A Proposal for a Generic Metadata Interface for the GRID

Birger Koblitz, Nuno Santos

3D Workshop, December 13th, 2004

Overview

- The scope of metadata on the GRID
- A generic definition of metadata
- Requirements on the functionality
- Interface definition
- Testing a Prototype Implementation
- Ideas for the Future

What is Metadata?

1. Definition:

Metadata is information on contents of files.
(File Metadata)

Also other information found in DBs necessary to run jobs on the grid, share problems:

- Grid authentication
- Overcoming firewalls
- Talking efficiently to DBs
- Replication
- Distributed updates?

2. Definition:

Metadata is all kind of data needed by jobs to run on the grid
(apart from what is in the files).

Hierarchy

Metadata needs a hierarchy to work well:

- Collect objects with shared attributes into collections

  ➔Allows queries on SQL tables

  (also other storage possible: XML-DB, DB-Files...)

  Analogy to file system (file metadata!):

  Collection ↔ Directory

  Object ↔ File

- Structure important for:
  - Structured searches
  - Schema handling
  - Distribution of databases

Experience

ARDA tested several Metadata solutions from the experiments:

- LHCb Bookkeeping: XML-RPC with Oracle backend
- CMS: RefDB
  (PHP in front of MySQL, giving back XML tables)
- Atlas: AMI
  (SOAP-Server in Java in front of MySQL)
- gLite (Alien Metadata)
  (Perl in front of MySQL parsing command, streaming back text)

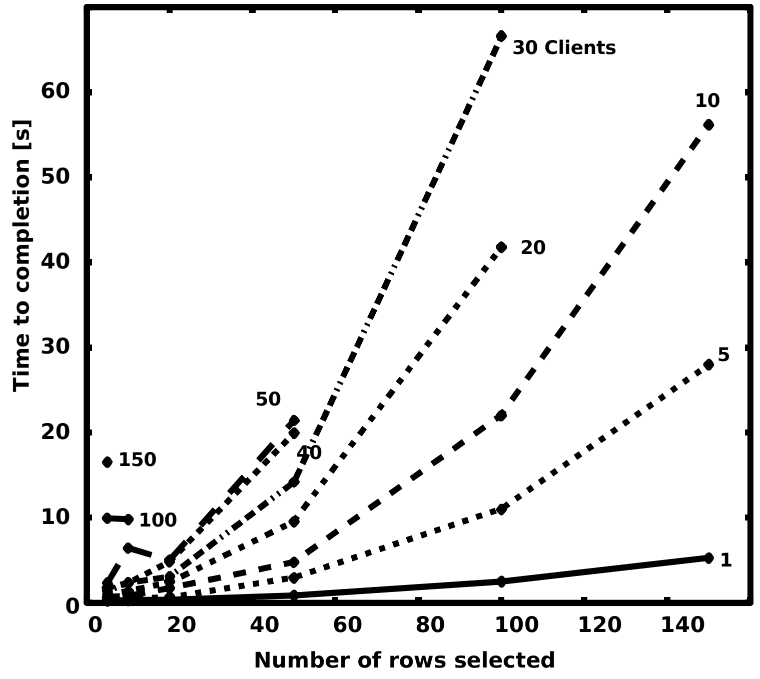
Learned a lot looking at existing implementations:

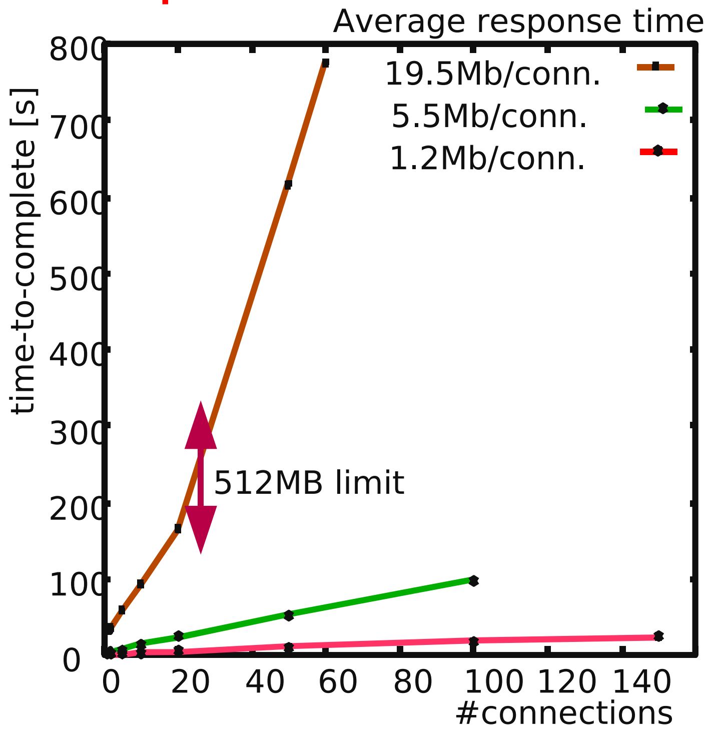
- Common pattern seen
- Implementations also share the same problems

