

## FIFE Architecture Committee

### Databases

Draft by Igor Mandrichenko, version 1, 6/11/2013

### Challenges and Goals

Typically, databases are used to store calibration or conditions data. These data are necessary for data processing and analysis, vast majority of which is done in batch environment, often on the grid. Typical data processing job retrieves calibration or conditions data related to the event or a group of events it is processing.

There are 3 major challenges presented by HEP and Astrophysics data processing and analysis in the area of databases:

- Data access performance – database access must be robust and provide low latency in data delivery so that the (otherwise CPU-bound) batch job can utilize grid resources in most efficient way. In other words, time spent by the job waiting for the data to be delivered must be significantly less than time spent performing data processing and calculations.
- Resource management – the whole point of using grid resources for data processing and analysis is to run as many data processing jobs as possible in parallel. In this case any resource, which becomes a bottleneck slows down the data processing and on the other hand, any resource must be able to sustain large numbers of simultaneous requests.
- Remote data delivery – in the grid or cloud environment, often, data processing is performed at remote sites, connected to the database location via WAN. Data access via WAN can be more challenging because the WAN may not be able to provide as large throughput as LAN. However, conditions and calibration data tend to be much smaller than actual event data, and therefore low WAN throughput does not always create significant problems.

Obviously these challenges are very tightly coupled. If individual data access transaction is very short in time, or a typical job accesses the database only few times during its life, that creates less load on the database and its interface. In other words, number of simultaneously processed data requests by the database interface is:

$$\text{Load} = N_{\text{jobs}} * N_{\text{access}} * (t / T) \quad (1)$$

Where  $N_{\text{jobs}}$  is number of jobs running simultaneously,  $N_{\text{access}}$  is number of times each job accesses the database during its life,  $t$  is duration of each individual data transaction and  $T$  is job lifetime. So by reducing  $N_{\text{access}}$  and or  $t$ , one can allow more simultaneously running jobs without increasing the load on the server. In the same

time, data access performance requirement means that  $N_{\text{access}} * t/T$  ratio should be minimized. So obviously minimization of  $N_{\text{access}} * t/T$  is the ultimate goal in designing the database access system for batch data processing.

### Web Services

Web services or their predecessors have been used by HEP experiments as a convenient and flexible tool to build database interfaces since more than 10 years ago. D0 has been using CORBA-based calibration database servers and CDF uses HTTP-based Frontier. Modern trend in the industry is to use simple REST (representational state transfer) style of HTTP-based services. Using REST with HTTP has many advantages:

- HTTP is a universally accepted standard used by Internet, developed and supported by W3C and IETF.
- HTTP is extremely simple protocol
- There are numerous libraries implementing the protocol available for all popular programming languages
- HTTP can be used to transfer data represented in practically any format, from plain text to encrypted compressed binary data
- There are such well developed and supported frameworks as Apache httpd, Tomcat, which make publishing an application as a web service very easy

Other important advantages of using a REST web service as an interface to the database are:

- Web service can make implementation of the client completely independent of the database implementation. The client can be implemented in terms of the application abstractions and data representation and the web service will translate the application specific representation to the database representation and back
- Web services technology is very well developed in the area of resource management. Frameworks like Apache httpd make it very easy to control and manage large number of incoming data requests, whereas databases themselves tend to be much more easy to overload.

### Data Caching

In certain cases when data processing is done remotely, it may be beneficial to consider using local data caching. Data caching can be used to increase data transfer efficiency under 2 conditions:

- Data is cacheable – same request is expected to produce same results. This is often the case, but not always.
- Requests are correlated – there is significant probability that 2 batch jobs running (almost) in the same time will request same, cacheable data.

These two requirements can be combined into one formula:

$$T_{\text{life}} > T_{\text{hit}} \quad (2)$$

Here,  $T_{\text{life}}$  is average lifetime of the cached data. Data lifetime is determined by several factors:

- Validity of data – sometimes data get overridden, e.g. when calibration constants are updated due to recalibration
- Cache preemption – new data requests preempt data left in the cache by previous requests

$T_{\text{hit}}$  is average time between requests for the same data. This quantity depends on the pattern of data processing. Unfortunately, typical data processing pattern (as opposed to data analysis) may not always lead to repeating requests for calibration or conditions data, because batch jobs cover their own data validity time intervals and never share calibration or conditions data.

There are 2 approaches to caching data delivered via web services:

- Application specific – data can be cached on by the web server internally so that not every request to the web service causes a database transaction; also data can be cached privately by the client application
- Application independent – data are cached using standard HTTP caching proxy mechanism

## Recommendations

We recommend using web services as a general approach to building application interfaces to databases.

Unless it is required that the client application must be aware of the implementation details of the database (platform used, database structure, schema) we recommend to implement web service interface in terms of application specific data representation rather than database representation such as SQL.

In case it is required that the client application is aware of the database implementation details, CMS-style Frontier should be considered.

When possible and beneficial, use of caching should be considered. Application independent HTTP caching can be performed by product called Squid. In cases when there are no benefits in using caching, i.e. when (2) is not satisfied, cache should not be used, because it increases communication latencies, complexity of the application and support cost.