

Access to HEP conditions data using FroNtier: A web-based database delivery system

Lee Lueking
Fermilab

International Symposium on Grid
Computing 2005
April 26, 2005

Credits

Fermilab, Batavia, Illinois

Sergey Kosyakov, Jim Kowalkowski, Dmitri Litvintsev,
Lee Lueking, Marc Paterno, Stephen White

Johns Hopkins University, Baltimore, Maryland

Barry Blumenfeld, Petar Maksimovic

Outline

- Introduction to the HEP Conditions DB Environment
- FroNtier project details and experience at the CDF Experiment.
- Possible use of FroNtier for CMS conditions data.

The HEP Database Environment

- Databases are used to maintain information about the detector's operation, and details needed for calibration and alignment of the many detector sub-systems.
- This information is needed on-line, in real time to, operate the detector and off-line to understand the physics content of the “raw signal” data coming from the detectors.
- The off-line environment is dependent on Grid computing to provide the resources needed to process and analyze the complex signal data at processing centers worldwide.
- A highly distributed database delivery system is needed to accompany the Grid computing machinery.

What are “Conditions” Data?

■ Monitoring

- Sensor channel values (HV, LV, Temp, Pressure,...) move independently of each other in time. Monitor information about the detector.
- Data is collected in real time and has a single “*version*”.

■ Calibration

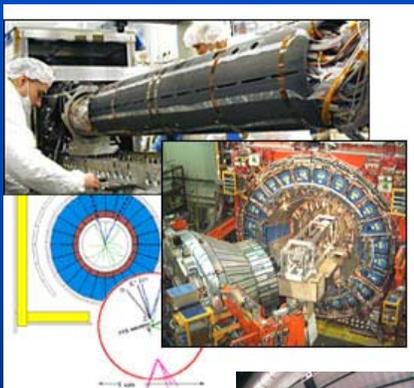
- Needed to understand the response of detector channels to signal input.
- “*Algorithms*” are used to create the data. More than one algorithm might be stored, and each might have multiple “*versions*”.

■ Alignment

- Precision alignment of components of the detector which are used for “particle track recognition” is essential.
- The many sub-systems comprising the detector must be aligned relative each other.

Characteristics of Conditions Data

- In general:
 - The frequency of access for a *data object* is dependent on the kind of object and what the requesting application is doing.
 - It is very likely that the same object will be accessed by multiple processing applications working on signal data taken at similar times.



- For the CDF Detector, conditions data objects vary in size from a few bytes, to a few MBs.



- For the CMS Detector, conditions data objects vary in size from a few hundred bytes, to a few hundred MBs.

Database Access Requirements

- Thousands of clients distributed at processing centers worldwide.
- Likelihood to reuse cached objects at each center by many clients is high.
- High availability for database access.
- Stateless servers are much preferred over database replicas that have higher administrative overhead.
- Security and Access Control that fits easily into the network. Compute servers will be behind firewalls and on private networks.
- Decoupling the client API from Database schema is highly desirable. This simplifies development and long-term maintenance of both.