Protocol: SOAP

Both AMI & RefDB ship responses in single XML package

➔They can't handle large requests





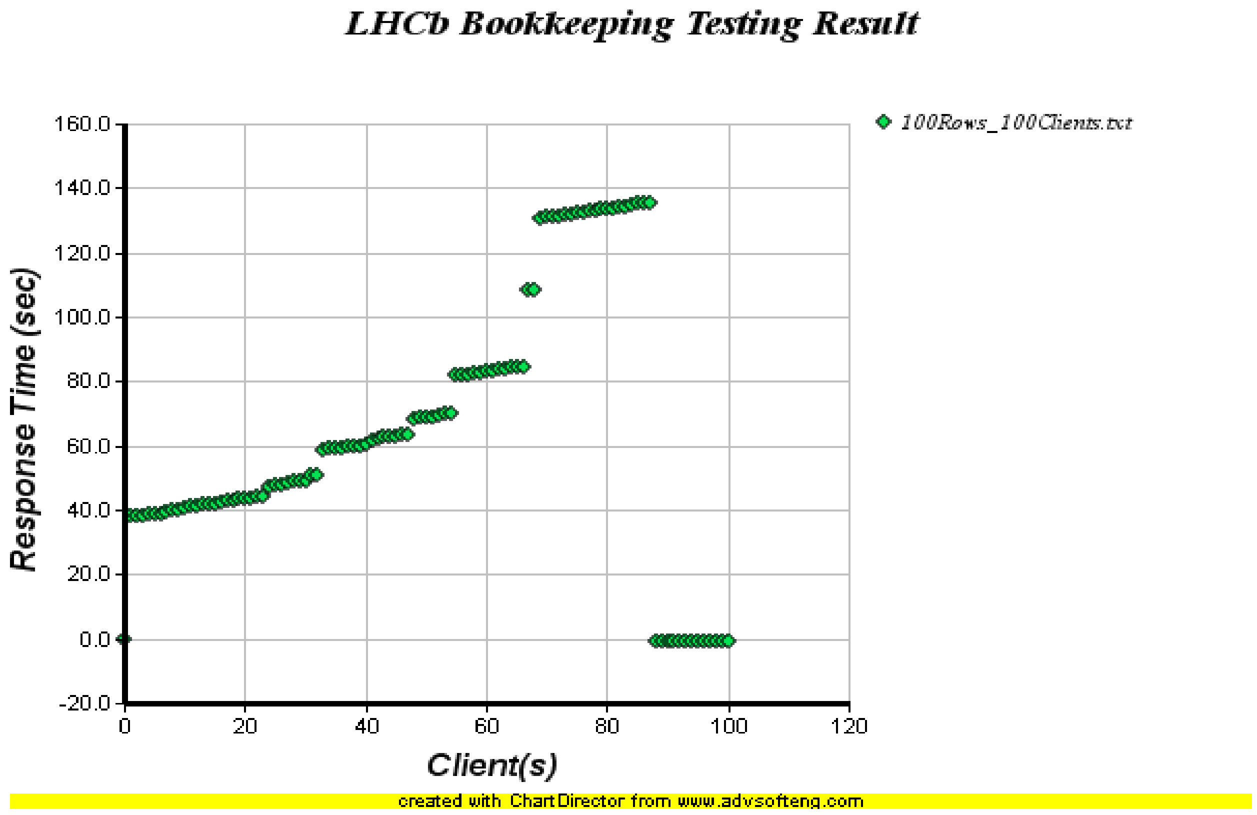
SOAP is particularly bad for Metadata:

- SOAP blows up data by factors 5-10
- SOAP for single, small queries

Metadata queries do require stateful connections with Streamed Data / Iterators as a response

Protocol: XML-RPC

LHCb uses XML-RPC(predecessor of SOAP):



Being also “one shot” query based, the solutions suffers from the same problems as the two based on SOAP

