Table of Contents

Frontier Distributed Database Caching System overview

This page gives an overview of the Frontier system for people that are considering whether or not to use it for new purposes.

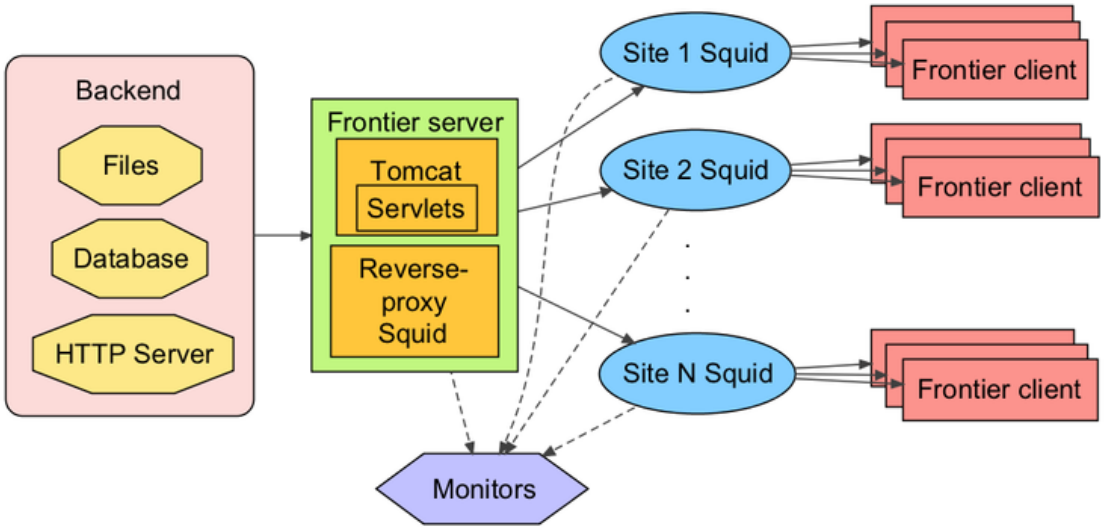
This is what is on this page:

What it is good for

The Frontier Distributed Database Caching system is ideal for applications where there are very large numbers of widely distributed clients that read basically the same data at close to the same time, with data that is best accessed as dynamically generated responses rather than static files. When data is to be widely shared but it is easy to store the data in files that don't change very often (such as software distributions), it is better to instead use the related CernVM FileSystem (CVMFS). Frontier is used to load calibration and alignment data from the large High-Energy Physics particle collider detectors CDF, CMS, and ATLAS, and has been deployed on hundreds of thousands of computer cores at hundreds of computing grid sites that do processing for those experiments. It is open source, has a general architecture, and has years of robust production experience, so it could be very useful for other applications as well.

What it is

The Frontier system is made up of Frontier clients, http proxy caches (squid), Frontier servers, and monitors. Clients make http requests to the proxy caches, which contact the Frontier servers (via a reverse-proxy squid on the same computer) when the proxies don't have a valid unexpired response already cached. The Frontier servers decode the requests and contact a back-end, either an SQL database server, another http server, or files on the Frontier server's local disk. The monitors watch for problems on the squid servers and Frontier servers, and plot behavior useful for debugging operational problems.



The current production Frontier systems do a very good job of concentrating the data so that it doesn't take very many central resources to provide service. For example, a 2012 CMS study measured about 5 million responses from 3 Frontier servers per day, containing about 40 GB data total, while the grid squid caches served many more requests per day, over 700 million, and delivered over 40 TB of data. That's a 140-to-1 concentration in requests and a 1000-to-1 concentration in data. The difference in the scale of the

concentrations of requests vs data is largely due to the If-Modified-Since feature (details below).

Major features

These are the major features of the Frontier system:

