FroNtier Overview and POOL Interface Prototype

Lee Lueking
Fermilab

LCG3D Workshop

December 14, 2004

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

- Overview
- Use in CDF
- CMS Prototype
- Frontier & POOL

Why a multi-tier approach?

- Decouples client API from Database schema simplifying development and long-term maintenance
- Ideal for write once, read many data
- Reliability and high performance w/o expensive DB servers
- Readily scalable to serve thousands of clients
- Easily deployed w/ low administrative overheads

FroNtier Overview

CDF Persistent Object Templates (Java)

C++ Header and Stubs

XML Server Descriptors

DDL for Table Descriptions

Client

HTTP

Caching

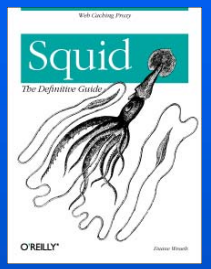
HTTP

FroNtier Server

JDBC

Database

FroNtier Client API Library



Squid Proxy/Caching Server

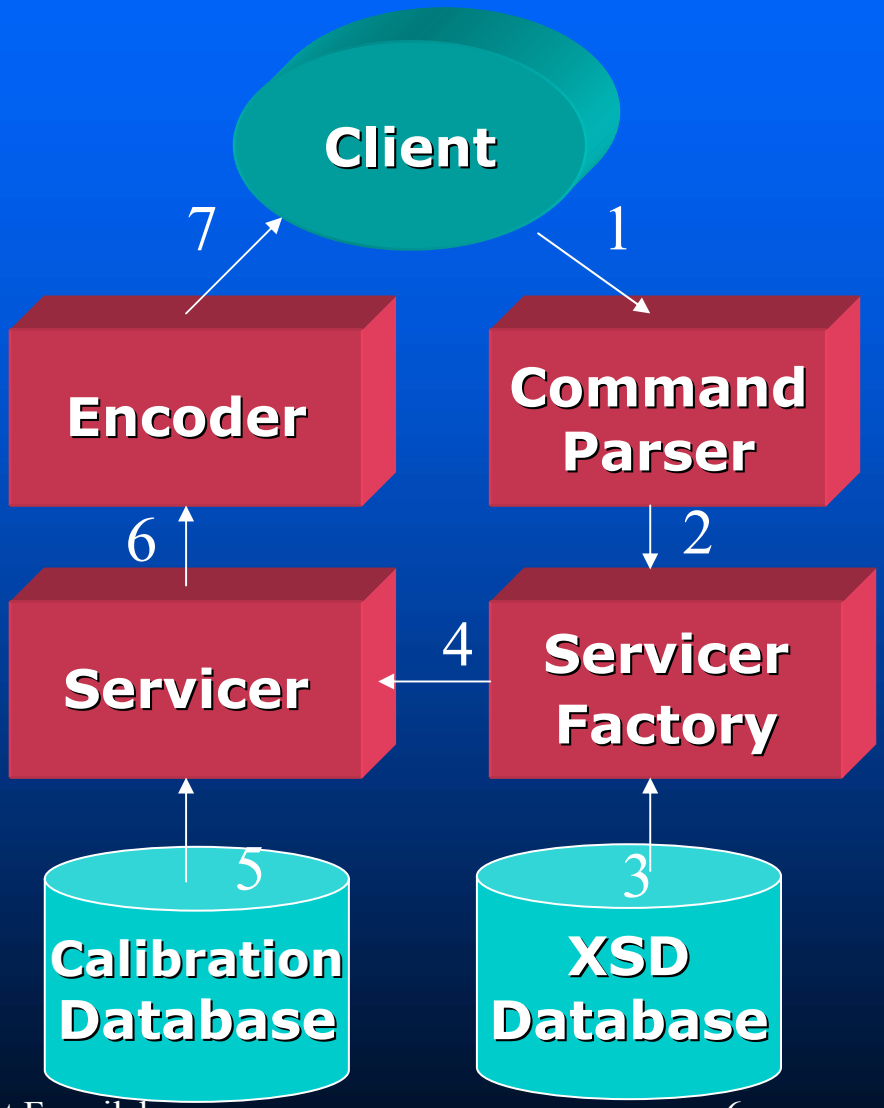
FroNtier Servlet running under Tomcat

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)