SOAP extreme

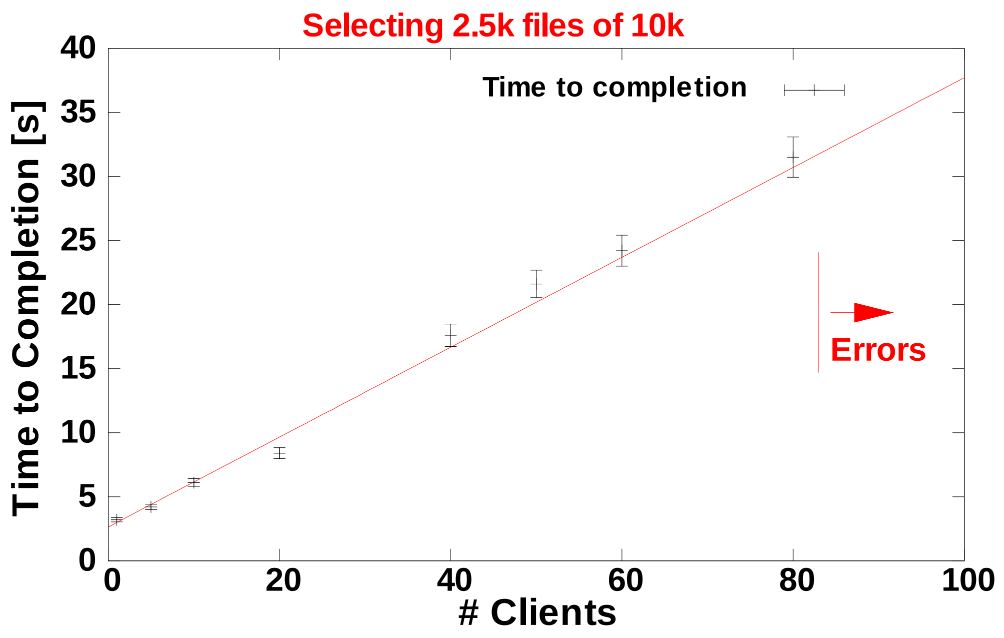
Snippet of client code from gLite Fireman:

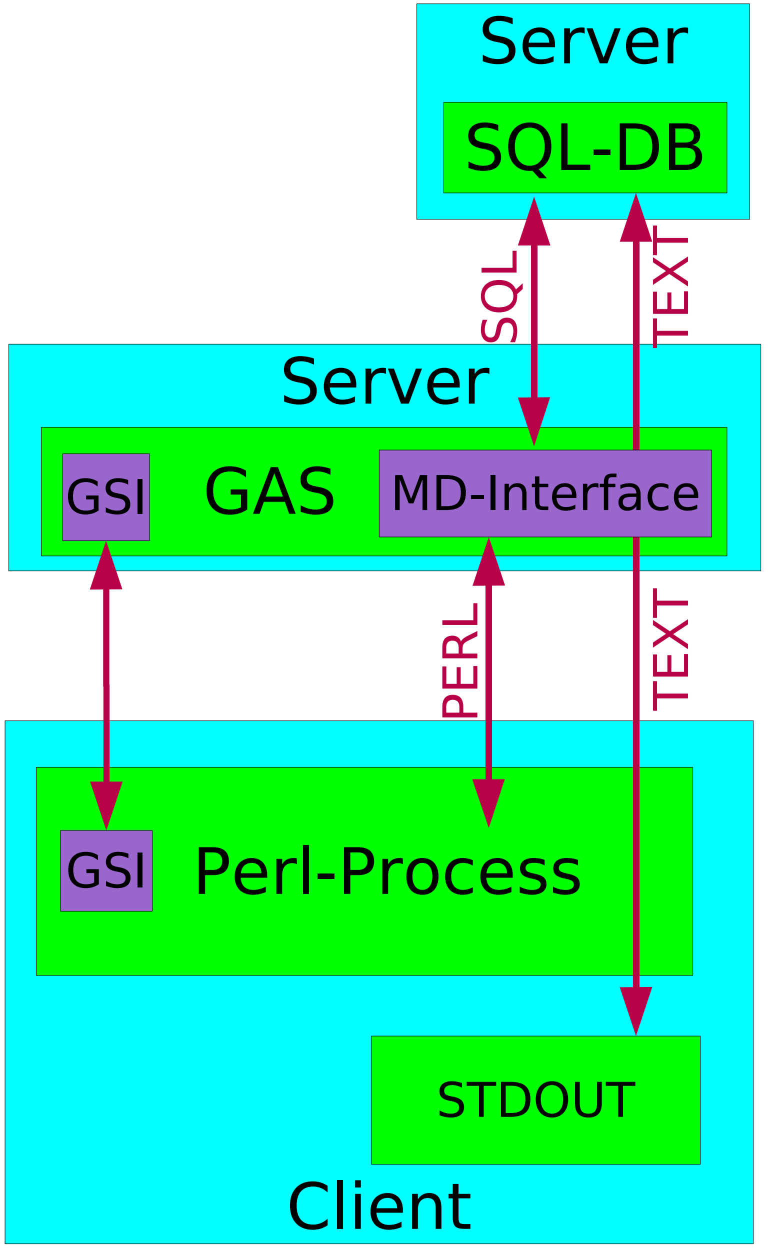
```
file::ArrayOfFCEntry fc_entry;
fc_entry.__size   = 1;
fc_entry.__ptr    = (file::glite__FCEntry**)
   soap_malloc(fileService.soap,sizeof(file::glite__FCEntry*));
fc_entry.__ptr[0] = file::soap_new_glite__FCEntry(fileService.soap,-1);
fc_entry.__ptr[0]->guid       = soap_strdup(fileService.soap,guid);
fc_entry.__ptr[0]->lfn        = soap_strdup(fileService.soap,lfn);
fc_entry.__ptr[0]->permission = 0;
fc_entry.__ptr[0]->lfnStat    = file::soap_new_glite__LFNStat(fileService.soap,-1);
fc_entry.__ptr[0]->lfnStat->type         = 0; // LFN
fc_entry.__ptr[0]->lfnStat->data         = 0; // Additional Information
fc_entry.__ptr[0]->lfnStat->modifyTime   = 0; // Use Default Value
fc_entry.__ptr[0]->lfnStat->validityTime = 0; // Use Default Value
fc_entry.__ptr[0]->lfnStat->creationTime = 0; // Use Default Value
file::file__createFileResponse out;

if(SOAP_OK != fileService.file__createFile(&fc_entry, out)){
        // TODO Exception Handling
        // Finalize service Object
      // finiFileService(&fileService);
      return -1;
}
```