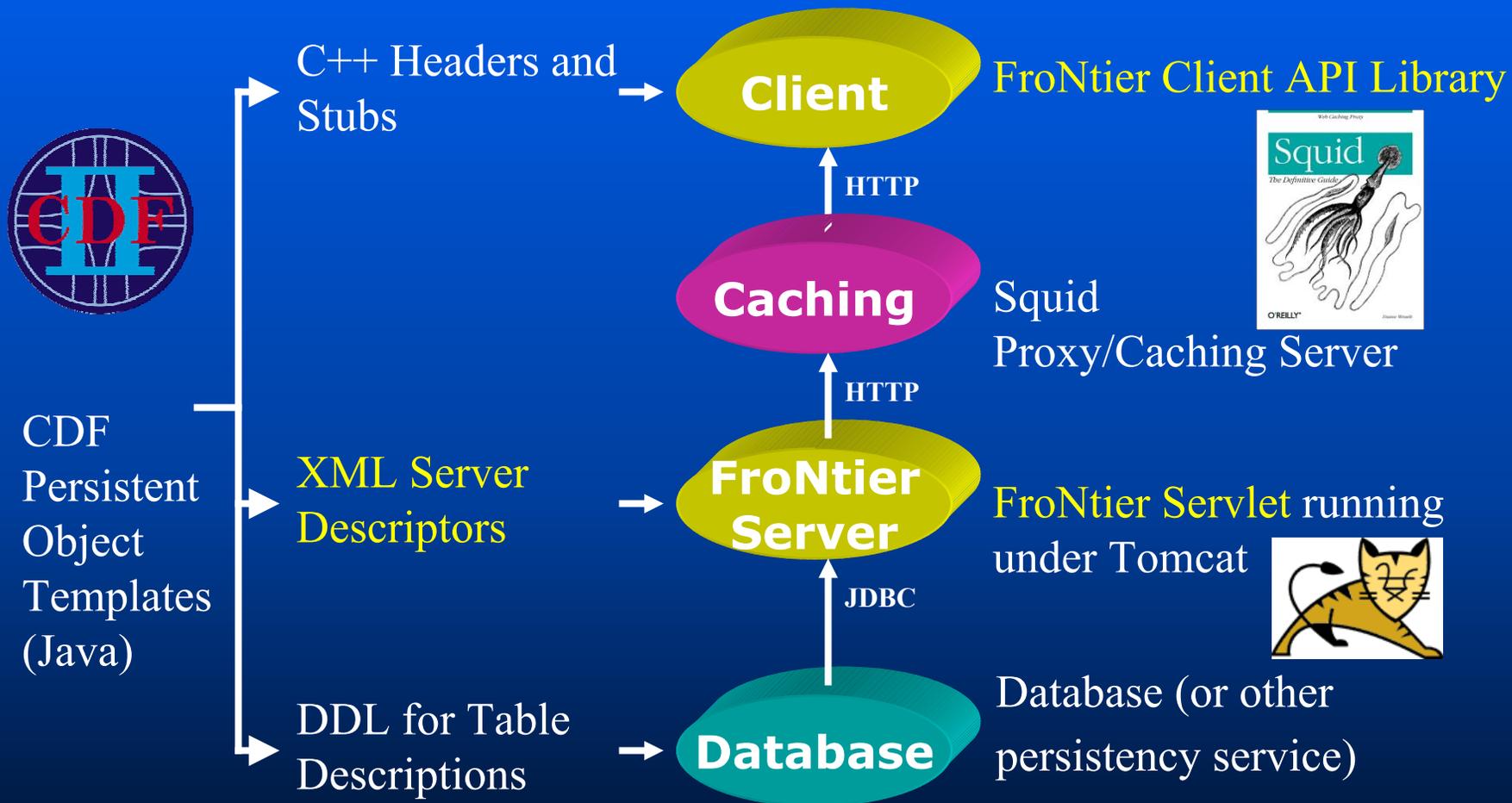
How to Best Deliver Data Objects?

- Central Database, or replicated to one or two additional sites for redundancy if needed.
- Stateless “application” servers configured for load balancing and failover, provide connection pooling to the DB.
- Stateless network components, proxy caching servers, at each GRID processing center provide access control and data caching.
- Grid jobs (clients), running on the Grid compute resources, need outgoing access to the internet, through the proxy caching service.

The FroNtier Project

- Goal: Assemble a toolkit, using standard web technologies, to provide high performance, scalable, database access through a stateless, multi-tier architecture.
- Pilot project Ntier tested the technology:
 - Tomcat, HTTP, Squid
 - Client monitoring w/ existing CDF tools (udp messages)
- FroNtier project was established to provide a production system for CDF and other interested users
- <http://whcdf03.fnal.gov/ntier-wiki/FrontPage>

FroNtier Overview



CDF
Persistent
Object
Templates
(Java)