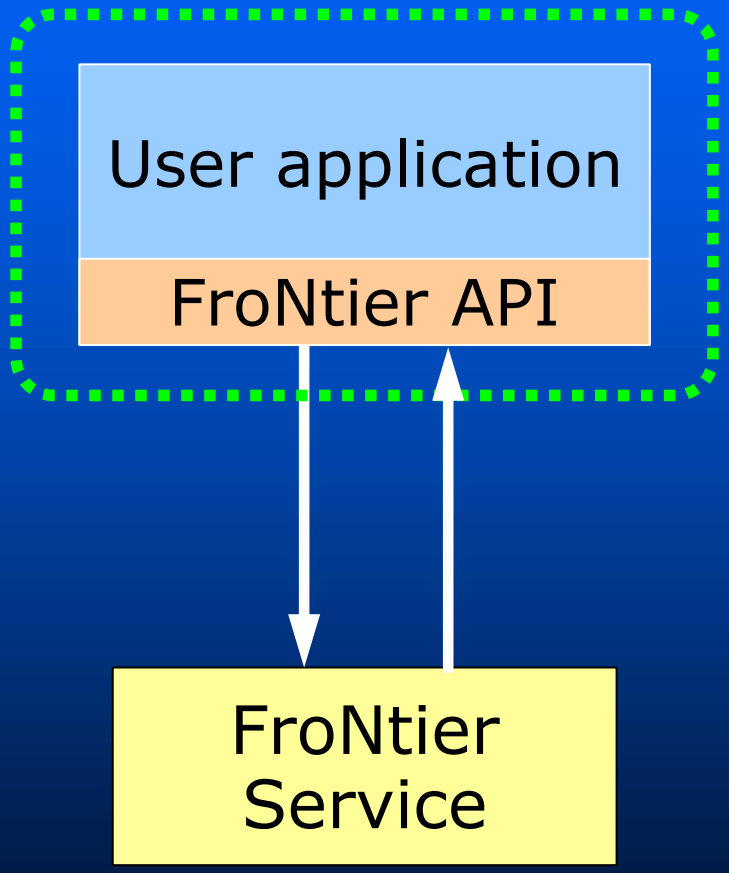
```
<descriptor type="CalibRunLists"
            version="1" xsdversion="1">
  <attribute position="1"  type="int"
  field="calib_run" />
  <attribute position="2"  type="int"
            field="calib_version" />
  <attribute position="3"  type="string"
            field="data_status" />
  <select>
    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 matches most requirements.

FroNtier client API features

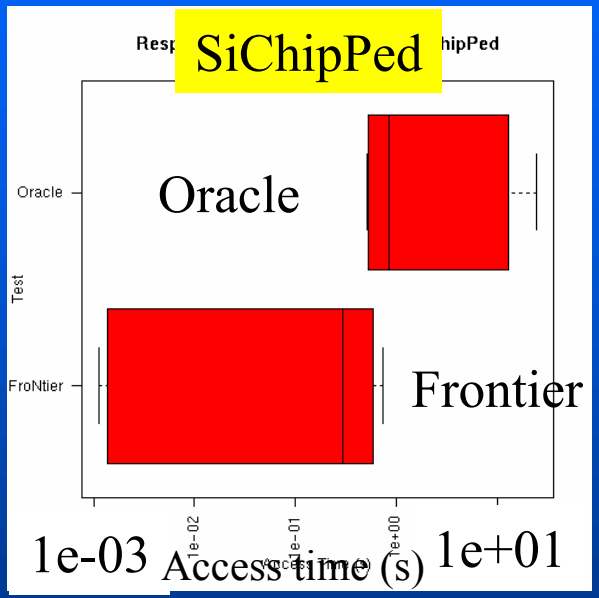
- Compatible with C and C++
- Portable
  - 32 and 64 bit systems tested
- Transparent object access
  - Type conversion detection
  - Preserves data integrity
- Multi-object requests
- Easy runtime configuration
- Extensive error reporting
  - Adjustable log levels

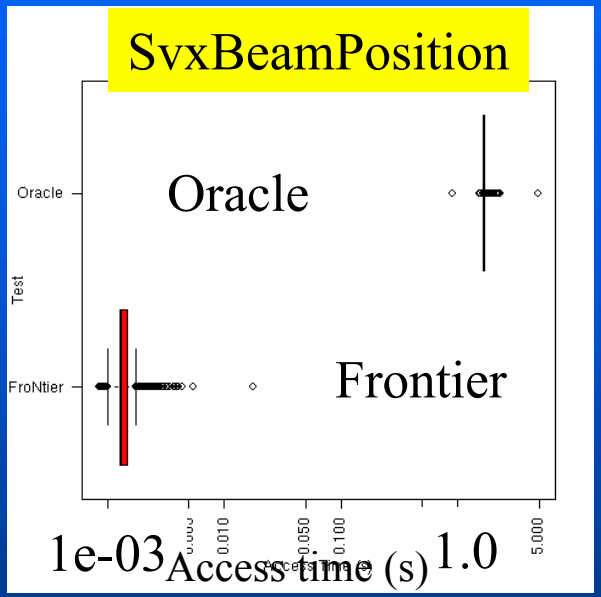


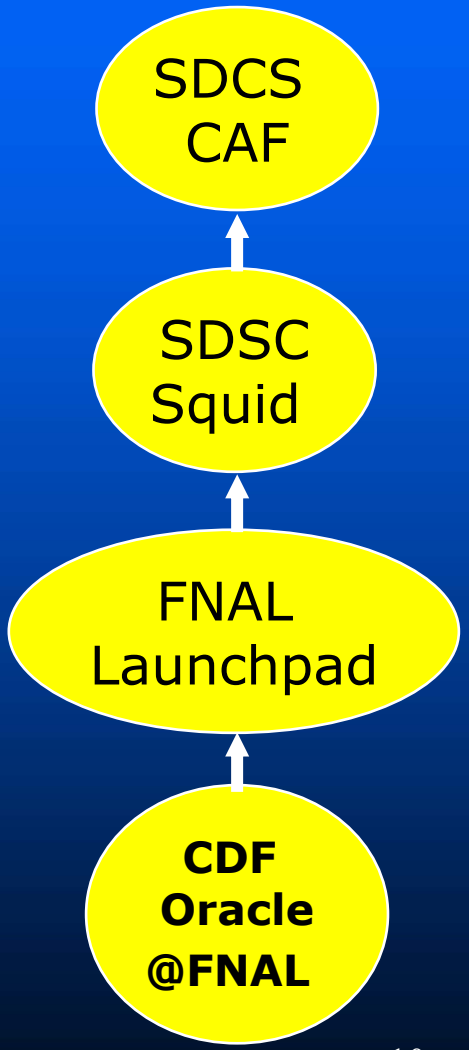
CDF FroNtier Testing at FNAL/SDSC

(San Diego Super Computing Center)

Access times for direct Oracle and Frontier

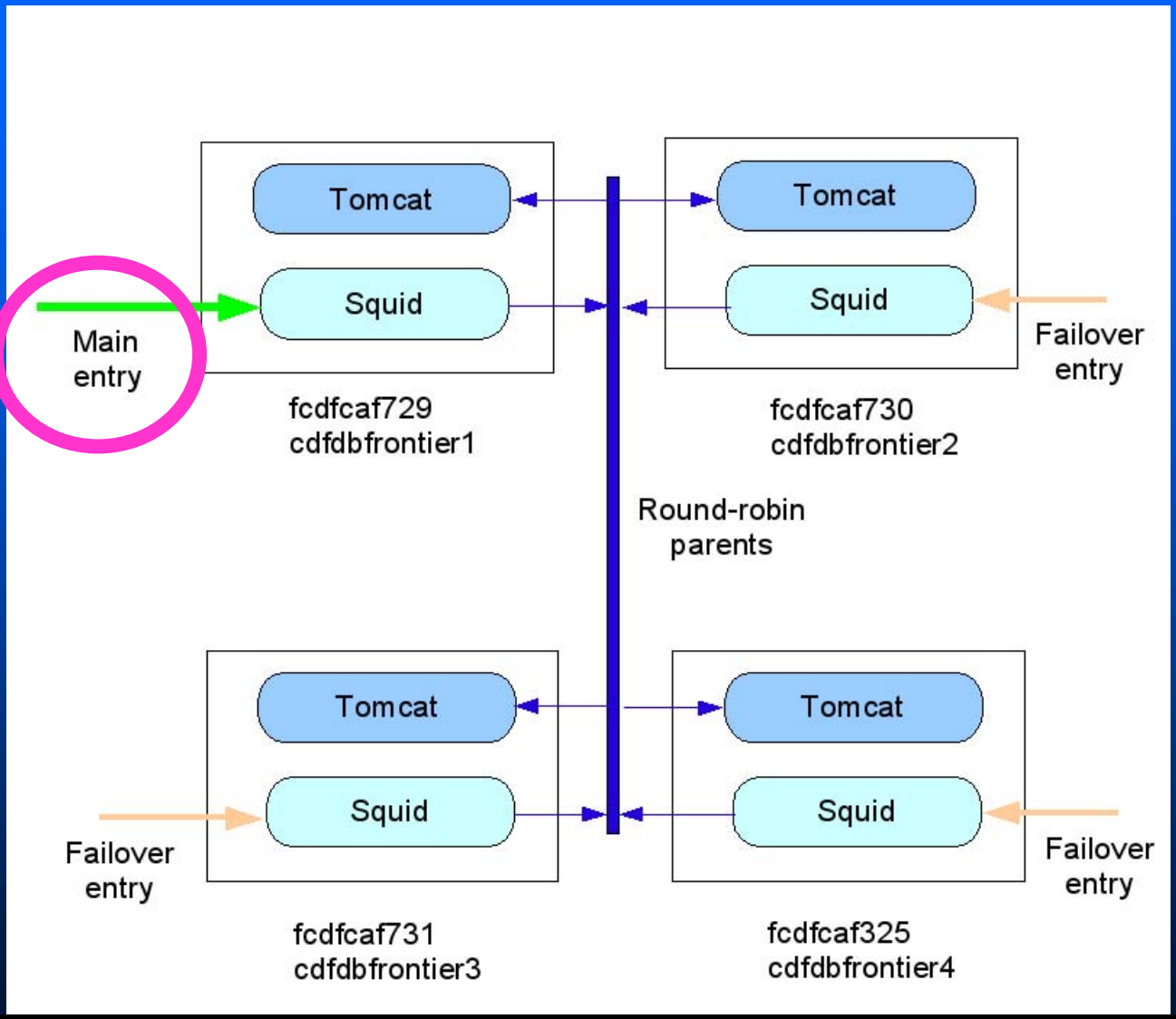






- SiChipPed objects are usually about 0.5 MB, up to 1.7 MB in size.