Complex (bulk ?) SOAP calls difficult to use without API!
Several incompatibilities among SOAP implementations!

Streamed Data

gLite streams responses to the perl implemented shell





Schema Handling

Schema evolution not really tackled by current Metadata Catalogues:

- Not really important for production...
- Admin can setup/copy new tables (work on backend)...

RefDB and Alien don't do schema evolution at all. AMI, LHCb-Bookkeeping via admins adjusting tables.

EGEE design for gLite does not foresee schema changes nor schema discovery

For analysis, the following capabilities are mandatory:

- User must be able to discover schema
- User can setup/change schema
- Offer solution for problems with storage types

POSIX Metadata

POSIX defines extended attributes (Metadata) for files:

- Key-Value pairs associated with a file
  - Key: \0-terminated string
  - Value: Binary data of arbitrary length
- Copying a file copies metadata
- Metadata can be attached to directories (no inheritance)
- Metadata attached to inode (security)

Extended attributes are now widely used
(NTFS, NFS, EXT2/3 SCL3, ReiserFS, JFS, XFS)
Used with Namespaces for ACLs

Metadata searches not defined yet (No FS-Impl.):

- Windows Longhorn (2005)
- ReiserFS 5

Metadata on Linux

On ext3, XFS or ReiserFS, Linux supports extended attributes (file metadata)

```
koblitz@pcardabk:~/test$ touch a
koblitz@pcardabk:~/test$ attr --help
Usage: attr [-LRSq] -s attrname [-V attrvalue] pathname  # set value
       attr [-LRSq] -g attrname pathname                 # get value
       attr [-LRSq] -r attrname pathname                 # remove attr
      -s reads a value from stdin and -g writes a value to stdout
koblitz@pcardabk:~/test$ attr -s gen -V lepto a
Attribute "gen" set to a 5 byte value for a:
lepto
koblitz@pcardabk:~/test$ attr -s version -V 1.0 a
Attribute "version" set to a 3 byte value for a:
1.0
koblitz@pcardabk:~/test$ getfattr -d a
# file: a
user.gen="lepto"
user.version="1.0"
koblitz@pcardabk:~/test$ grep home /etc/fstab
/dev/hda5 /home ext3 defaults,acl,user_xattr,auto 0 0
```

Can we have a similar semantics on the Grid?