1. Extendable architecture. The Frontier system supports a mix of data types and multiple back-ends. The client API generically supports multiple data types, and the Frontier server has an extendable back-end plugin architecture. Currently there is an SQL plugin to connect to SQL databases of multiple types via JDBC (Oracle and MySQL/MariaDB databases have been tested), and a "file" plugin that can either serve ordinary files from the local disk of the Frontier server or read data streams from another server via http. Serving ordinary files through the Frontier system makes the most sense when either CVMFS is not available or the files change frequently.
2. Queuing. With so many active clients, sometimes the clients together request more items simultaneously than the back-end server is configured to allow. The Frontier server queues those requests until a back-end connection is available. The maximum number of connections to the back-end and the maximum number of queued requests are configurable. A monitor plots queue sizes and response times in the server and sends alarms when the queue length goes over a configurable threshold.
3. Fault tolerance. The system is designed to continue to function even if there are failures in all proxies and all but one server. The client is configured with the URLs of all the servers and proxies, including backup proxies, and tries them in an optimized order. If no ordinary proxies work, backup proxies are tried and/or servers are directly contacted. That is important especially for smaller grid sites that only have one proxy, but sometimes even when proxies are running they have overloads or other conditions that cause some of the requests to fail. Monitors watch for significant numbers of direct connections to servers or backup proxies to prevent such a non-optimal operation from continuing for very long (CMS usually experiences several of these failover incidents each week). Clients are configured with the names of all the individual servers, even if they are available in a round-robin, because it is not possible to tell a proxy which server in a round-robin to try. The individual IP addresses could in theory be inserted into the URL given to the proxy, but that interferes with access controls put on proxy squids that limit the destinations to the names of specific servers (and putting IP addresses in the access controls requires more undesired maintenance effort) and also interferes with authenticity checking. More details on fault tolerance are available in this client retry strategy document.
4. Keepalive messages. Because there are so many combinations of proxies and servers to try for fault tolerance, timeouts are set to short values by default to prevent failures from introducing long delays. On the other hand, queuing or back-end server load can cause a response to take a considerably long amount of time, so the Frontier server periodically sends keepalive messages while waiting to prevent clients from timing out.
5. Load balancing. If proxies are specified as round-robin names, every one of the IP addresses listed will be tried by the Frontier client if previous proxies have errors. As mentioned above, this is not the case for servers because there is no way to do it using the http proxy standard. The client also has options to do its own random load balancing between all specified proxies (not counting the backup proxies) or between all specified servers, if using round-robin is not desired.
6. Flexible cache coherency options. The Frontier system is especially good for data that is dynamically generated, which means that the data can change, which means that cache coherency strategies are needed. These are the different strategies that the Frontier system supports:
    1. Multiple servlets. Different workflows typically need to respond to changes more quickly, so the first part of the strategy is that the Frontier server can be configured with different servlets that each can have different cache expiration times for the same back-end server (or they can be configured to go to different back-ends). The different servlets have sufficient protection between them so they do not interfere with each other.
    2. Multiple time-to-live levels. Well-designed applications can split up their data into smaller requests of data that might change and larger requests that don't. An API to the Frontier client

can tell it to distinguish between the two types of requests, and the Frontier server can be configured to have different cache expiration times for the "short" time-to-live requests and the "long" time-to-live requests. A third level "forever" is also supported to distinguish types of requests that are guaranteed to never change, as opposed to "long" requests which aren't supposed to change but occasionally might.

3. Fresh URL key. For applications that need very fast updates, the Frontier client supports an option to add a given string key to the URL, so that whenever the value changes it is guaranteed to get fresh cache values yet the cache can still be shared among a large number of jobs. CMS uses this for some workflows by filling in the "run" number designating a data-taking session, and applies it only to the smaller requests that might change.
4. If-Modified-Since. Finally, the Frontier system supports the http feature that uses the Last-Modified/If-Modified-Since headers. This enables frequent checking for changes at very low cost, such that if there are no changes the response is a very fast "NOT MODIFIED" which revalidates cached items so they can continue to be used for another period of time. Details are in this CHEP 2009 paper. This is supported in both the SQL plugin and the file plugin, the latter with both files and http server back-ends.

7. Short cache expiration times for errors. If a query encounters an error from the backend server, the Frontier system makes sure that it is cached for a short time. Not caching at all is not good because it can overload the Frontier system if an error persists, and caching for the full time is very bad because then it can take a long time to recover after an error goes away. If an error occurs after the cache expiration time HTTP header has already been sent, the Frontier server specifies a short maximum cache age (default 5 minutes) at the end of the response and then when that time has elapsed the frontier client retries with an HTTP header that tells the cache to limit the cache age to that length of time. For caches that ignore the client-requested cache age limits (such as Cloudflare), the server has an additional optional feature to prevent those kinds of error responses from being cached at all by avoiding sending the HTTP/1.1 zero-length last chunk. Error messages are also specified at the end of the response so clients can be informed of what went wrong on the server.
8. Data compression. The Frontier system uses data compression by default, saving both cache space and network bandwidth at the cost of a small amount of extra server and client CPU time.
9. Persistent connections. Network connections are re-used throughout the system for multiple requests, between the Frontier client and squid, between squid and another squid, between squid and the Frontier server, and between the Frontier server and an SQL database. This significantly reduces network overhead because of the fewer packets that need to be sent back and forth.
10. Authenticity/integrity of data. The Frontier system supports optional digital signatures on the responses based on standard X.509 certificates. When enabled, the Frontier client verifies that the data matches the signature and that the signature comes from a verified certificate matching the host name of the server. This prevents the possibility of tampering with the data which could potentially steal computing resources on a large number of grid computing nodes. Details are in this CHEP 2013 paper.
11. Proxy auto-discovery. It can be a very difficult problem for a job to discover where the local squid proxies are when it arrives at a grid site, and for administrators at large sites to control which types of jobs use which proxies. For this reason, the Frontier client supports the Web Proxy Auto-Discovery (WPAD) internet standard. This can make use of a general WPAD service for finding web proxies on the WLCG.