- SvxBeamPosition objects are 502 Bytes
- The real savings are in the reduced DB access.

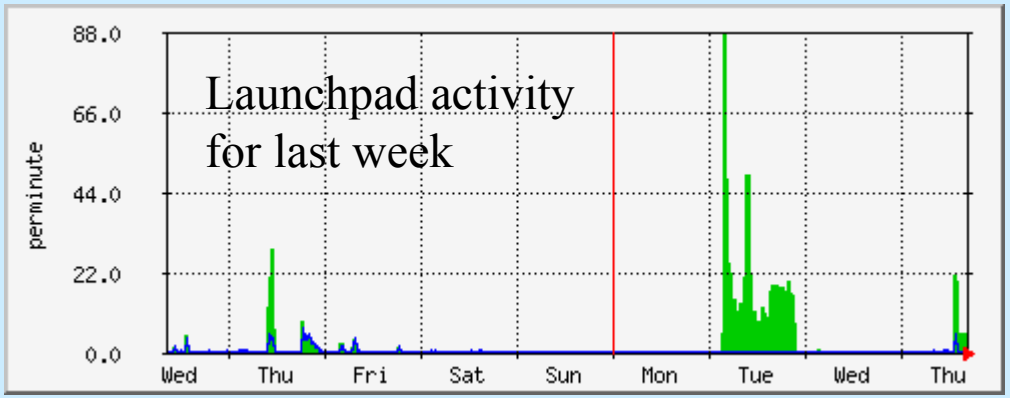
CDF Launchpad at FNAL



- Four former 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.
- Production executables built and tested with FroNtier.
- Widespread use will begin as new releases are built.
- Squid deployment at CDF processing centers in San Diego (SDSC), Bologna (CNAF), Karlsruhe (GridKa)
- Working to extend the use in CDF beyond calibration data to trigger and other static data needed offline.



Max HTTP reqs 88.0 req/min Average HTTP reqs 4.0 req/min Current HTTP reqs 0.0 req/min
Max HTTP fetches 7.0 req/min Average HTTP fetches 0.0 req/min Current HTTP fetches 0.0 req/min

CDF Testimonial

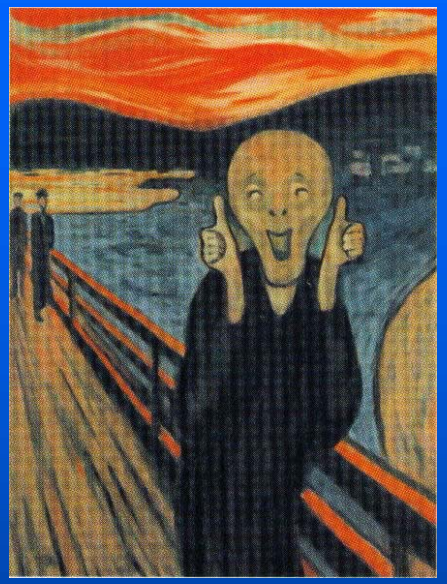
Hello Petar,

Finally I had more time to make some tests with the frontier server. I am the one who installed it at GridKA. The status after one month is the following. Besides me, another user used it the whole time and it did not give any