Requirements for Scalable Conditions Data Delivery

Lee Lueking, Igor Mandrichenko

September 16, 2010

Conditions Data Description

Conditions data is an umbrella term that refers to information that describes detector and beam conditions. It is generally valid for detector data taken during specific periods of time, sometimes referred to as Intervals Of Validity (IOV). It is a type of meta-data necessary to make sense of the detector data, and it includes calibration, alignment, attenuation, pedestal, etc. for detector channels, as well as information about the intensity and characteristics of the beam.

Some of this information is required for processing and analysis of detector data and thus access is required by many clients running simultaneously on interactive and GRID resources. Much of this data is stored in central databases or files, and approaches to scale the delivery to thousands of clients are needed.

Requirements

Following are parameters that define the problem. Typical values need to be obtained form experiments and/or estimated.

1. Expected request rate
    a. Peak
    b. Average
2. Data unit size
3. Latency requirements
4. Accepted failure rate
5. Some estimate of time correlation between requests
6. Boundary conditions
    a. hardware to be used
    b. network bandwidth available
    c. technologies to use and not to use

Example Solutions and Constraints

Two classes of solution are possible, 1) scaling up the database service, and 2) providing additional caching tiers between the database server and the client. The first option is easiest as it preserves the original interfaces used by each experiment; however it is not always feasible or practical. In the environment of the Intensity Frontier there are many database technologies being used and each experiment has chosen the solution that works best within its development environment and is supported by its software framework. Among the solutions being employed are Oracle, MySQL, PostgresSQL, SQLite and file-based options such as ROOT or ASCII formats . Simply scaling up the central service is limited by server hardware and network constraints. In some cases scrutinizing the way each experiment stores and retrieves information to and from the database can also provide significant improvements in performance.

By including additional caching layers, or tiers, lightweight, high performance components can be deployed. In many cases this can be done close to where the clients are running and significantly improve throughput while maintaining low central server and network loads. These caching layers can be in the form of files delivered to the processing site, or some form of proxy/caching server deployed at or near processing centers. For conditions data that is more-or-less static, distributing files with the detector data being processed is a satisfactory solution. If the conditions information is changing then delivering files can be problematic and requires a method of indexing them to ensure that the proper info is provided and used for specific processing. Exporting conditions data from the central database to SQLite can provide a convenient method to capture the relational nature of the data in a transportable static file. Some of the frameworks used by experiments already support SQLite and it is therefore an attractive solution.

For conditions data that is changing proxy/caching servers represent a good solution. This is typically done with SQUID and this has been shown to be reliable and versatile. The difficult part is incorporating the layer into the experiments software stack. There are two major requirements

1) IOV table: the requests from the client must refer to an IOV tag which is loaded into the client initially, before any requests for conditions data. In the simplest cases, experiments will request conditions data based on an event time, say the time for the first event in a file. This will not work since files with different starting events will not be able to take advantage of cached data since, to the cache, the requests all appear different.
2) API: Each experiments database access needs to be adapted to a single "standard" API that can be shared by everyone.
3) Cache coherency policy: In a cached system, the cache can be stale and the refresh policy must be understood. Several techniques have been developed to mitigate potential issues in this area. These need to be clearly understood and appropriately implemented.
4) Deployment: A central service that has connection pooling to the central database is required. This service must be able to translate the HTTP: (or other protocol) from the client into a SQL command appropriate for the underlying central database service. It must also encode the database response and send it back to the client.

A desirable is a mechanism that allows monitoring and recording the requests that are being sent by the clients. This adds to the understanding of who, where, and what kinds of requests are being made. Having a middle tier reduces the need for some of this monitoring since the chance for bottlenecks is greatly reduced and any serious contention at the central service can be traced by looking at the access logs.