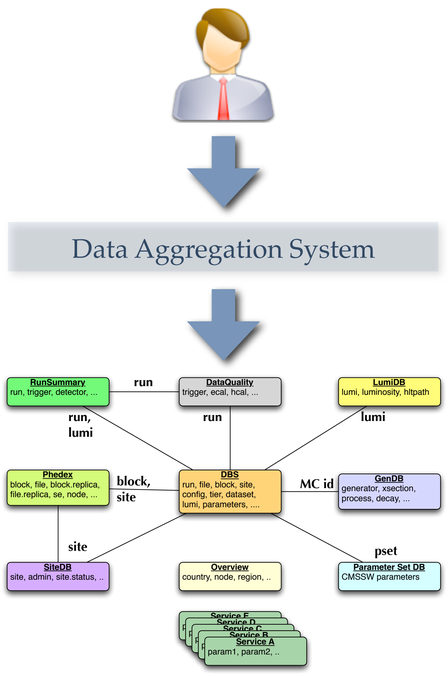
Data Aggregation System, an information retrieval on demand over relational and non-relational distributed data sources

Valentin Kuznetsov (Cornell, USA), Dave Evans (FNAL, USA), Simon Metson (Bristol, UK), Gordon Ball (Imperial, UK)

Motivations ...

- A user wants to query different data services without knowing of their existence
- A user wants to aggregate information from different data services
- A user has domain knowledge, but needs to query X services, using Y interfaces and deals with Z data formats to get the data



DAS in nutshell

DAS provides a novel approach to aggregating data from multiple sources without applying any requirements on data providers. Once data is accessible on a web DAS can handle the rest.

DAS leaves data management up to the data-providers. It is true that they know better how to handle, preserve and secure their data.

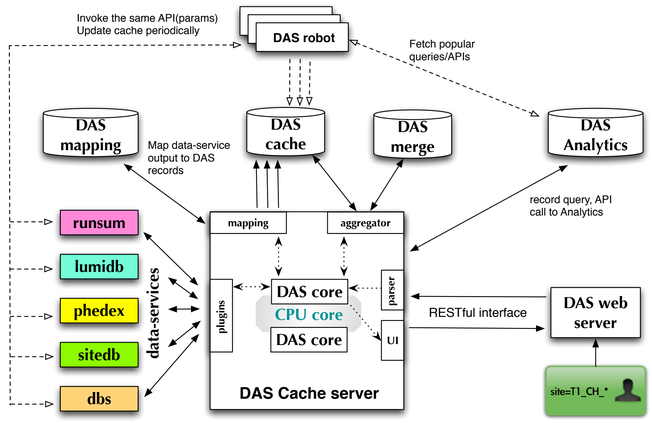
DAS is agnostic to data content. Thanks to NoSQL document-based database MongoDB we're able to store any type of meta-data documents provided by data-providers.

DAS provides a free text-based query language to ease data-lookup. It should be as simple as you search on Google.

DAS uses filters and aggregators to help you navigate through your data.

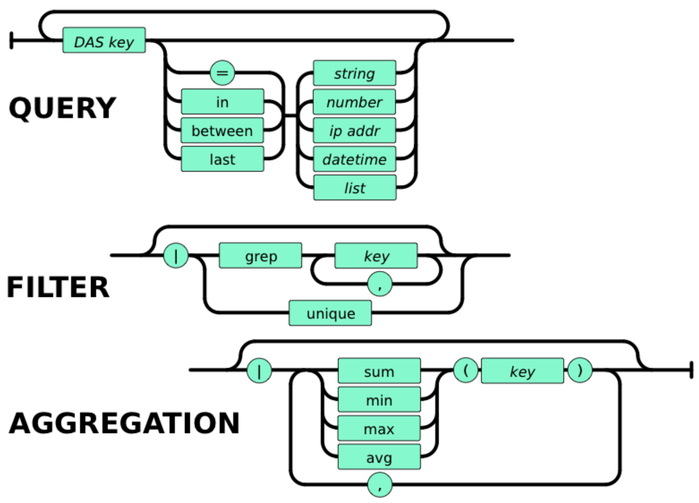
DAS has been developed in CMS to deal with broad variety of existing distributed data services, majority of them are RDMS based.