Additional component-specific features

This section contains additional component-specific features not covered above.

Frontier clients

Frontier clients have both C and C++ language APIs. A python binding has been written that isn't yet part of the client package. Details about the client including configuration options are on this web page. Some of the features not mentioned above are:

1. The client has a configuration option to have the client itself cache results that are smaller than a given size. This is useful for applications that can't easily avoid repeating queries.
2. The client sends information about itself plus a string the application chooses, to identify the client and job in squid logs and Frontier server logs for debugging.
3. The client package includes a command line tool called 'fn-fileget' for easy retrieval of files using the server's file plugin.
4. The client package includes two command line clients for testing with the SQL plugin, one a very simple python tool called 'fnget.py' that does not use the full frontier client (for testing the server), and one called 'fn-req' that does use the full frontier client library.
5. The client package also contains a command line tool called 'frontierqueries' that pulls out SQL queries from a client debug log.

CMS and ATLAS use the Frontier client SQL plugin through an abstract database API system called CORAL, but the client can also be used directly. ATLAS additionally uses another generic layer above that called COOL that provides an Interval-Of-Validity system. CMS has its own simpler layer.

Frontier squid distribution

The Frontier project supports its own distribution of the standard squid package, called **frontier-squid**. It distributes the squid version that has been proven to work the best with applications like Frontier that use If-Modified-Since and that have large numbers of clients requesting the same data at around the same time. The squid versions distributed with Redhat's EL5, EL6, and EL7 do not work well with either of those requirements (see the NOTE at the top of this page for details). The rpm also has these additional features:

1. A configuration file generator, so configuration customizations can be preserved across package upgrades even when the complicated standard configuration file changes.
2. The ability to easily run multiple squid processes listening on the same port, in order to support more networking throughput than can be handled by a single CPU core (squid is single-threaded, although squid3 added native support for multiple squid processes).
3. Automatic cleanup of the old cache files in the background when starting squid, to avoid problems with cache corruption.
4. Default access control lists to permit remote performance monitoring from shared WLCG squid monitoring servers at CERN.
5. The default log format is more human readable and includes contents of client-identifying headers.
6. It chooses default options found to be important by years of operational experience on the WLCG.

One of the advantages of using http proxies is that they can be nested in a hierarchy wherever additional bandwidth is needed. The CMS detector's High Level Trigger cluster (which filters out undesired LHC collision events) makes especially good use of this feature because it needs to simultaneously load data to thousands of CPU cores as quickly as possible. It uses a separate custom squid rpm on every node in the cluster which configures itself into a hierarchy so that each node sends data to no more than four other nodes.

Frontier server

The Frontier server is distributed in an rpm called **frontier-tomcat**. It includes a standard Apache Tomcat web server written in Java, the Frontier application servlet, and a configuration system that configures the servlets in a concise format. The best documentation with all the details is in the frontier-tomcat installation instructions page. The important features are all covered well above.

Monitors

These are the primary monitoring systems in use in the Frontier system.

1. MRTG - the production squids at grid sites around the world are all monitored centrally via SNMP

requests from a server at CERN http://wlcg-squid-monitor.cern.ch using the web standard Multi Router Traffic Grapher. This collects information every 5 minutes that is very powerful for debugging problems.
2. awstats - reverse-proxy squids that run on the same machines as the frontier-tomcat servers, and also the proxy squid used for backups, use an additional log analyzing monitoring tool from the internet called http://awstats.sourceforge.net. The web interface to the existing monitors are available on the frontier.cern.ch server. This monitor gives additional useful information for debugging, especially summaries of where the connections are coming from. The part that is installed on each of the Frontier servers are distributed in another rpm called **frontier-awstats**.
3. failover monitor - another monitor analyzes the output of awstats and looks for failovers coming directly from grid site worker nodes that should be going through their local squids instead. This monitor plots failovers coming from the sites and sends alarms to the site administrators when the number of failovers go over a threshold.
4. queueing monitor - this monitor analyzes the frontier-tomcat logs to plot the behavior of queuing and response times, and it sends alarms when the number of waiting threads goes over a threshold. The web interface is available on the frontier.cern.ch server. The part that is installed on each of the Frontier servers is included in the same rpm as awstats.
5. SLS and SUM tests - these do periodic sanity tests of the servers and the grid site proxy squids. Links to them for CMS and ATLAS are on the Frontier web server home page.

More details on the Frontier monitors are available in this CHEP 2012 paper.

Ongoing support

The Frontier system is in active use and is continuing to improve and evolve to have additional features. It is supported by Fermilab. There is a frontier-talk@cern.ch mailing list for discussions between users, a tracking ticket system, and a `frontier-support@fnal.gov` mailing list for support questions.

This topic: Frontier > FrontierOverview
Topic revision: r14 - 2020-01-08 - DaveDykstra