C++ Headers and
Stubs

XML Server
Descriptors

DDL for Table
Descriptions

Client

Caching

FroNtier
Server

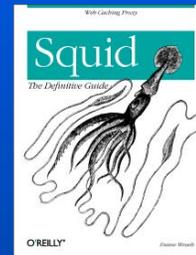
Database

FroNtier Client API Library

Squid
Proxy/Caching Server

FroNtier Servlet running
under Tomcat

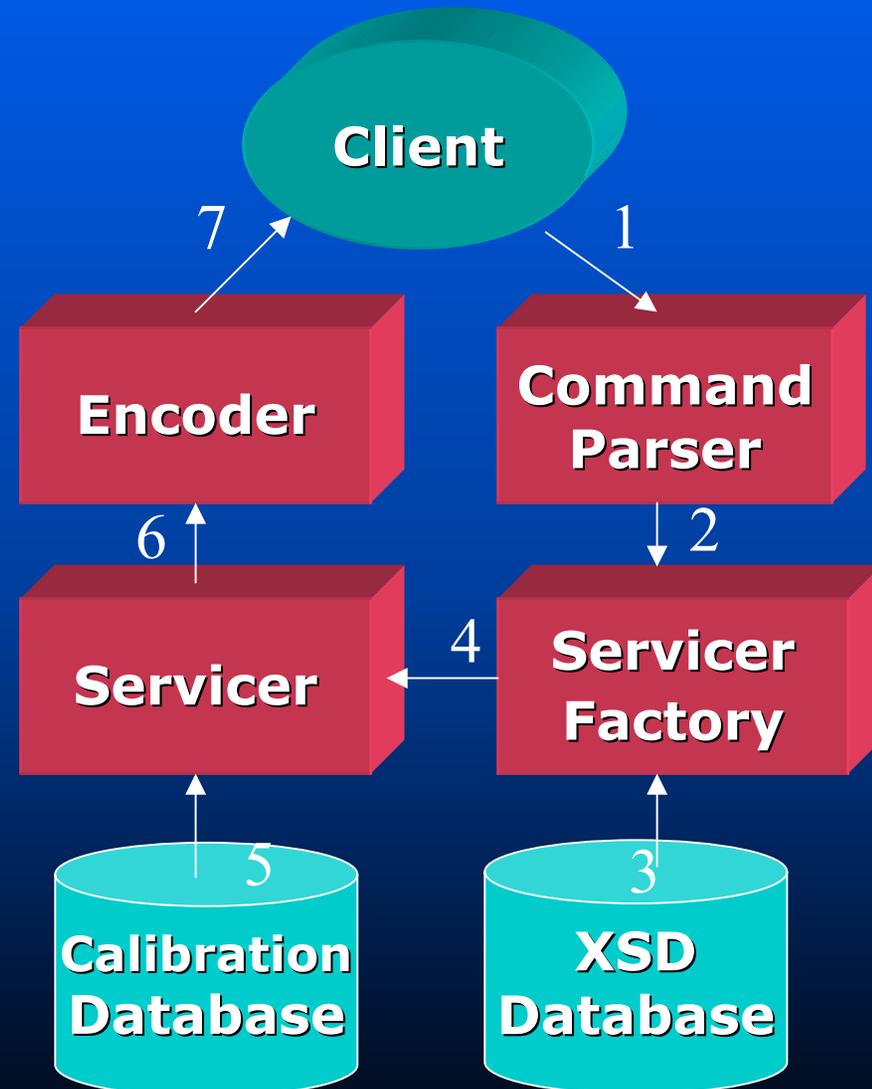
Database (or other
persistency service)



FroNtier components in yellow

The FroNtier Servlet

1. Client sends request (URI)
2. *Command Parser* translates URI into commands + values
3. *Servicer Factory* gets XSD (XML Server Descriptor) from database and
4. Instantiates a *Servicer*
5. Servicer queries database and
6. Results sent for encoding
7. *Encoder* marshals (serializes) the data to requesting client



FroNtier XML Server Descriptor (XSD)

- Object name and version information
- Response description
- The SQL mapping to the database
 - Select statement
 - From statement
 - Where clause
 - Special modifiers (*order by*, etc)

