of results), DAS has to fetch ALL records matching the query instead of only the first page[9].
- Efficient distributed search: as filtering criteria may reside on two or more autonomous sources, given the current APIs much more items than in the final result may need to be fetched from each source. Items that do not meet all selection criteria could be filtered out only afterwards in DAS. To improve the performance, data provider's APIs for the most used queries may need to be redesigned.

3.4 Generic connector accessing relational databases

A generic connector accessing relational databases with minimal human effort could be useful for integrating proprietary systems that were not yet integrated when there's no resources to build the APIs. A possible use-case could be the *Prep* database, but this still has to be discussed. There was a project[10] at CERN which tried to generalize DBS-Query Language for any database that could be used as DB side mediator.

[5] Actually, the names in DAS mediated schema refer to entities fairly consistently named in the real-world.

[6] Except a couple of manually hard-coded "views" in a virtual *"combined"* data provider

[7] Actually the fields of each API (an entity could be mapped to a number of these) could be inferred from past queries or by predefined 'bootstrap' queries. Currently the field lists may differ slightly depending on the parameters

[8] This is partially because of large amounts of data stored at the providers (order of 1TB per year in total) and as their main goal is supporting data taking from the CMS detector their systems were not optimzed for exploratory queries

[9] Although it would be possible to show partial results for data coming from a single data-source (still loading in background, or the providers' APIs would have to be modified)

[10] `https://github.com/vkuznet/PyQueryBuilder`

4 Preliminary Literature review

4.1 Overview

Since the late 1990's, several *Enterprise Information Integration*[11] (EII) products have appeared in the market (e.g. *Information Manifold* by AT&T Lab) and an significant experience has been accumulated on data integration formalisms, ways of describing heterogeneous data sources and their abilities (e.g. RDBMS vs web form), query optimization (combining sources efficiently, source overlap, data quality, etc)[HAB+05]. Recent research in Enterprise Information Integration mostly focused on approaches minimizing human efforts on source integration, e.g. on uncertainty based self-improving systems[AD12, ch.19].

The problem of keyword search over structured sources received significant attention within the last decade. Keyword search over relational, XML and other databases was explored from a number of perspectives: returning top-k ranked result tuples vs suggesting structured queries as SQL, performance optimization, user feedback mechanisms, as well as keyword searching over distributed sources. If there is no need for 100% result exactness, keyword search combined with probabilistic schema matching provides lightweight exploratory data integration with almost no human effort upfront with ability to improve the results by adjusting to users' feedback[AD12, ch.16]. On the other hand, the *SODA* system has shown that if enough meta-data is in place, even quite complex queries given in bussiness terms could be answered over a large and complex warehouse.

In the setting of Boolean Keyword-based Retrieval over heterogeneous sources a keyword query could be translated into a ranked list of structured queries (similarly as *SODA* system that propose SQL, *Query Forms* approach that propose SQL templates or Keymantic trying to achieve this without accessing the data). In the extreme case of having no control over a web-service that do not publish its contents, techniques like Google's Deep-web surfacing could index a subset of its contents enabling keyword search to some extent.

4.2 Composing multiple data sources efficiently: Query translation using Logical Views

The *Information Manifold*[LRO96] is virtual integration (EII) system that represents both it's *queries* and *source descriptions* through a dialect of *description logics* (could be thought as *datalog*). It describes each source as a *logical view* over the global (mediated) schema, augmented with source capabilities (e.g. what are possible and the required input parameters for the source to return results[12]). This allows designing algorithms that could efficiently answer complex queries that require composing multiple the data sources that is done by finding maximally contained rewriting of the (conjunctive) input query in terms of logical views representing the sources (that is, finding an optimal way to compose the sources).