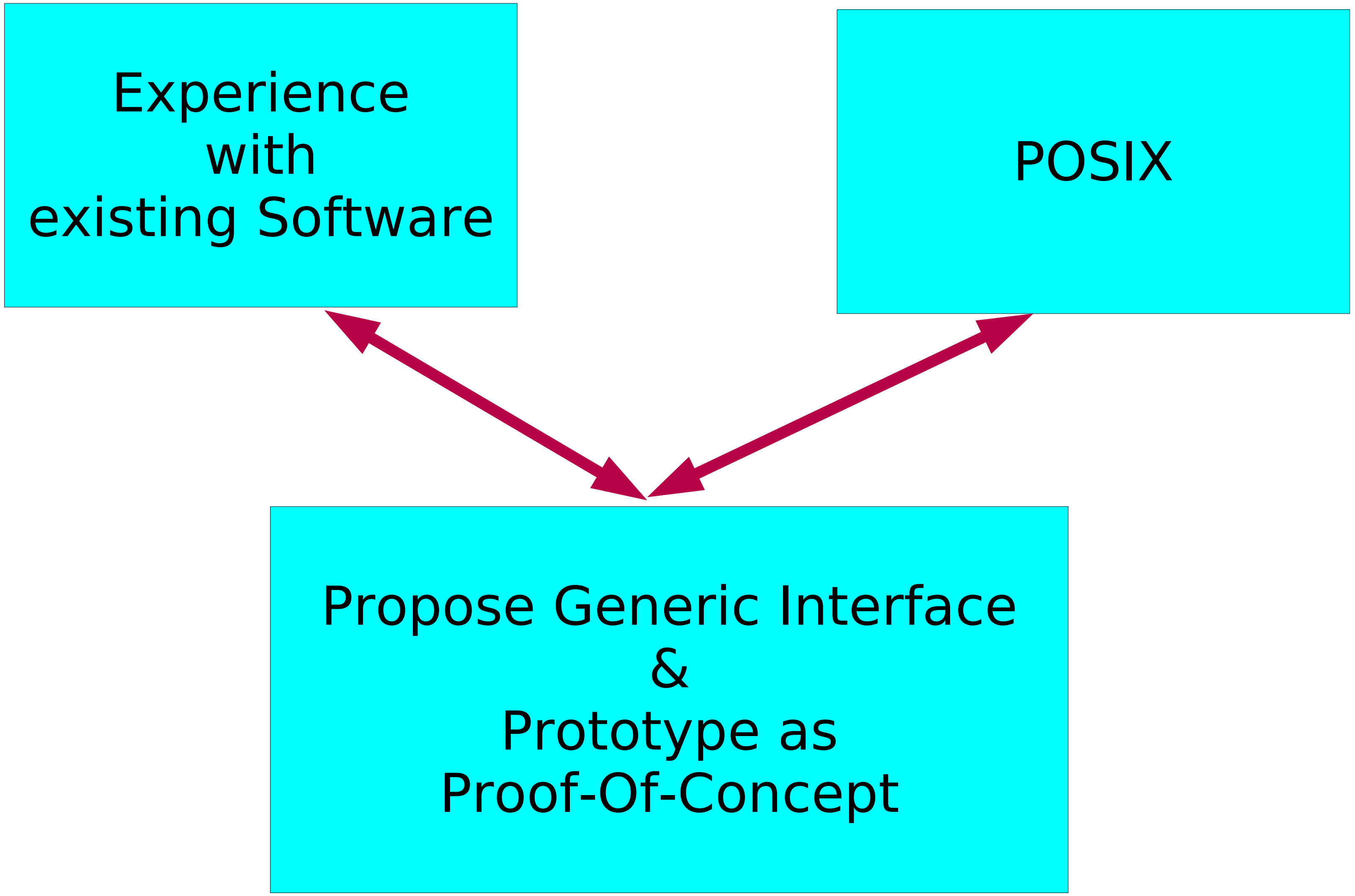
PS: API is POSIX, not the commands!

GRID Metadata

A possible Grid approach:

- Metadata attached to LFN
  - ➔ LFN is entry point to File-Catalogue, attached Metadata can be easily searched
- Files without LFN: GUID in special dirs
  - ➔ Otherwise problems with global searches
- Metadata for directories should provide default schemas/values for files
  - ➔ Easy schema copying
- Restrict values to ASCII strings
  - ➔ Backend is unknown: FileSystem/DB
- Need to define ways how to search for Metadata: Search restricted to (sub-)directories
  - ➔ Allows hierarchical databases, applicable for FS

Synthesis


Experience with existing Software
POSIX
Propose Generic Interface & Prototype as Proof-Of-Concept

Terminology

Define common terms, first

- **Metadata: Key-Value pairs**
  Any data necessary to work on the grid not living in the files
- **Entry: Entities to which metadata is attached**
  Denoted by a string, format like file-path in Unix, wild-cards allowed
- **Collection: Set of entries**
  Collections are themselves entries, think of Directories
- **Attribute: Name or key of piece of metadata**
  Alphanumerical string with starting letter
- **Value: Value of an entry's attribute**
  Printable ASCII string
- **Schema: Set of attributes of an entry**
  Classifies types of entries: Collection's schema inherited its entries
- **Storage Type: How back end stores a value**
  Back end may have different (SQL) datatypes than application
- **Transfer Type: How values are transferred**
  Values are transported as printeable ASCII