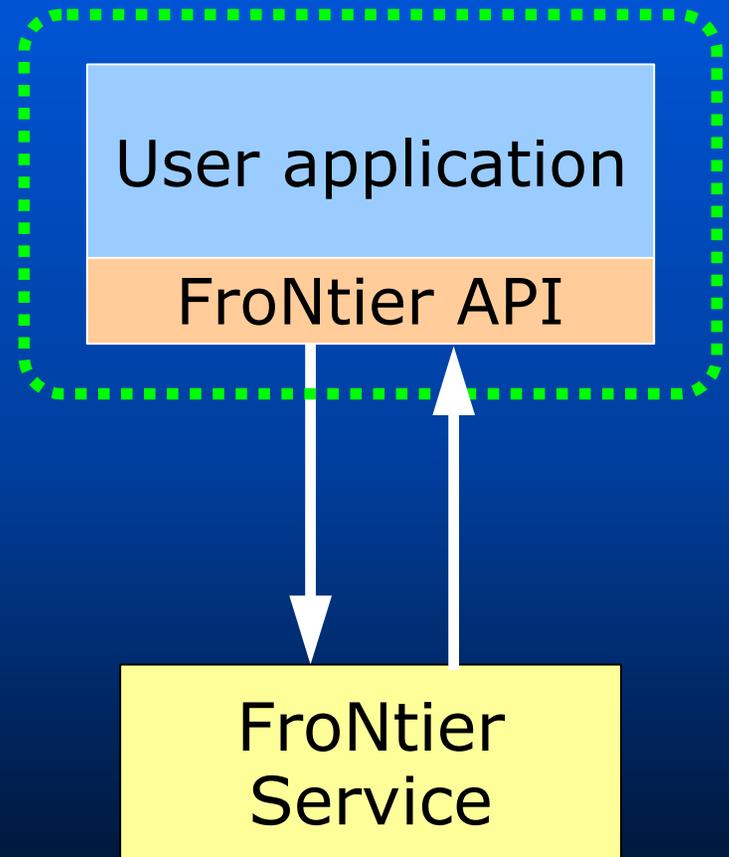
```
<descriptor type="CalibRunLists"
            version="1" xsdversion="1">
  <attribute position="1" type="int"
    field="calib_run" />
  <attribute position="2" type="int"
    field="calib_version" />
  <attribute position="3" type="string"
    field="data_status" />
  <select>
    calib_run, calib_version, data_status
  </select>
  <from> CalibRunLists </from>
  <where>
    <clause> cid = @param </clause>
    <param position="1" type="int"
      key="cid" />
  </where>
  <final> </final>
</descriptor>
```

FroNtier use of Squid Cache

- HTTP Proxy Caching Server: <http://www.squid-cache.org>
 - Well documented, widespread operational experience
 - Easily installed and maintained
 - Highly configurable for access control, disk cache tuning, distributed cache peer relationships, and more.
 - Monitoring built in through SNMP-2 interface
- Cache Refresh options
 - Servlet: expiration time sent in HTTP header
 - Client: forced object refresh through request
 - Administrative: Delete each Squid's cache files and rebuild the cache
- However, the objects being delivered are generally not changing, so a static cache meets most requirements.