trouble. Especially if you developing some code it helps a lot, since for a short test that runs only over a few hundred events the database access is quite a big factor. For example my executable was processing 100 events and it took with the oracle database more then 5 minutes. Most of the time waiting for the database. After switching to frontier it took less then 3 minutes. If it is important I can make some more detailed measurements with more statistics, but so far nearly 50% time saving is really great.

Since you are working on the physics validation, perhaps you are interested in some tests I did. I used a little job (J/Psi to ee)and compared the results for the oracle and frontier database access. I tried MC and real data and for both I got always the same result between oracle and frontier. I also tried the frontier version 1.2.0 and there were no differences.

I also have a short question, which is the best mailing list, to get all the information, updates and questions related to frontier. Since I would like to keep it running and here in Karlsruhe more and more people will be using it in the near future, after all the successful tests.

Cheers, Michael



CMS FroNtier

- CMS is interested in using FroNtier approach for offline and possibly some online DB access.
- Developed a working prototype to provide an interface for ORCA (CMS reconstruction) to HCAL 2004 testbeam conditions database.
- A prototype of a FroNtier “plug-in” for POOL is also working.

HCAL TB2004 DB – ORCA Prototype

- Conditions data for the 2004 HCAL testbeam run were loaded in to a simple schema designed for calibration data; pedestals, gains, and their errors using run ranges as IOV.
- An interface was prepared called FrontierCalib which provides the CalibObject within ORCA.
- Uses the same interface as Conditions DB for reading.
- Test example on next page.

Example of reading Calibration DB Data into ORCA-test program

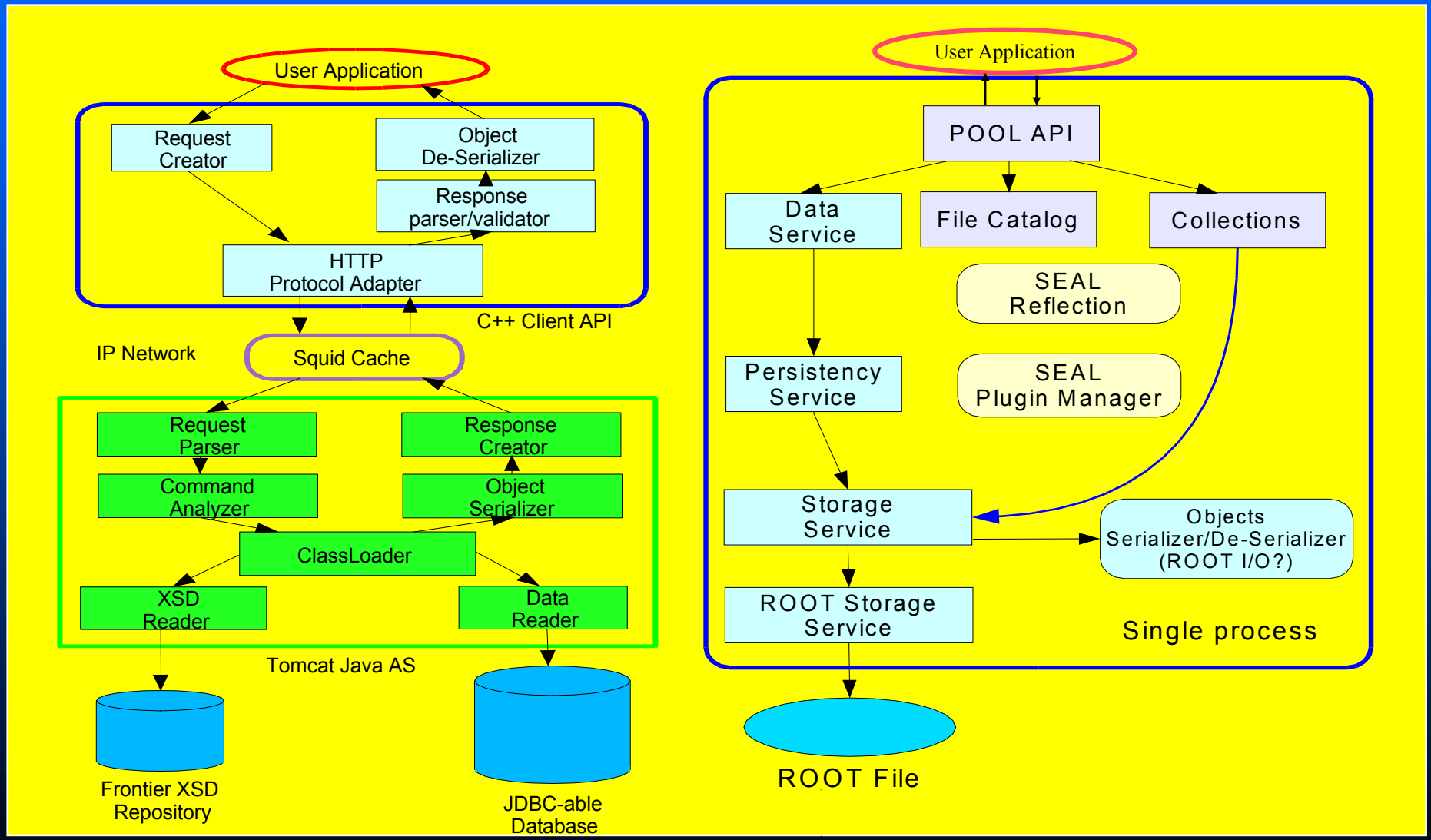
```
int main(int argc, char **argv)
 {
  try
   {
    std::string tag="TAG";
    long long runnumber=11800;
    std::vector<int> eta;