For example, consider such sources expressed as views (on the left) in terms of global predicates of the mediated schema (on the right) in datalog notation (based on [Ull00]):

```
v1(E, P, M) :- emp(E) & phone(E, P) & mgr(E, M). # employees, their phones and managers
v2(E, O, D) :- emp(E) & office(E, O) & dept(E, D). # offices and departments of employees
v3(E, P) :- emp(E) & phone(E, P) & dept(E, toy_dept). # phones of employees only in Toys dept.
```

Suppose we wanted to know Sally's phone and office. We express this again in datalog over global predicates:

```
q1(P,O) :- phone(sally,P) & office(sally,O).
```

There are two minimal solutions (as sources could be incomplete, the full solution is union of the two):

```
answer1(P,O) :- v1(sally,P,M) & v2(sally,O,D).
answer2(P,O) :- v3(sally,P) & v2(sally,O,D).
```

Notice that the expansions of these solutions (e.g. answer1_exp) are not equivalent to q_1, but only the conjunctive queries that are closest and still contained in q_1 (as they are the only usable views provided by the sources):

```
answer1_exp(P,O) :- emp(sally) & phone(sally,P) & mgr(sally,M) & emp(sally) & office(sally,O) & dept(sally,D).
```

After this the *Information Manifold* would find an executable order that adheres the capabilities of the sources, by iteratively considering any sources whose input parameters are satisfied.

[11] Enterprise Information Integration (EII) is about 'integrating data from multiple sources *without* having to to first load data into a central warehouse'[HAB+05, p.1]

[12] this makes these logical views quite similar to source APIs used by DAS, with difference that DAS currently only describes the parameters APIs and only partially the results

4.3 Keyword search over a Relational Database

The basic approach would first build an inverted index on values in database specifying where each comes from (usually only on text columns), as well as schema items (table and column names). Then, given a keyword query, the occurrences of the keywords would serve as entry points for the search, from where it would try to construct join paths (based on Foreign keys) that would unite tuples containing these keywords.

Ranking Query Templates based on keyword query

A simple way to access relational database could be through a predefined set of 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 to use this for answering keyword queries: instead of returning database tuples one could rank query forms and the user could choose the intended one (if they are properly named this is fairly easy). The ranking is based on how well the given keywords match table names and column values contained in each template.

Actually, a *Query Template* is functionally similar to any autonomous web service (which given the parameters would in turn execute that query on its database), with the only difference that access to potentially complex web service is often more expensive than to a database entry. In case of DAS, a user after providing a 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.

4.4 Keyword search over web services

What if there is 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 parameters to the input fields in the web form. Second, the likelihood of a remaining keyword to matching a metadata item is computed in order to rank different options of executing this keyword query on the "Hidden Web"[BJK+12, p.942].

5 Proposed solutions

5.1 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
- JavaScript-based query interpreter with auto completion to ease query writing: could suggest searchable entities and maybe even entity attribute names.
- Given a keyword query, suggest the best matching structured queries (as a ranked list)[13]:
  - the algorithm will generate mappings between each keyword to either an entity name, selection attribute name or its value (resembling 'Query forms' and 'Keyword cleaning' approaches)
  - ranking could be based on: how closely keywords are matching (some or all) required API's input parameters, popularity of certain queries, users' feedback and user's personal profile (e.g. search history, department, etc)
  - build an inverted index of attribute values with some good full-text search toolkit[14]. If no match found (e.g. new entry not yet in our cache) pattern matching could be used to make a best guess
  - the system could improve its results from users' feedback through machine learning (e.g. HMM)
  - multi-wildcard queries (e.g. dataset=*Zmm*special*RECO*) are common at CMS that will need special attention[15].
  - knowing all possible APIs' parameter values, in addition to historical values seen so far, would improve quality of the keyword search (better mapping from keywords to these values) or even the structured search e.g. through query cleaning & auto-completion. Not yet known if available[16].
    * having direct access to DB, one could incrementally index target database[17] tables semi-automatically mapping it's columns into API parameters.

5.2 Supporting complex queries

At the moment this still needs some further analysis/consideration, however it seems even the existing data-source definitions could be used for building more complex queries.

DAS Service definitions contain: for each data service (API), mapping of its parameters and result type into entities of mediated schema (but NOT ALL their fields are defined a priori). Even if DAS definitions do not describe every attribute of API result, as all input parameters are attributed to a mediated schema entity, these definitions seem mappable into *Information Manifold's* data source definitions described on Sec.4.2 of Literature review.

