Greatly improved cache update times for conditions data with Frontier/Squid

Dave Dykstra and Lee Lueking
Computing Division, Fermilab, Batavia, IL, USA

E-mail: dwd@fnal.gov

Abstract. The CMS detector project loads copies of conditions data to over 100,000 computer cores worldwide by using a software subsystem called Frontier. This subsystem translates database queries into HTTP, looks up the results in a central database at CERN, and caches the results in an industry-standard HTTP proxy/caching server called Squid. One of the most challenging aspects of any cache system is coherency, that is, ensuring that changes made to the underlying data get propagated out to all clients in a timely manner. Recently, the Frontier system was enhanced to drastically reduce the time for changes to be propagated everywhere without heavily loading servers. The propagation time is now as low as 15 minutes for some kinds of data and no more than 60 minutes for the rest of the data. This was accomplished by taking advantage of an HTTP and Squid feature called **If-Modified-Sinc**e. In order to use this feature, the Frontier server sends a **Last-Modified** timestamp, but since modification times are not normally tracked by Oracle databases, a PL/SQL program was developed to track the modification times of database tables. We discuss the details of this caching scheme and the obstacles overcome including database and Squid bugs.

1. Introduction

The Compact Muon Solenoid (CMS) detector project of the Large Hadron Collider (LHC) generates too much particle collision data to be processed all at one site, so the processing is distributed around the world. All of the over 100,000 computer cores at approximately 80 worldwide sites need access to data describing the alignments and calibrations of the detector at the time of the collisions. This data is known as “conditions data.” Very many cores need to access the same data, so it is well-suited for caching. The amount of the data varies but is on the order of 100 Megabytes per processing job.

The caching subsystem that CMS uses for conditions data is called Frontier [1]. Frontier translates database queries into HTTP, looks up the results in a central database at CERN, and caches the results in an industry-standard HTTP proxy/caching server called Squid. One of the most challenging aspects of any cache system is coherency, that is, ensuring that changes made to the underlying data get propagated out to all clients in a timely manner. Recently, the Frontier system was enhanced to drastically reduce the time for changes to be propagated everywhere without heavily loading servers. This paper discusses the details of how this was accomplished.

2. Background: Frontier usage in CMS

The usage of Frontier in CMS is depicted in figure 1.