FroNtier client API features

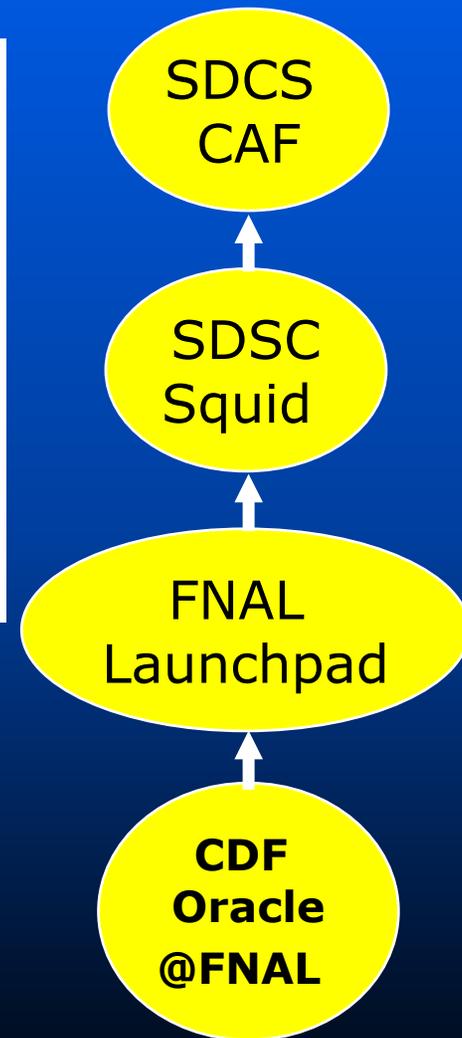
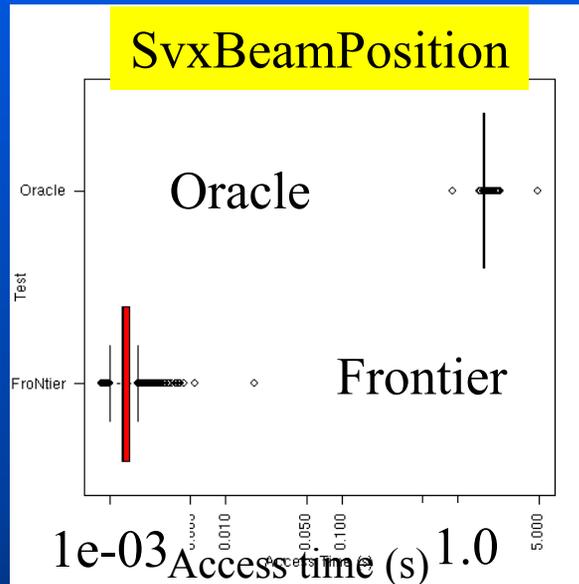
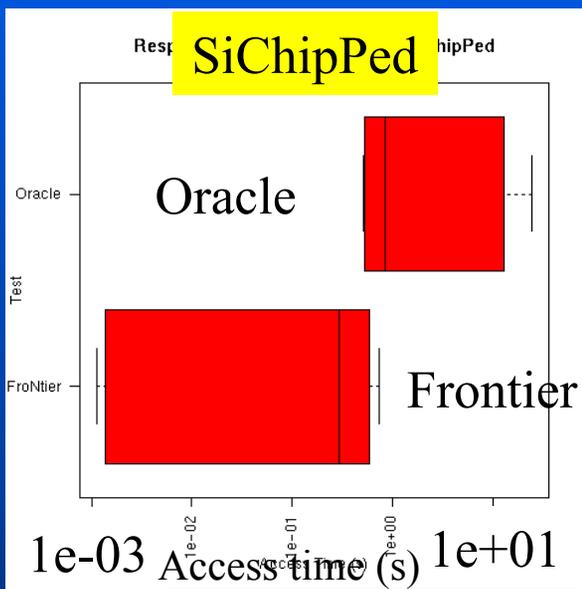
- Compatible with C and C++
- Portable
 - 32 and 64 bit systems tested
- Transparent object access
 - Type conversion detection
 - Preserves data integrity
- Multi-object requests
- Easy runtime configuration
- Extensive error reporting
 - Adjustable log levels