Interface I: Entries

The following protocol is proposed which clients talk to servers via sockets:

- **int addEntry(string entry, string type)**
  Adds a new entry to the catalogue
  Type can be “Collection” or “Entry”
  (extensions from implementation: Inheriting collections, views…)
  Returns integer errors code: MD_SUCCESS=0, MD_ERR_NOENT, MD_ERR_PERM, MD_ERR_INT

- **int addEntries(list<string> entries,**
  list<string> types)
  Entries and types lists for bulk insertion into catalogue
  Very difficult to implement on backend if transaction safe,
  Implementaton may limit updates to one collection

- **int removeEntries(string pattern)**
  Pattern for intuitive bulk deletion

With R. Rocha(gLite), V. Pose, N. Santos

Interface II: Attributes

Schema management and metadata reading/writing:

- **int addAttr(string entry, list<string> keys, list<string> types)**

  Adds a new attribute (key) to an entry (collection)
  Implementation may have only per-collection schema
  Types is contains the desired storage type (backend dependent)

- **int setAttr(string pattern, list<string> keys, list<string> values)**

  Bulk setting of the keys of all entries matching pattern to new values.

- **int clearAttr(string pattern, string key)**

  Resets the keys of all entries matching the pattern
  Application will get empty string if asking for value. Behaviour in queries like NULL in SQL.

With R. Rocha(gLite), V. Pose, N. Santos

Interface: Retrieving data

The Bulk transfers to client are done through session handlers and iterators on the backend:

- **Handler getAttr(string pattern, list<string>keys)**
  Returns values for all keys of the entries matching pattern

```
struct Handler {
    handle_t handle;
    list<string> values;
    int error;
    bool last;
}
```

  The values contain names of matching entries end the data:
  - ➔ Client knows semantic

- **Handler getNext(handle_t handle)**
  Returns the next bunch of values

- **Handler listAttr(string entry)**
  Lists all attributes of an entry

With R. Rocha(gLite), V. Pose, N. Santos

Interface:Searching

Physics analysis needs powerful tool to find entries, more than attribute-value matching:

- **Handler find(string pattern, string query)**
  Returns all entries (excluding collections) matching the pattern and fulfilling the query

```
Example query:'tracks > 10 and sin(p_angle) <0.5 and trigger & 2'
```

  Query needs to be parsed:
  - SQL injection prevention
  - Separate user & system namespace: events → “user:events”
  - Interpret for different backends

- **Handler listEntries(string pattern)**
  Returns all entries and collections matching the pattern and giving their type

With R. Rocha(gLite), V. Pose, N. Santos

Protocol Choices

Presented interface is SOAP compatible:

- Fulfil formal requirements on EGEE
- SOAP implementation is work in progress: Efficient session handling & iterators very demanding

So far implemented as TCP/IP streaming:

