Conditions DB Scalable Access

Lee Lueking/Igor Mandrichenko

November 17, 2010

Overview

- “Conditions Data” is an umbrella term referring to information describing detector and beam conditions.
- Examples
  - calibration, alignment, attenuation, pedestal, etc. for detector channels,
  - information about the intensity and characteristics of the beam.
- Valid for specific periods of time, referred to as Intervals Of Validity (IOV)
- 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 required.

Requirements

- Following are parameters that define the problem. Typical values need to be obtained from experiments and/or estimated.
- Expected request rate
  - Peak
  - Average
- Data unit size
- Latency requirements
- Accepted failure rate
- Some estimate of time correlation between requests
- Boundary conditions like
  - hardware to be used
  - network bandwidth available
  - technologies to use or not to use

MINERvA (typical)

Description	Estimate	Comment
Job Duration	1 hour	
Number of simultaneous Running Jobs	100	
Events per job	1000	
Internal job cache hit ratio	100%	Most events processed by a job use the same conditions data set
Size of conditions data set	3 MB (uncompressed, binary)	

And now, some assumptions…

- Number of requests processed simultaneously without significant loss of scalability: 5
- Peak job start rate = 10 times average job start rate
- Allowed latency = job duration/10 = 6 min

MINERvA with assumptions

Description	Average	Peak	Comments
Job start rate	2/minute	20/minute	N running jobs/job duration
Conditions data requests rate	2/minute	20/minute	assuming 100% internal job cache hit ratio = job start rate
Network bandwidth	100KB/sec (3MB * 2/minute)	1MB/sec (3MB * 20/minute)	
Disk throughput	100KB/sec	1MB/sec	
Time to retrieve and deliver 1 data set to sustain the request rate	30 sec	3 sec	Single threaded DB server
	150 sec	15 sec	Five threaded DB server

ID Scheme and Version Control

- Need mechanism so conditions data can be managed as "*sets*", valid for a given time interval (A.K.A. *IOV*).
- Changes to the *conditions set* need a mechanism for "*tagging*" them so reproducibility in processing can be insured.
- Requests from a client must refer to the *set*, *IOV* and *tag*, or some similar unique identification scheme.
- What to avoid
  - Clients request conditions data based on an event time, say the time for the first event in a file.
  - Cached data is not used effectively since the requests for the same data all appear different.

Central vs. Distributed

- Central database service sized to meet peak demand, or
  - Simple (especially since each experiment has unique solutions)
  - Not always feasible or practical.
  - Limited by server hardware and network constraints.
- Providing additional caching tiers between the database server and the client.
  - lightweight components can be deployed to unload the demand on the central database service and provide additional reliability.
  - This can be done close to where the clients are running and significantly improve throughput while maintaining low central server and network loads.

Caching Layer Options (1/2)

- Database replicas,
  - Some database technologies provide replication software that can make this fairly straightforward.
  - A read-only replica is practical to set up and support
  - Difficult to support beyond central site.
- Files delivered to the processing site
  - Static data can be delivered w/ software, or some other mechanism
  - SQLite files maintain relational aspects of data and are convenient to use.

Caching Layer Options (2/2)

- http proxy/caching servers (Typically SQUID)
  - If the client requests are done properly, and the SQUID cache can be used effectively, the performance achieved with modest hardware can be extremely high.
  - Redundancy is also easy to design into the system so high reliability can also be achieved.
  - Most OSG GRID facilities have SQUID services already in place providing a standard infrastructure near the processing client.
  - Requires central "translation" service.

Cache Coherency

- 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 for any type of caching system

Reliability

- Uptime must be very high
- Failover mechanisms must be transparent and "intelligent" (client knows when to wait, or give up and return an error).
- Redundancy where possible makes the system more scalable and avoids emergency (i.e. off hours) intervention.

Monitoring

- A way to monitor and record the requests that are being sent by the clients is very useful
- This adds to the understanding of who made the request, what kinds of requests are being made and where they are coming from.
- In a distributed system, at the access logs at each level are useful, although sometimes difficult to compile into a comprehensive picture.

Short-term

- For the most part, each experiment has chosen a different approach to managing their conditions data information.
- Solutions like replicated databases may make it possible to scale to increasing numbers of clients under such an environment.
- Some frameworks, specifically GAUDI/COOL//CORAL, support multiple technologies including Oracle, SQLite, MySQL (deprecated in recent versions) and FroNTier.
- This provides some flexibility as to the choice of solution, but nevertheless requires effort to set up infrastructure and understand feature sets that are, sometimes, not well documented.
- Other frameworks, like FMWK, are more specific to particular solutions, PostgresSQL in this case. i

Long-term

- In the long term, providing common solutions would simplify deployment and support.
- Common API's would make documentation and user buy-in straightforward.
- In some cases shared repositories may be possible
- Features such as monitoring could also be uniform and shared.

Conclusions

- An initial look at MINERvA access patterns give a rough idea of "typical" performance requirements.
- Having a consistent scheme for identifying conditions data sets and IOVs, with proper version management, is essential.
- Reliability is also an important requirement for such a service.
- Several approaches for deploying a system to meet the performance and reliability requirements are possible.
- Short-term working with existing systems to improve performance may be possible.
- Long-term, using common solutions for client API, middle tiers, and monitoring is a target.

finis