CDF FroNtier Testing at FNAL/SDSC

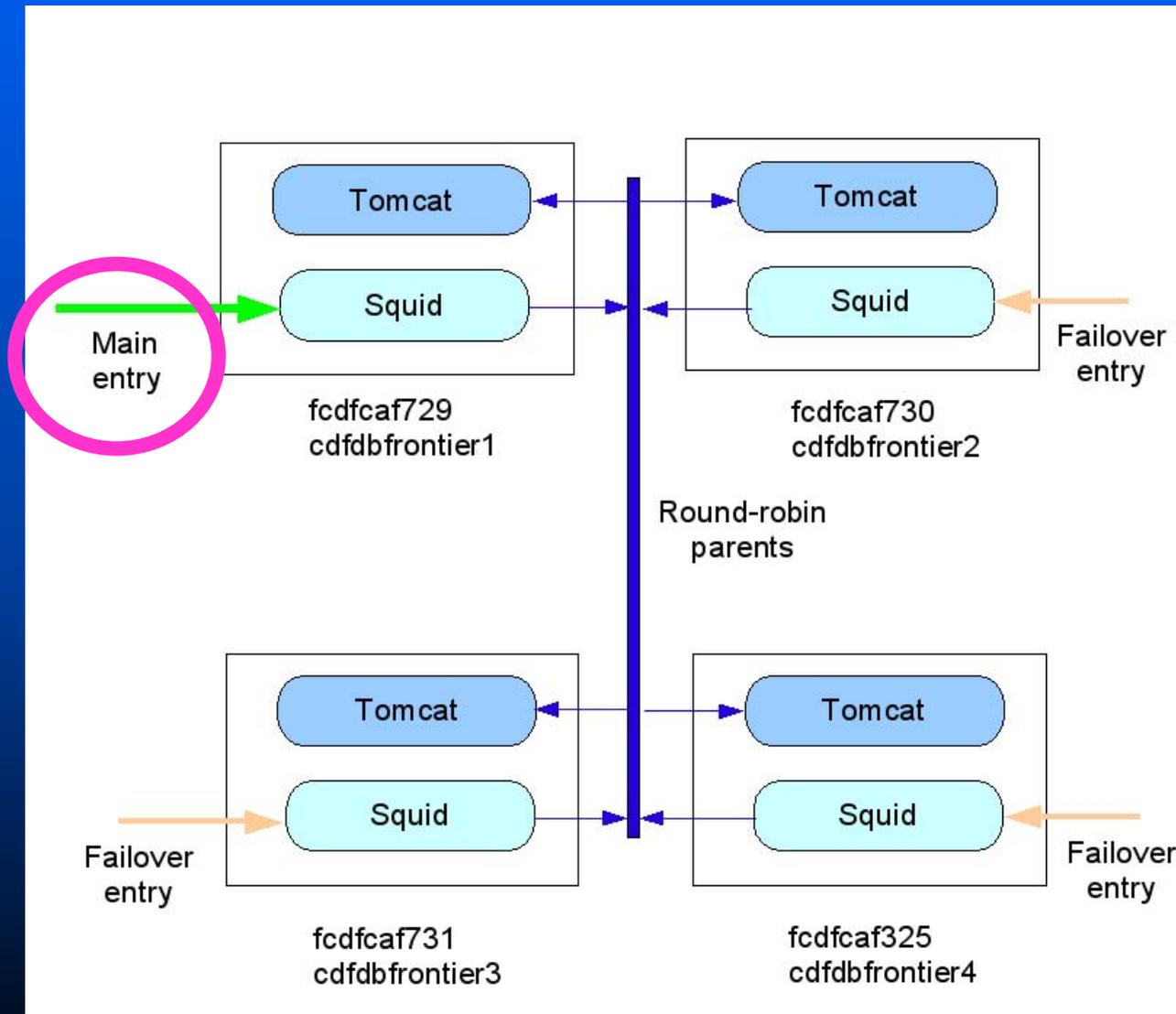
(San Diego Super Computing Center)

Access times for direct Oracle and Frontier



- SiChipPed objects are usually about 0.5 MB, up to 1.7 MB in size. (Silicon Chip Pedestals)
- SvxBBeamPosition objects are 502 Bytes (Silicon tracker beam position)
- The real savings are also in the reduced DB access.

CDF “Launchpad” at FNAL



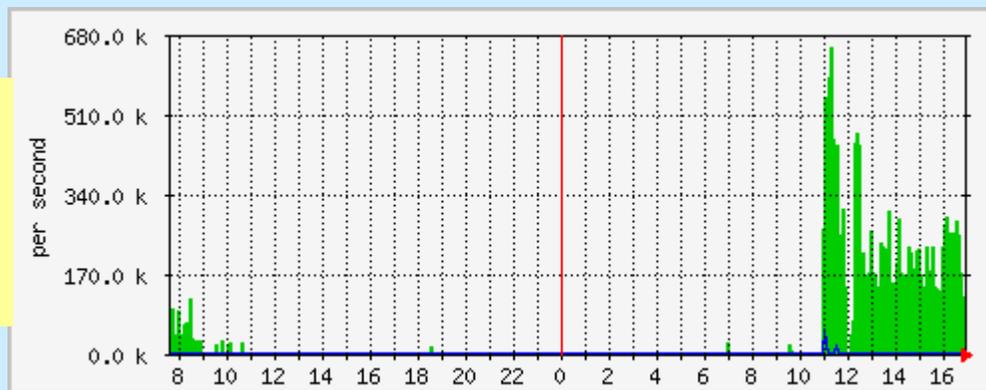
- Four general processing nodes
- CPU: dual 2.4 GHz
- Memory: 2MB
- Disk 100 GB
- NIC: GBit Ethernet
- Main entry squid uses tomcats in round robin fashion



CDF FroNtier Status

- Client library is included in CDF production code.
- DB access includes calibration, trigger, and other conditions information.
- Extensive validation confirms data obtained with direct Oracle access is the same as via Frontier.
- Squid deployment at CDF processing centers in San Diego (SDSC), Bologna (CNAF), Karlsruhe (GridKa), Toronto, Rutgers, MIT.
- Still being phased in, but activity is increasing rapidly.