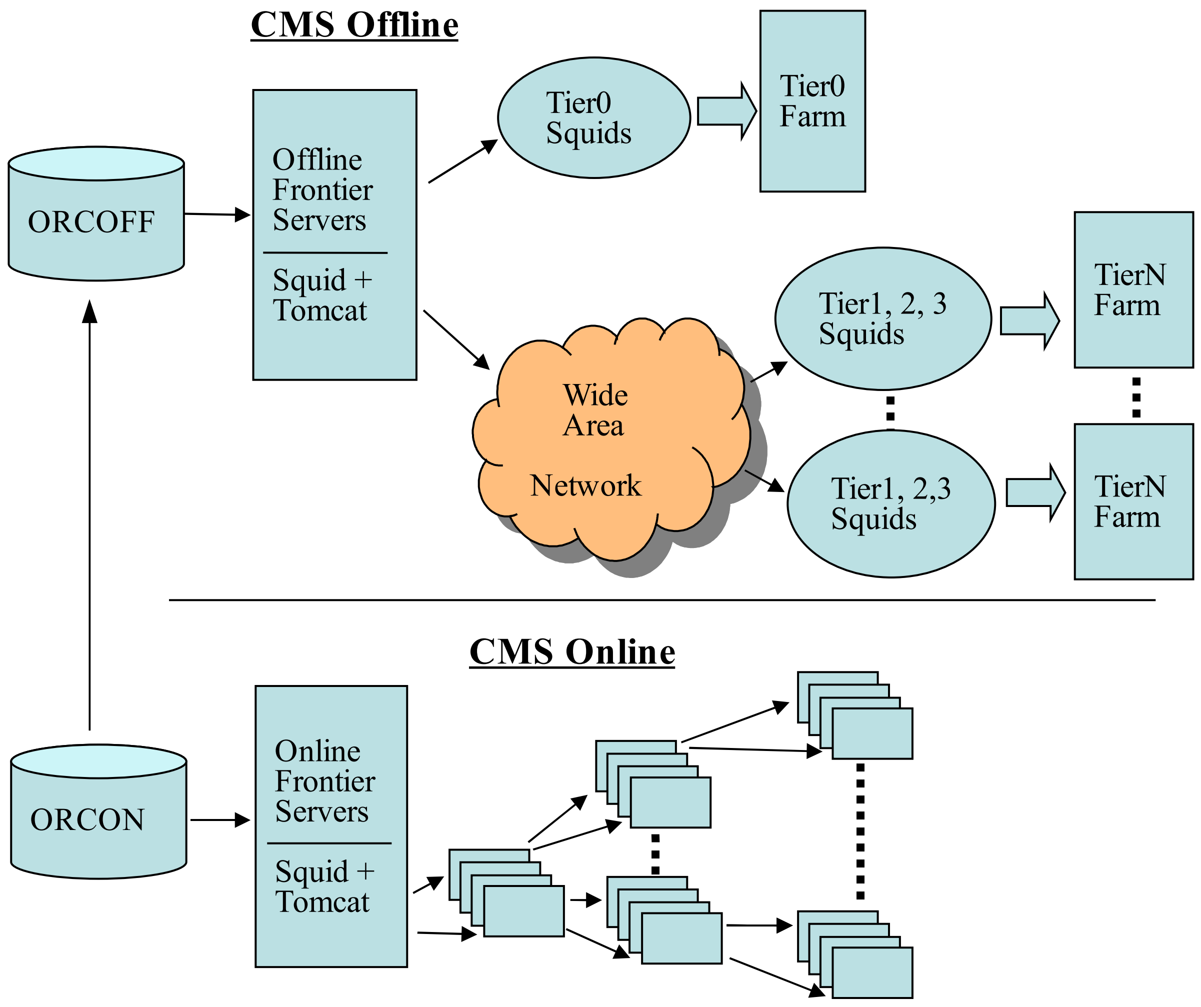


Figure 1. How Frontier is used in CMS

CMS software running on each core of the worker node farms worldwide include a Frontier client library that converts SQL database queries to HTTP. Each Offline site that processes collision events has one or more Squids to cache local copies of the conditions data that they retrieve from the central Offline Frontier servers at CERN. Those central servers each also have their own Squid for caching plus a Frontier Servlet running under Tomcat that converts the HTTP requests to Oracle database requests and returns the results back over HTTP.

CMS conditions data is first loaded into the Online Oracle database server (a.k.a. ORCON) which is co-located onsite with the CMS detector, then replicated to the Offline Oracle database server (a.k.a. ORCOFF) at the main CERN site. Over 10,000 computer cores also co-located with the detector need simultaneous access to that data to assist in Online real-time filtering of collision events. In order to have simultaneous access, there is a Squid on every node and they are arranged in a hierarchy where many nodes feed four others. This eliminates the network as a limiting factor on the speed of loading the conditions data and enables the data to be loaded to all cores in less than one minute.