DAS architecture



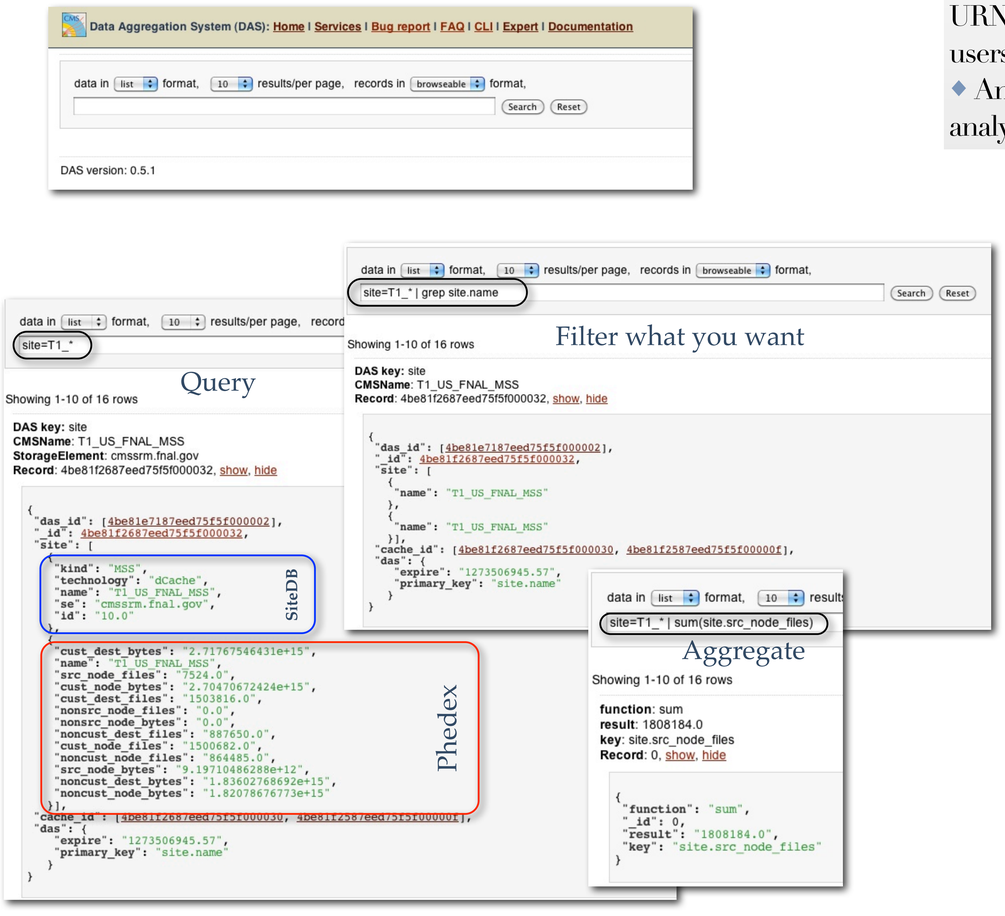
- Web server/CLI tool to communicate w/ end-users
- Cache server to handle requests flow
- Cache DB to store results from data-providers
- Merge DB to store aggregated results from DAS cache
- Mapping DB to keep info about data-providers (URIs, URNs, expire timestamps) and DAS keys used by end-users
- Analytics DB to keep track of user requests and query analysis

DAS Query Language



- <expressions> | <filters> | <aggregators> or <map-reduce functions>
  - <expressions> represented in a form of <*key*> and/or <*key*> <*operator*> <*value*>
  - **keys:** *dataset, block, file, ...*; **operators:** *=, in, between, last*; **values:** *int or string (including patterns)*; **filters:** *grep, unique*, **aggregators:** *sum, count, avg, min, max*
- Examples
  - *site=T1_CH_CERN; site=T1_* | grep site.name*
  - *run=20853; run in [20853,20859]; run between [20853,20859]; run last 24h*
  - *block dataset=/a/b* | grep block.size | sum(block.size)*

DAS user interfaces



To simplify data look-up we used

- Presentation maps for data records, which shows a snapshot of data content, e.g. for site record we only show site name and SE info
- Filters in form of standard UNIX pipes, which select a sub-set of data record
- Aggregators, e.g. sum, count, which allow get snapshots of data
- Map-reduce functions for more sophisticated data analysis

Data from data-providers were converted into common JSON data-format

Data notations has been centralized across multiple data-services by using DAS maps, e.g. runNumber, Run were converted into run_number notations.

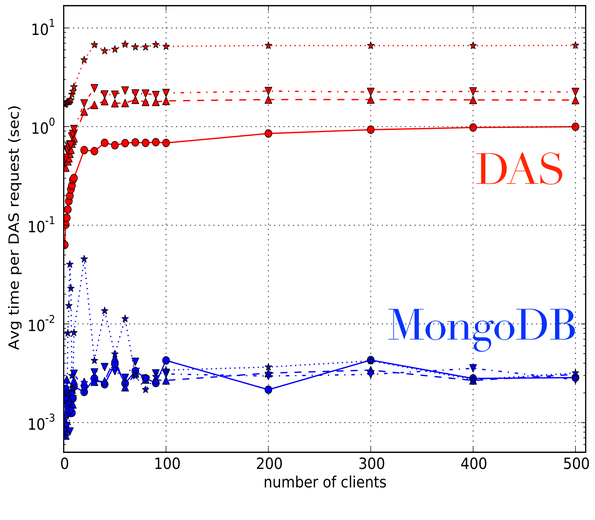
DAS performance

DAS benchmarks has been done on Linux 64-bit 8 core node w/ 16GB of RAM.

We measured separately write and read time for different set of meta-data.

Write time has been driven by data-providers. Due to the fact that data need to be shipped over the wire from remote source to DAS we only measured internal insert rates. The MongoDB back-end can handle up-to 20K docs/sec, while DAS merge time was around 5K doc/sec.

The read time benchmark has been done when DAS was populated with 50K dataset and 500K block meta-data information. We used query patterns, e.g. dataset=/A* and measured server response time to find a first matched record.



Access random record using

- ★ 500M blocks, 100x of projected statistics
- ▼ 50M blocks, 10x of projected statistics
- ▲ 500K blocks, current # of CMS blocks
- ● 50K datasets, current # of CMS datasets