Other issues to consider:

- how these queries shall be entered by the user and how ambiguity shall be handled especially if queries entered in some simplified (semi-structured) language like DAS Query Language currently is
- any optimizations/heuristics if multiple possible solutions exist
- handling complex and heavy queries with large number of results, e.g. lazy loading with pagination (at least at the level of service composition as APIs do not yet support pagination/sorting)

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

[14] e.g. Xapian (http://xapian.org/) that is known for its flexibility, or Sphinx (http://sphinxsearch.com/)

[15] Matching a wildcard query into a schema field will be harder. Also for performance reasons some web-services will introduce limitations on the format of wilcard queries, while DAS wants to allow more flexible queries, at least for *datasets*

[16] problem: the owners of services may not want to provide direct access to DB because of no trust, security or performance issues. Otherwise, specific APIs could be developed to return possible parameter values but this is quite an overhead

[17] 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

5.3 Performance

More intelligent caching

Some of the data instances changes very rarely. For example in DBS system old datasets would never change, while new entities are constantly added (some of their attributes may change, but only rarely, e.g. validity of a dataset). 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. 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).

Other way to boost performance is pre-fetching most common queries from time to time (determined manually and/or automatically). Also it's possible have different validity dates for certain fields. if no volatile fields are not explicitly requested, even a very old cache could be used

Data providers' performance issues

In the cases when new records are coming but 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 especially for popular and heavy queries containing joins and/or aggregations.

This is exactly the case for the most popular expensive query (see query #7 in A.2 DAS log analysis) over DBS system (80GB + 280GB indexes): 'find files where run in [r1, r2, r3] and dataset=X' that needs to join all the biggest tables in that database.

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

Having a materialized view with 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 the only expense of just couple of times of storage space (storage is bound by the size of the largest table anyway).

For Oracle *materialized refresh fast views with query rewriting* would be completely transparent and would not need any changes to proprietary systems, but it has a couple of limitations on the queries[ea11]. Otherwise some other continuous view maintenance tool (e.g. DBToaster[18]) could be used, however this wouldn't be as transparent.

Integrating distributed information efficiently

Bloom-join (which could be quite transparent and implemented even on DB side (efficient pure SQL is doable for MySQL, and probably for Oracle too) quite almost transparently (a helper function could take the initial query and bloom-filter's bit-vector as parameters).

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

6 Work status

```
- set-up of development environment
- preliminary literature review (to be continued deeper)
- initial analysis of DAS logs
- obtained DB copy of biggest data provider DBS (currently 80 GB + 200 GB indexes)
- some work done on solving wildcard query restrictions

Upcomming Work items:
- benchmark data provider's performance (per API based on historical queries)
- couple of fairly simple prototypes of UI/access patters for simpler DAS querying
- check performance improvements after creation of materialized view(s)
- look into possible implementations for combined queries
```

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.

[HAB+05] Alon Y. Halevy, Naveen Ashish, Dina Bitton, Michael Carey, Denise Draper, Jeff Pollock, Arnon Rosenthal, and Vishal Sikka. Enterprise information integration: successes, challenges and controversies. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 778–787, New York, NY, USA, 2005. ACM.

[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.

[LRO96] A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. Technical Report 1996-61, Stanford InfoLab, 1996.

[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.

[Ull00] Jeffrey D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189 – 210, 2000.