    FrontierCondObject* fc1=new FrontierCondObject(FrontierCondObject::GAINS,tag,runnumber);
    fc1->getIntData("EtaIndex",eta);
    std::cout<<"ETA="<<eta[0]<<std::endl;
    delete fc1;

    runnumber=11040;
    FrontierCondObject* fc2=new FrontierCondObject(FrontierCondObject::PEDESTALS,tag,runnumber);
    fc2->getIntData("EtaIndex",eta);
    std::cout<<"ETA="<<eta[0]<<std::endl;
    delete fc2;
   }
  catch(std::exception &e)
   {
    std::cout<<"Error: "<<e.what()<<std::endl;
    exit(1);
   }
  exit(0);
 }
```

Frontier & POOL (Simplified)

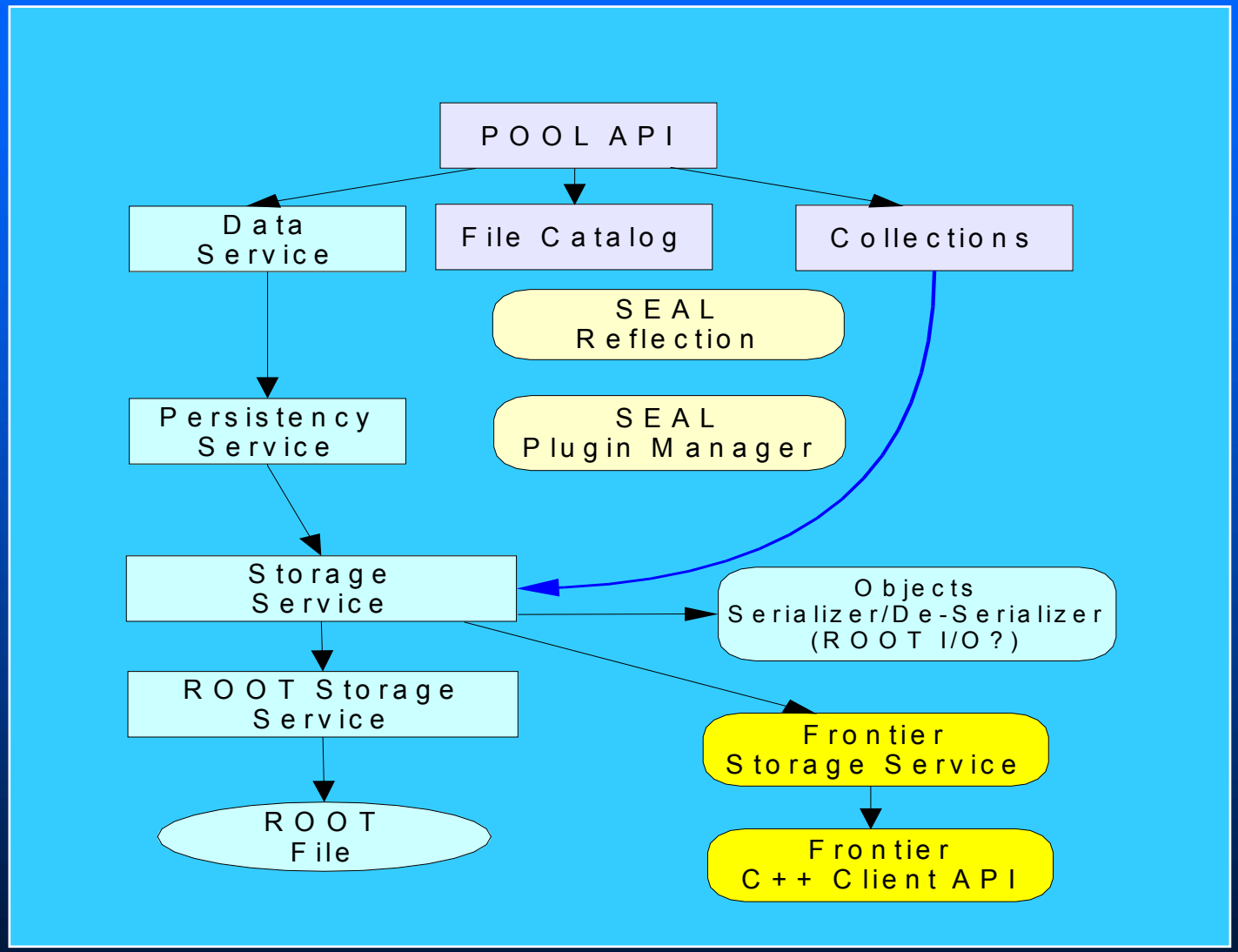
Sergey Kosyakov



Goal: access CMS calibration data using POOL via Frontier

- Create set of classes representing CMS calibration data (5 classes so far for HCAL Testbeam).
- Create Frontier XSD for each calibration class.
- Generate SEAL reflection dictionaries for these classes.
- Create plugin for POOL to request data from Frontier and map it into objects of calibration classes – the major challenge.

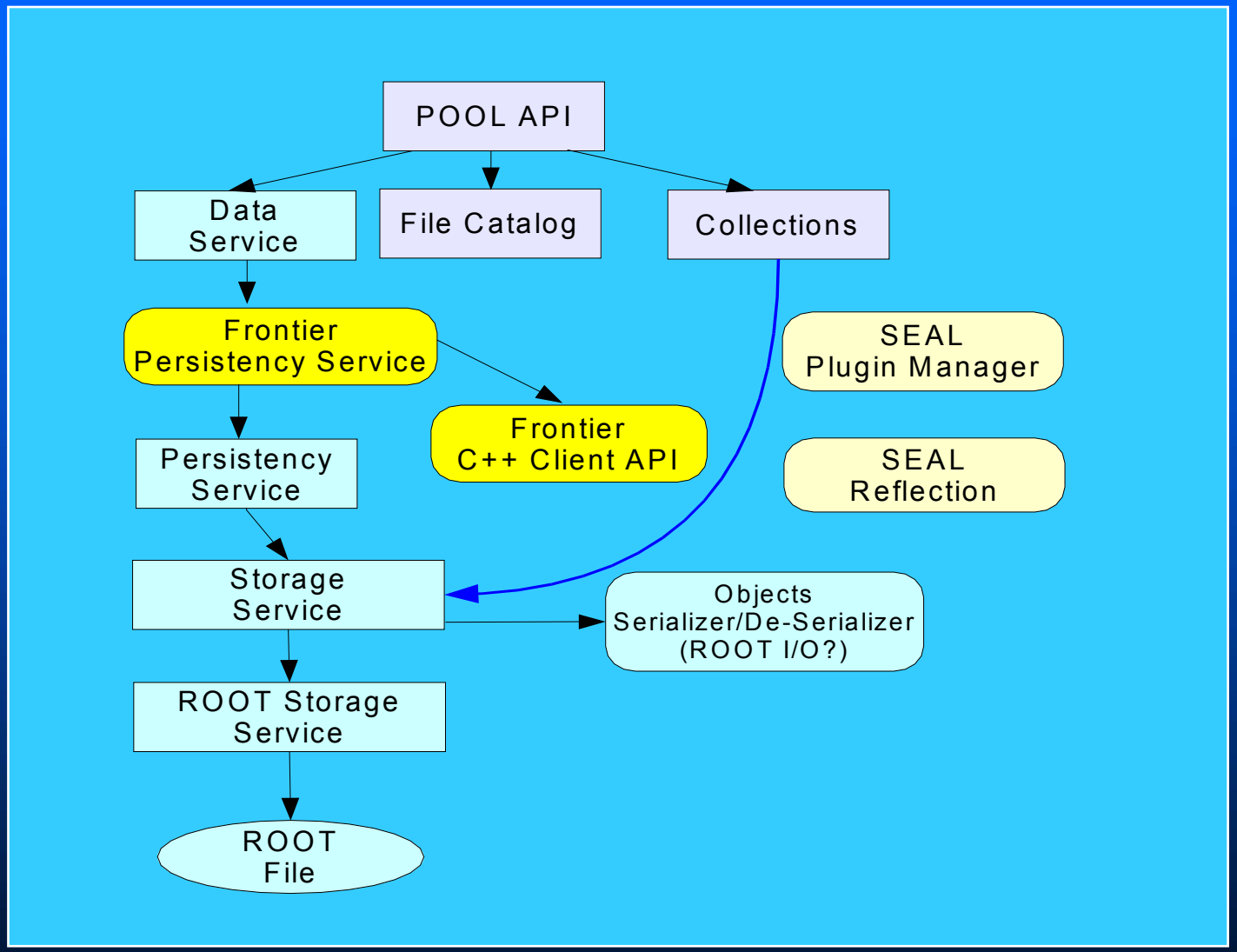
FrontierStorageSvc



FrontierStorageSvc

- Using Db*Imp
  - This approach would provide seamless integration with POOL
  - Problems:
    - » The framework expects special information in three containers: ##Shapes, ##Links, ##Params. The info is a set of strings which is generated when an object is written into storage.
    - » The framework expects the object to be serialized in predefined way and stored as a single column.
- Using IDb* interfaces
  - Reimplementation of FrontierStorageSvc using IDb* interfaces from StorageSvc sub-project: IOODatabase, IDbDatabase, IDbContainer, etc.
  - Problems:
    - » IDb* interfaces are bound to the same Db*Imp framework from StorageSvc. For exmaple, IDbCursor uses DbContainer, which call for ##Shapes, ##Links and ##Params containers. So this approach did not fix the problem.