SNMP data for
Data throughput
on Fermilab Squid
server. (KB/s)



Max **Total** 656.0 kB/s Average **Total** 106.0 kB/s Current **Total** 125.0 kB/s
Max **Fetches** 55.0 kB/s Average **Fetches** 1.0 kB/s Current **Fetches** 0.0 kB/s



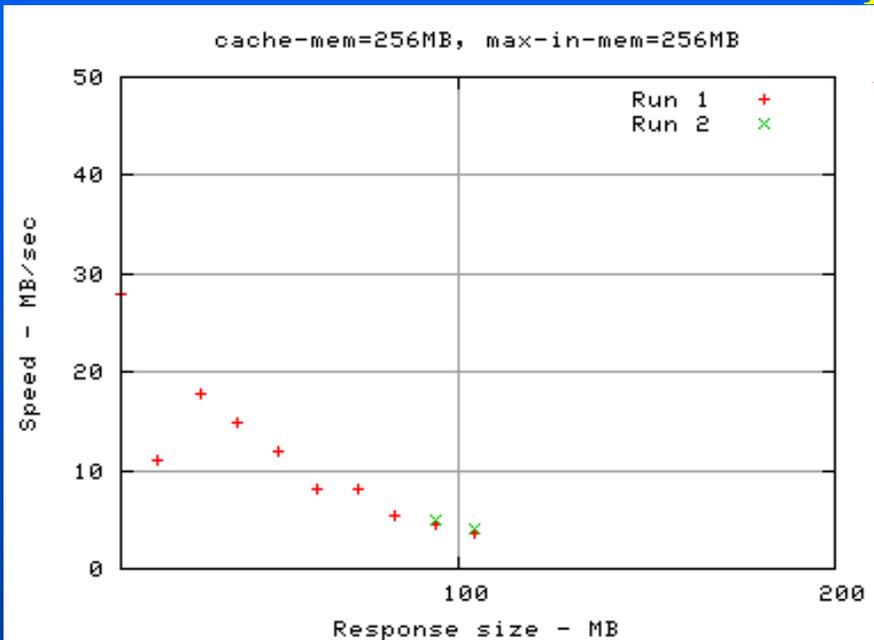
CMS FroNtier

- CMS is interested in using FroNtier approach for offline and possibly some online DB access.
- Off-line Requirements for DB access include large (several hundred MB) data “objects” by computing resources distributed worldwide.
- On-line needs include the High Level Trigger (HLT) farm with large objects and high demands on the cache.

CMS HLT: Challenging Environment

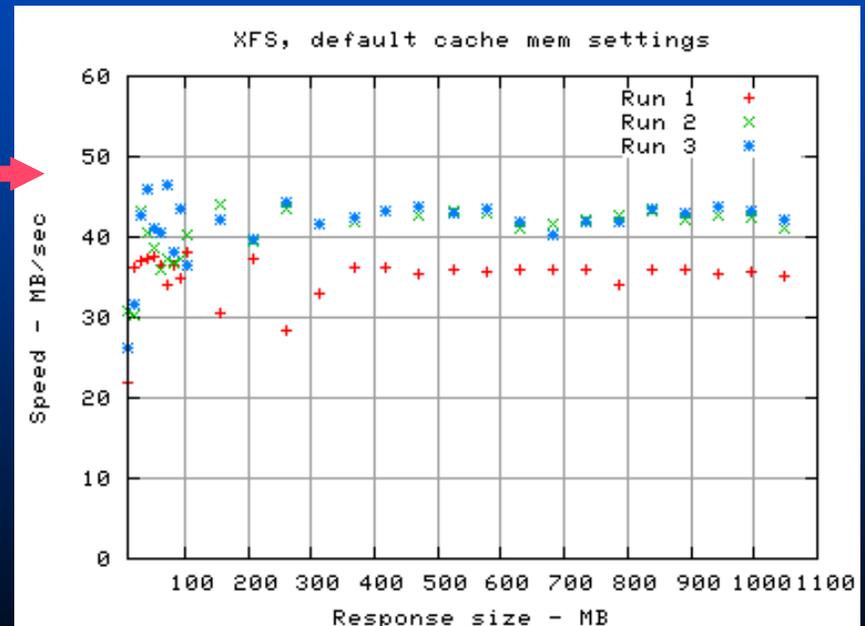
- The High Level Trigger Farm is a very interesting environment.
 - 1000 nodes, running ~4000 processes
 - Object sizes range up to several hundred MB.
 - Near real-time demands for new object caching.
- It has not been established yet that the Frontier approach will be used, however it is attractive.
- Concerns:
 - Will performance be sufficient for large data objects?
 - Is reliability sufficient under the heavy load?
 - What are the hardware and configuration needs?