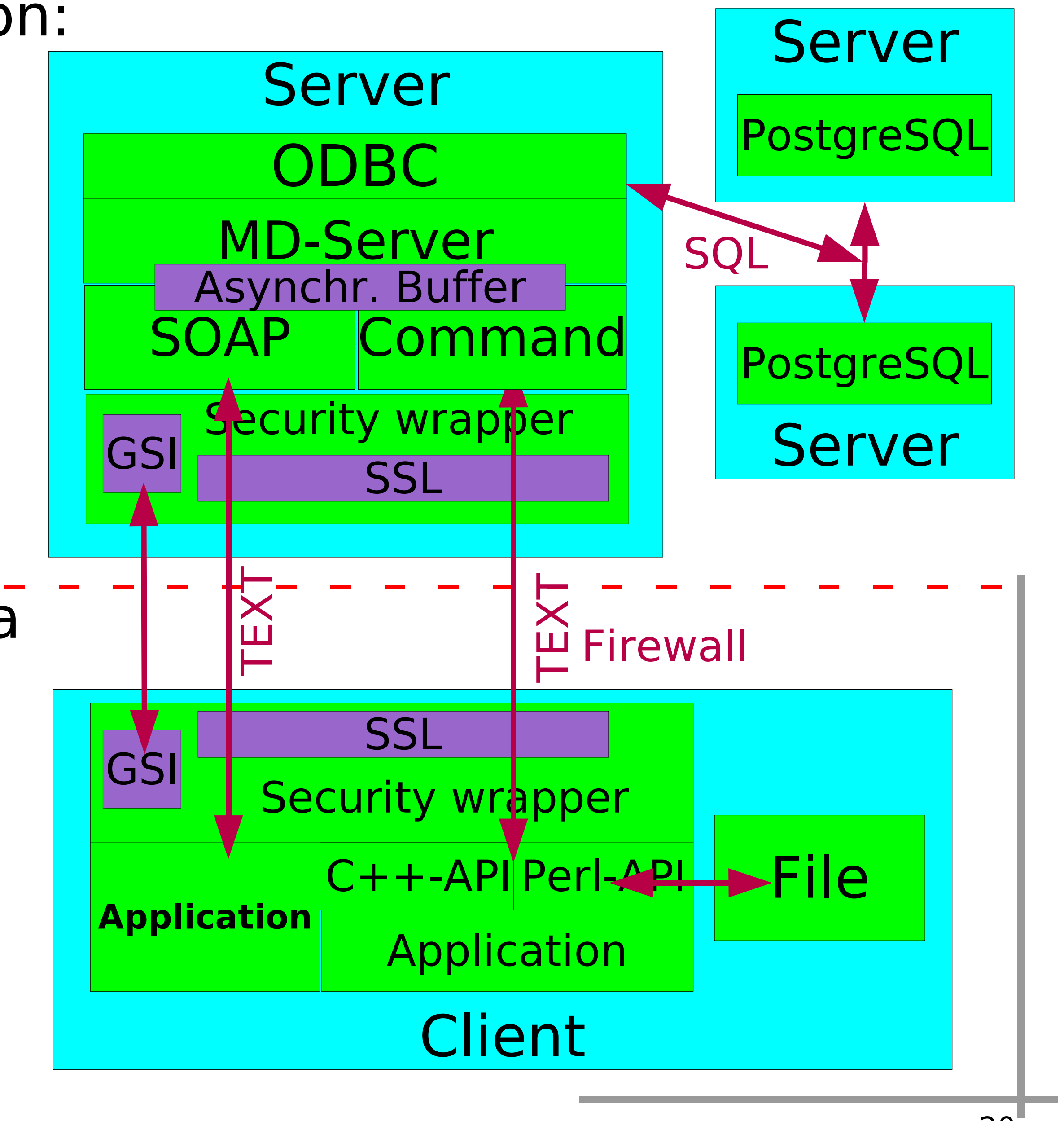
- Use plain text (ASCII)
- Query consists of one line of command
- Response returns 1 line of return status (OK/Error) and result line by line (and EOT at end)
- Result is in ASCII, user needs to encode/decode
  - Commands are Line of ASCII, e.g:
  - `getattr file(s) key1 key2...` Returns value of keys
- SSL for authentication and encryption implemented
- GSI authentication in progress

Implementing client APIs very simple!

Prototype

Prototype Implementation:

- Multi-threaded C++ server in front of PostgreSQL
- Streams responses asynchronously
- Uses ODBC as RDBMS abstraction Layer: ODBC-types
- Access restrictions via ACLs
- Bison/flex parser for queries
  →Other backends
  →Query validation
  →Security



API examples

Python example:

```
client=mdclient.MDClient('localhost', 8822)
client.getattr('/testdir/t1', ['eventGen', 'events'])
while not client.eot():
  res, file, values=client.getAttrEntry()
  if not res:
    print file, values
  else:
    print "Error: %d" % res
```

Same in C++:

```
AttributeList attributeList(2);
list< string > attributes;
attributes.push_back("eventGen");
attributes.push_back("events");
if( (res=getAttr("/home/koblitz/*", attributes, attributeList)) == 0){
  cout << "  Result:" << endl;
  while(!attributeList.lastRow()){
    vector< string > attrs;
    string filename;
    attributeList.getRow(filename, attrs);
    cout << "File: >" << filename << "<" << endl;
    for(int i=0; i< attrs.size(); i++)
      cout << "  >" <<  attrs[i] << "<" << endl;
    cout << endl;
  }
}
```