3. The problem: cache coherency

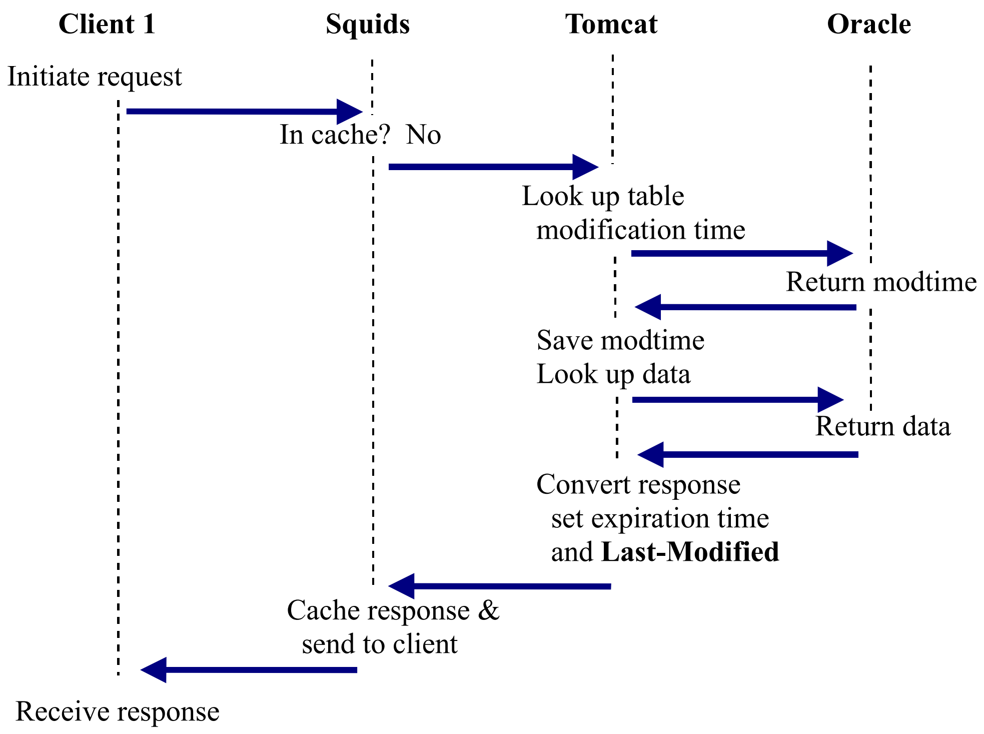
Every caching system must deal with cache coherency, that is, it has to be able to track changes in the underlying data. Previously, this was managed in Frontier by separating queries into two types: (1) those that were expected to change and (2) those that weren't. Those that were expected to change were set to expire sooner than those that weren't. In CMS Offline, even the short times were still quite high, however: most were once per day. The long times were very high: a year. Setting the times much shorter would result in much higher delays and stress on the infrastructure, because the data had to be reloaded all the way from the database, even if it hadn't changed.

The problem with this previous approach was that there were often times when data would change in the database even though it wasn't supposed to. There were many occasions where caches had to be manually cleared.

In CMS Online, everything was set to be expired very frequently so there wasn't a coherency problem. However, all conditions data had to be reloaded every time the filtering was started. That took less than a minute, but during that time all collisions at the very expensive detector are lost, so that is still too long: it was required to be less than 10 seconds. Fortunately, the new solution (described below) re-validates expired cached items so unless the data changes, it doesn't need to be completely reloaded and so is much faster.

4. The solution: If-Modified-Since

The HTTP protocol [2] and Squid already had a solution to the cache coherency problem: if a server supplies a **Last-Modified** timestamp in the HTTP header of a response, and a query comes to Squid for an object in its cache that has expired, Squid automatically adds an **If-Modified-Since** header to the request including that timestamp. The server may then compare the cached timestamp to a timestamp currently associated with the data, and if nothing has changed it can send a very small **NOT MODIFIED** response rather than reloading all of the data. Figures 2 and 3 show an example of this. Figure 2 shows the steps to fill the cache and figure 3 shows the steps after the response has expired.



Client 1
Squids
Tomcat
Oracle
Initiate request
In cache? No
Look up table modification time
Return modtime
Save modtime
Look up data
Return data
Convert response set expiration time and **Last-Modified**
Cache response & send to client
Receive response

Figure 2: An example of filling data into the cache

In figure 2, a client initiates an SQL request, converts it to HTTP, and sends the request to its local Squid. The Squid finds that the result is not yet in the cache, so it forwards the request on to the central Squid (the second Squid is not shown separately in the figure since both Squids do the same thing.) When the request reaches the Frontier Tomcat, the servlet converts the request back to SQL and parses the SQL to find the database table name that is being queried. In this example it has not recently looked up the timestamp for the table so it asks Oracle for the table modification timestamp. Oracle returns the timestamp, and Tomcat saves it and then right away asks Oracle for the data. After Oracle returns the data, Tomcat converts it into an HTTP response and includes in the response the timestamp in a **Last-Modified** header and also includes a header indicating the cache expiration time for the response. The Squids cache the response and sends it back to the client.