    - » Making these containers is not straightforward and does not look like a good solution (compatibility problem, for example).
    - » Does not solve “single column” problem.

FrontierPersistencySvc



FrontierPersistencySvc

- IPersistencySvc can be used to control the DataSvc context
- FrontierPersistencySvc is an implementation of IpersistencySvc.
- It actually implements only `readObject(Token &token, DataHandler handler)`. All other requests are forwarded to default PersistencySvc implementation (which is obtained using the factory).
- A sample application which reads ROOT stored object from ROOT collection and then reads GainErrorAllByTagTimei from Frontier works.
- The collection interface does not work in this case, but collections don't really make sense for Frontier.
- Hidden obstacles? E.g. `readObject()` may be expected to change the state of other modules.

POOL via Frontier – general object selection problem

- POOL uses Tokens to identify objects. A token consists of ClassID (GUID), database name, container name, technology ID (e.g. ROOT_Key or Frontier), type ID and object ID. The last is unique object identifier and is a `pair<long,long>`.
- Frontier requests objects by server name, object name, object version, and 0 or more pairs key_name:key_value. The same object could be potentially requested in multiple ways.
- POOL's Token does not provide the ability for multiple key_name:key_value parameters. One of the approaches – use the container name as an encoded request string.

POOL via Frontier – serializer problem

- Frontier has it own platform-independent and network safe binary objects serializer and de-serializer
- POOL expects objects to be serialized in a particular way (Shape).
- Re-shaping Frontier objects for POOL would be inefficient, and hard to implement (because of ##Shapes, ##Links and ##Params containers the StorageSvc expects).

CMS Calibration Class Example