A Project Milestones

Initially written up by Valentin (technical lead of the project from US):

```
Sep:
   - familiarize with techniques, research activities in the field
   - setup all tools and learn code to do actual work
   - describe and put on paper CMS use case
   - propose several approaches to target the project
Oct:
   - pick-up couple of approaches for development
   - develop initial prototype in high-level
      - outline workflow
   - math proof of choosen approaches
   - present results at EPFL & CMS DMWM team
Nov:
   - made a choice among target approach and start working on real prototype
   - it would be nice to see math foundations in place, including
     inital benchmarks, time estimates, stats, etc.
   - mid-term write-up document about current achievements,
     description of choosen approach and its implementation milestones
   - at this stage work with couple of data-providers (or their limited scope)
     to demonstrate capabilties of the chosen approach
   - I do not expect at this stage any optimizations, but would like to hear
     where those can be done, including strategies, technical improvements, etc.
   - do not limit yourself to CMS use case, explore possibility to apply
       choosen approach outside of CMS field
     - a toy prototype with non CMS data-providers is desired to have
           - demonstrate and discuss if it can be applied elsewhere
   - present results at EPFL & CMS DMWM team
   - Peter's action: based on the work presented, make a decision about conference
     submission (see below Jan milestone for approval)
Dec:
   - expand existing work from couple of data-providers into full set of CMS  data-services
   - work on integration with existing DAS code and prepare patch for  production deployment
   - plenty of testing, including but not limited to:
     - unit tests
     - integration tests
     - end-user UI studies
          - good to have prototype and give it to end-users
          - study and observe user behavior
   - present final strategy at EPFL & CMS DMWM team, including benchmarks, stats, user UI studies
Jan:
   - work on optimization (if required)
   - integration with DAS and production deployment
   - prepare main sections of write-up document
     - target approach & math foundation
     - prototype description
     - user studies, benchmarks, stats
   - submit write-up for EPFL & CMS review
     - if it is done, Peter will need to get some approval for conference presentation
Feb:
   - final document in place describing work, achievements, results
   - make conclusion about contribution to the field
     - discuss innovation work
     - discuss engineering work
     - discuss CMS use case and achievements
     - discuss possible application outside the field
Mar 15:
    - thesis delivery
Apr:
   - defense
```

A DAS Query logs (from 2011-06-21 to 2012-10-01)

From the logs it can be seen that queries requiring heavy joins are quite common (100K queries through command line interface) that makes it worth investigating the possible performance optimizations.

A.1 Common query patterns through Web browser

total valid queries: 569,408
not well formed queries (e.g. free text, typing mistakes, spam): 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=?
```

```
Interesting non-valid queries:
* keyword search: *herwig*/AODSIM
* Users may like more complex combined queries:
    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 mixing up the post and pre filters:
    file dataset=/MuEG/Run2011B-PromptReco-v1/AOD, file.size >1
    file dataset=/MinimumBias/Run2010A-valskim-v6/RAW-RECO* | grep run between  [138923, 144086]
```

A.2 Common query patterns through Command Line

total valid queries: 6,9M
non valid queries: 76K

```
39.56% (2735728):       dataset dataset.name=?
22.80% (1576886):       dataset dataset.name=? status.name=?
12.49% (863636) :       file dataset.name=?
12.44% (860626) :       run run.run_number=?
4.60%  (318248) :       site dataset.name=?
2.18%  (150845) :       run dataset.name=?
1.71%  (118404) :       file dataset.name=? run.run_number=?
1.32%  (90988)  :       file block.name=?
1.26%  (87266)  :       file file.name=?
0.50%  (34482)  :       block site.name=?
0.35%  (24049)  :       lumi file.name=?
0.25%  (17282)  :       release dataset.name=?
0.18%  (12556)  :       parent file.name=?
0.09%  (6341)   :       file dataset.name=? lumi.number=? run.run_number=?
0.06%  (4306)   :       file dataset.name=? site.name=?
0.05%  (3547)   :       dataset dataset.name=? primary_dataset.name=? release.name=? tier.name=?
0.05%  (3352)   :       dataset file.name=?
0.01%  (996)    :       parent dataset.name=?
0.01%  (755)    :       dataset site.name=?
```

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)
```

Not so relevant from Literature Review: Deep web search at Google

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

Deep-web surfacing - surface deep-web (e.g. web forms) adding their results into the standard search index easily allowing to using existing IR technology that scales well. There exist algorithms which allow to iteratively choose input parameters to the forms to surface a considerable part of the 'hidden' data without large overhead[19].

Pay-as-you-go 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 the relevant sources with help statistical methods that are used to model uncertainty at all levels: queries, mappings and underlying data.

[19] 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.