Initial Squid Tests



- Attempting to use a large Squid memory cache fails miserably.
- cache_mem 256MB
maximum_object_size_in_memory 256MB
- Obviously, memory cache is not designed to work with big objects.

- Performance much better when NOT using Squid memory cache.
- In this test cache_dir was created on XFS disk partition.
- Results are 7 to 10 MB/sec better, compared to Ext2, with large RAM and good disk hardware XFS can perform even better.

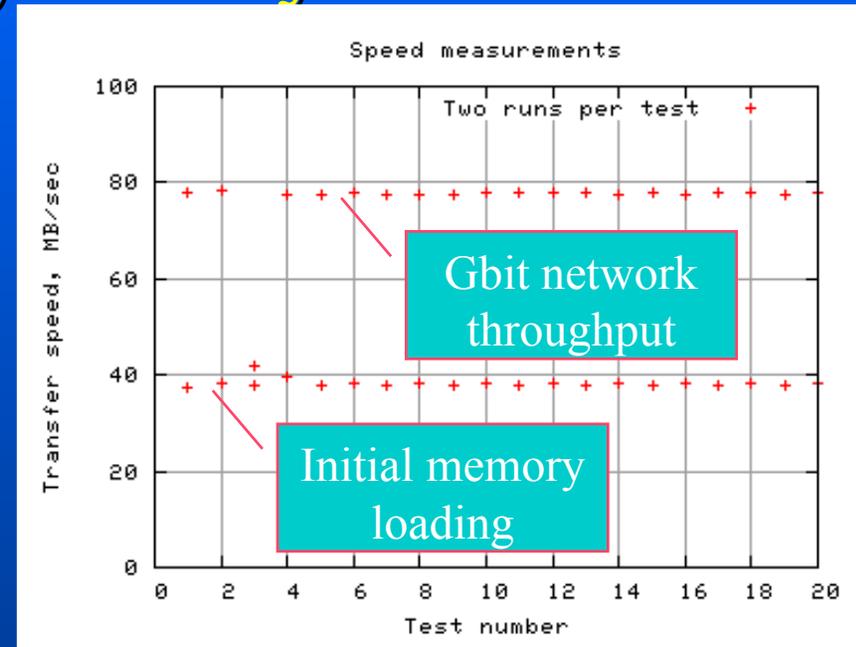


Evaluation Summary

- Squid performs very well with big objects, showing no decrease in performance.
- Attempts to improve performance by putting big objects into memory reduce performance dramatically
- For big objects Squid's performance is limited only by IO subsystem
- Performance can be improved by using good IO hardware and software: e.g. fast SCSI RAID in striping mode and non-journaling file system.

Using a Memory File System

- Configuration:
 - cache_dir of the Squid points to memory-based file system of 1200 MB size.
 - Memfs is sufficient to keep 2 calibration objects of 512 MB each, plus bookkeeping data.
 - Hard drive is used for keeping log files only.



- Using memory-based file system could be a very good solution for the on-line HLT farm, and other high demand environments.
- It is fast, cheap, and virtually maintenance-free (memfs regenerates itself on each OS restart)
- Bigger data (if needed) could be handled with bigger or multiple memfs systems, but, for sizes more than 3GB, 64-bit OS could be needed.

Summary

- HEP Conditions databases are essential to the operation of the particle detectors and needed for understanding the physics data.
- FroNtier is a multi-tier architecture providing high throughput, low latency, scalable access to a persistent store, such as a database.
- The CDF DB access framework has been adapted to use the FroNtier approach. It is in production and users are enthusiastic about the advantages provided.
- CMS is interested in using the Frontier approach for offline, and possibly some online, DB access. Evaluations are underway to understand how the system will perform. Results are promising.

References

- FroNtier Talks and Papers:
 - http://lynx.fnal.gov/ntier-wiki/Additional_20Documentation
- FroNtier working page:
 - <http://lynx.fnal.gov/ntier-wiki>