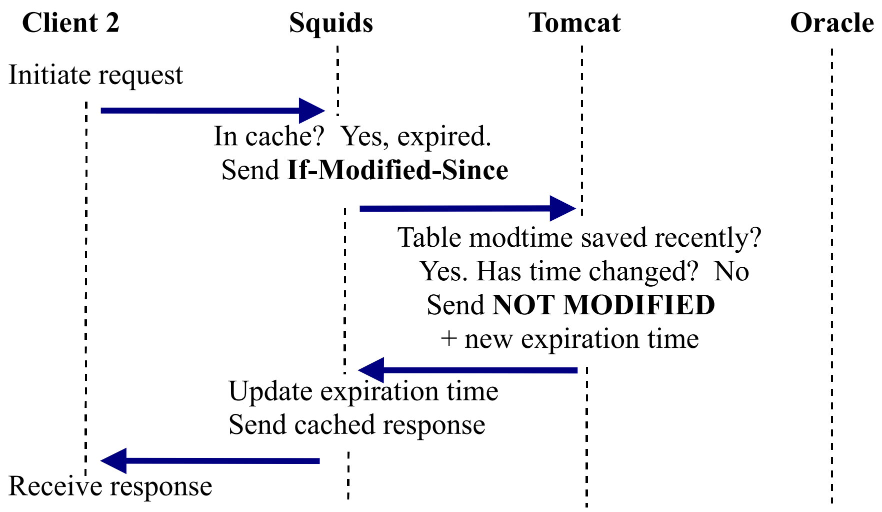


Figure 3. Another example client request after data item expires

In figure 3, a different client using the same Squid does the same query, sometime later after the cached result expired. The Squid determines that the item is in the cache, but expired. Since there is a **Last-Modified** header in the saved item, Squid adds an **If-Modified-Since** header to the request and sends it up to Tomcat. In this example Tomcat finds that it has looked up the modification time recently (from another transaction not shown) so it does not need to contact Oracle at all. It simply returns the very small **NOT MODIFIED** response plus a new expiration time. Squid updates the expiration time on its cached item and sends the response back to the client.

So the long-distance traffic, and the work that Tomcat and Oracle have to do, is greatly reduced when the data is not modified. Instead of including the full data payload, the response is a very small message. Tomcat saves the per-table modification times and re-uses them for up to 5 minutes (typically) so if there are many queries for the same table close together Oracle doesn't have to be contacted every time. These things allow us to greatly reduce the expiration times without overloading the servers, and so allow clients to notice changes much faster.

5. Getting modification times from Oracle databases

The most challenging part of this deployment was that the Oracle database does not normally track modification times. However, this capability was added using PL/SQL scripts. Three different approaches have been taken:

5.1. SQL triggers

The first approach was to use SQL triggers to track all table modifications and keep a timestamp in a table in each account. This worked, but database administrators worried that the triggers would be too heavy of a load on the database servers during updates so it was never deployed.

5.2. DBMS_CHANGE_NOTIFICATION

The second approach was to use Oracle's DBMS_CHANGE_NOTIFICATION package to update timestamps upon database COMMITs, and triggers to only track table CREATEs and DROPs. A bug was found that limited the length of account plus table names to an unreasonably short amount, but Oracle fixed that, so this is the approach that was first deployed. However, experience in production showed a few more problems:

1)Sometimes it stopped updating the timestamps.

2)The PL/SQL script for this had the inconvenience along with solution #1 of having to be installed in every account, and had to be reinstalled after an account's entire content is copied to another account.

3)Sometimes it interfered with streaming data from one server to another, as CMS does from its Online to Offline databases.

5.1. Copying from ALL_TAB_MODIFICATIONS

The final approach was to use Oracle's DBMS_STATS package to put modification times into an ALL_TAB_MODIFICATIONS view, and copy the timestamps from there to a single table for the database. This method was put into production in March 2009. It requires periodically flushing the database statistics, so a disadvantage is that it adds an additional 5 minute delay for changes to be noticed. The big advantage is that only has to be set up once per database which greatly reduces the maintenance work.