```
class GainErrorAllByTagTimeiStruct
 {
  public:
  int eta;
  double phi;
  int depth;
  double value;
  double sigma;

  virtual ~GainErrorAllByTagTimeiStruct(){}
 };

class GainErrorAllByTagTimei
 {
  public:
  std::vector<GainErrorAllByTagTimeiStruct> data;

  static const char *getGUID(){return "4AF7DAD1-E51E-D911-829B-000D616B4939";}

  void append(void *ptr)
   {
    data.insert(data.end(),(GainErrorAllByTagTimeiStruct*)s);
   }

  virtual ~GainErrorAllByTagTimei(){}
 };
```

Sample Application Snaplet

```
pool::IFileCatalog      *catalog;
...
pool::FrontierPersistencySvc *ipsvc=new pool::FrontierPersistencySvc(*catalog);
pool::IDataSvc *context=pool::DataSvcFactory::instance(ipsvc);

seal::SharedLibrary::load(seal::SharedLibrary::libname("FrontierDict"));
seal::SharedLibrary::load(seal::SharedLibrary::libname("SealSTLDict"));

context->transaction().start(pool::ITransaction::READ);

pool::Guid guid(GainErrorAllByTagTimei::getGUID()); // GainErrorAllByTagTimei
pool::Token tok;
tok.setTechnology(pool::FRONTIER_StorageType.type());
tok.setDb("http://edge.fnal.gov:8000/FrontierCMS");
tok.setCont("tag='TAG'&date_from='08/01/2004'&date_to='09/01/2004'");
tok.setClassID(guid);

pool::Ref<GainErrorAllByTagTimei> ref(context,tok);

std::cout<<"eta="<<ref->data[0].eta<<'\n';
std::cout<<"eta="<<ref->data[1].eta<<'\n';
std::cout<<"eta="<<ref->data[99].eta<<'\n';
```

Summary

- FroNtier is a multi-tier architecture providing clients 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 Beta test and users are excited by the results.
- CMS is interested in using the Frontier approach, and has successfully used it to access conditions data for ORCA.
- A working example of a POOL-Frontier "plug-in" has been built, and demonstrated using the CMS HCAL testbeam database.

References

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