6. Squid bugs

We also encountered two Squid bugs that affected this deployment. These were fixed in a very timely manner by the Squid developers. Without these recent fixes, it would not have been possible to deploy this in systems like ours that have a Squid that feeds other Squids.

6.1. An 8 year old bug that had just been fixed

The oldest open bug in the Squid bug tracking system, bug #7 from August 2000, reported that HTTP headers were not rewritten in the cache when they are updated. It is necessary to update the cached **Date** header in order to use **If-Modified-Since** when one Squid feeds another. Very fortunately for us, the latest stable Squid release 2.7 had finally fixed this 1.5 months before we ran into it so all we had to do was upgrade to that version.

6.2. A related bug we found

During testing, we found a case where the **Date** header of cached items was still not updated. This was Squid bug #2430 and was fixed in October 2008 in Squid release 2.7STABLE5.

7. Effect on Frontier infrastructure

The load on the 3 central Frontier servers at CERN did go up significantly when we cut over in mid-December 2008 but they still have plenty of spare capacity. Figure 4 shows a plot of the combined number of requests per minute for the 3 central Squids over the last year (averaged per day).

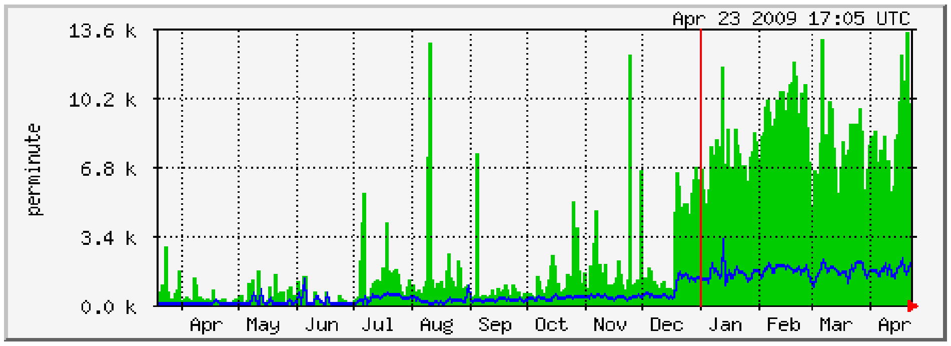


Figure 4: Requests per minute on CMS central squids at CERN over the last year

The green area is the requests received from all other Squids and the blue line is the requests passed on to Tomcat. Squids at the Tier sites often serve 40k+ requests per minute each and under extreme conditions some have been measured as high as 280k requests per minute, so a few thousand requests per minute for each of the three servers is easy. Also, almost all of the requests that are passed on to Tomcat are the very small revalidation requests and most of those don't contact the Oracle database.

8. Conclusions

This new cache coherency mechanism has significantly reduced the time it takes for changes to propagate, and has successfully avoided heavily loading the CMS conditions data infrastructure. It has been a great help in improving availability and avoiding the need for interventions by the people responsible for maintaining the infrastructure. We took advantage of an HTTP capability that has been available for a long time, but we couldn't have done it much sooner using Squid because the capability was only very recently fully implemented in Squid.

For more information on Frontier see its web home page http://frontier.cern.ch.

9. Acknowledgements
We would like to thank DOE and NSF for support of the project under which this work was conducted. This effort is the result of many within CMS working together to anticipate the needs of the experiment. The CMS Computing and Offline Software organization has provided the overall data model and software framework in which Frontier is used. We greatly thank the numerous individuals within CMS and CERN IT who have contributed to its deployment, testing, and operation. Fermilab is operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

References

[1] Blumenfeld B, Dykstra D, Lueking L and Wicklund E 2008 CMS conditions data access using FroNTier *J. Phys.: Conf. Ser.* **119** 072007 (7pp)
[2] Berners-Lee T, Fielding R. and Frystyk H 1996 RFC 1945