Example Session

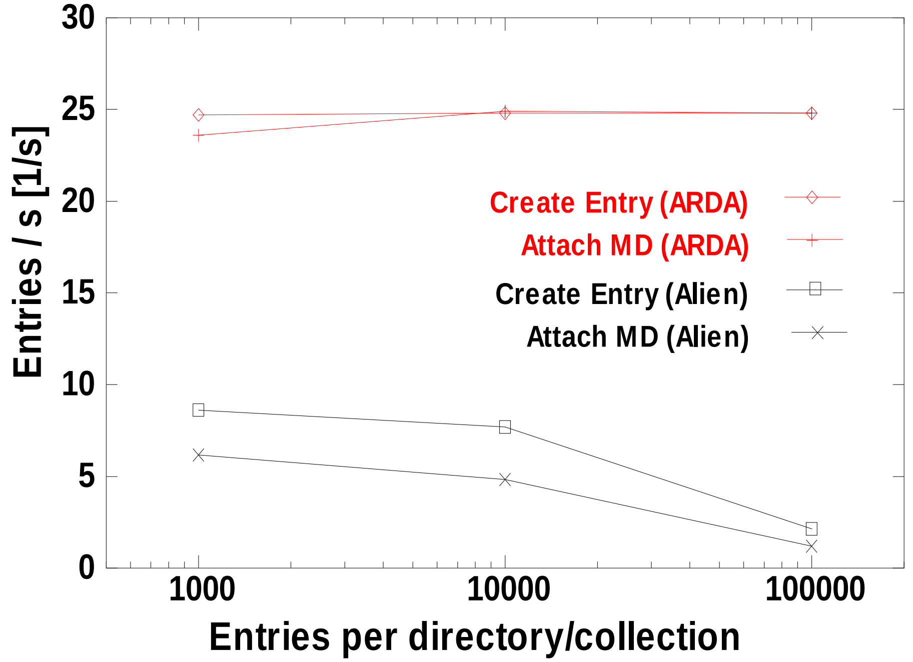
```
koblitz@pcardabk:~/mi$ ./mdterm
Connected to DB
Query> getattr /home/koblitz/a gen
  >select table_name from masterindex where directory='/home/koblitz';<
  >select gen from dir1 where file='a' and gen is not null;<
0
lepto
Query> addattr /home/koblitz version float
  >select table_name from masterindex where directory='/home/koblitz';<
  >alter table dir1 add version int;
0
Query> setattr /home/koblitz/a version 1.0
  >select table_name from masterindex where directory='/home
  >select version from dir1 where version is not null limit 1;<
  >alter table dir1 add version varchar(256);<
  >insert into dir1 (file, version) values ('a' ,'1.0');<
  >update dir1 set version='1.0' where file='a';<
0
Query> getattr /home/koblitz/a version
  >select table_name from masterindex where directory='/home/koblitz';<
  >select version from dir1 where file='a' and version is not null;<
0
1.0
Query> getattr /home/koblitz/b version
  >select table_name from masterindex where directory='/home/koblitz';<
  >select version from dir1 where file='b' and version is not null;<
2
Query> quit
```

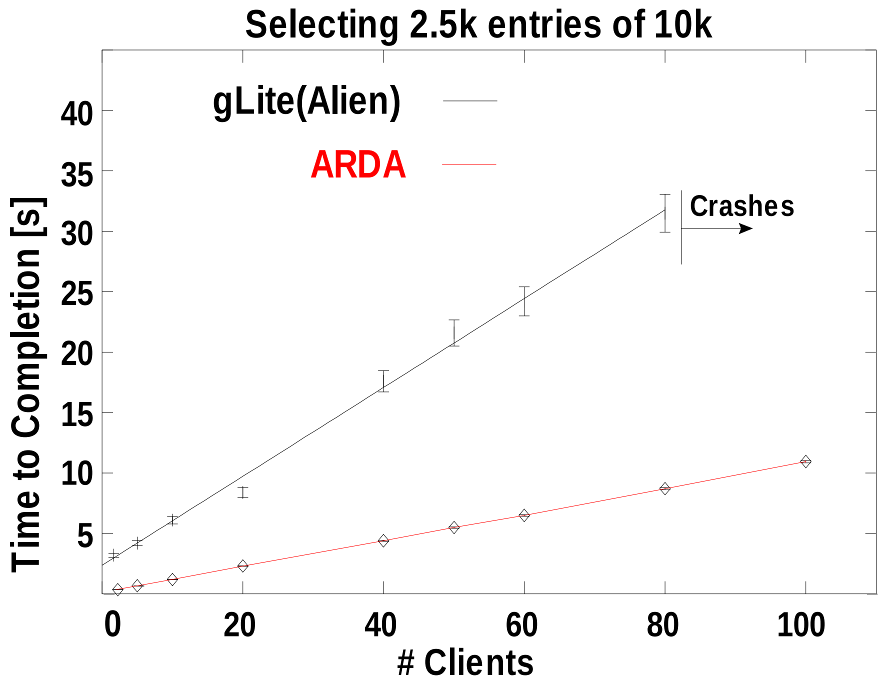
```
metadata=# select * from dir1;
 file |  gen    | events
------+--------+--------
 a    | lepto  |    101
 b    | phytia |    101
 c    | lepto  |  20001
 d    | lepto  |  30001
```

```
metadata=# select * from dir1;
 file |  gen    | events | version
------+--------+--------+---------
 b    | phytia |    101 |
 c    | lepto  |  20001 |
 d    | lepto  |  30001 |
 a    | lepto  |    101 | 1.0
```

Reality Check

Good experiences with ARDA prototype using streaming:

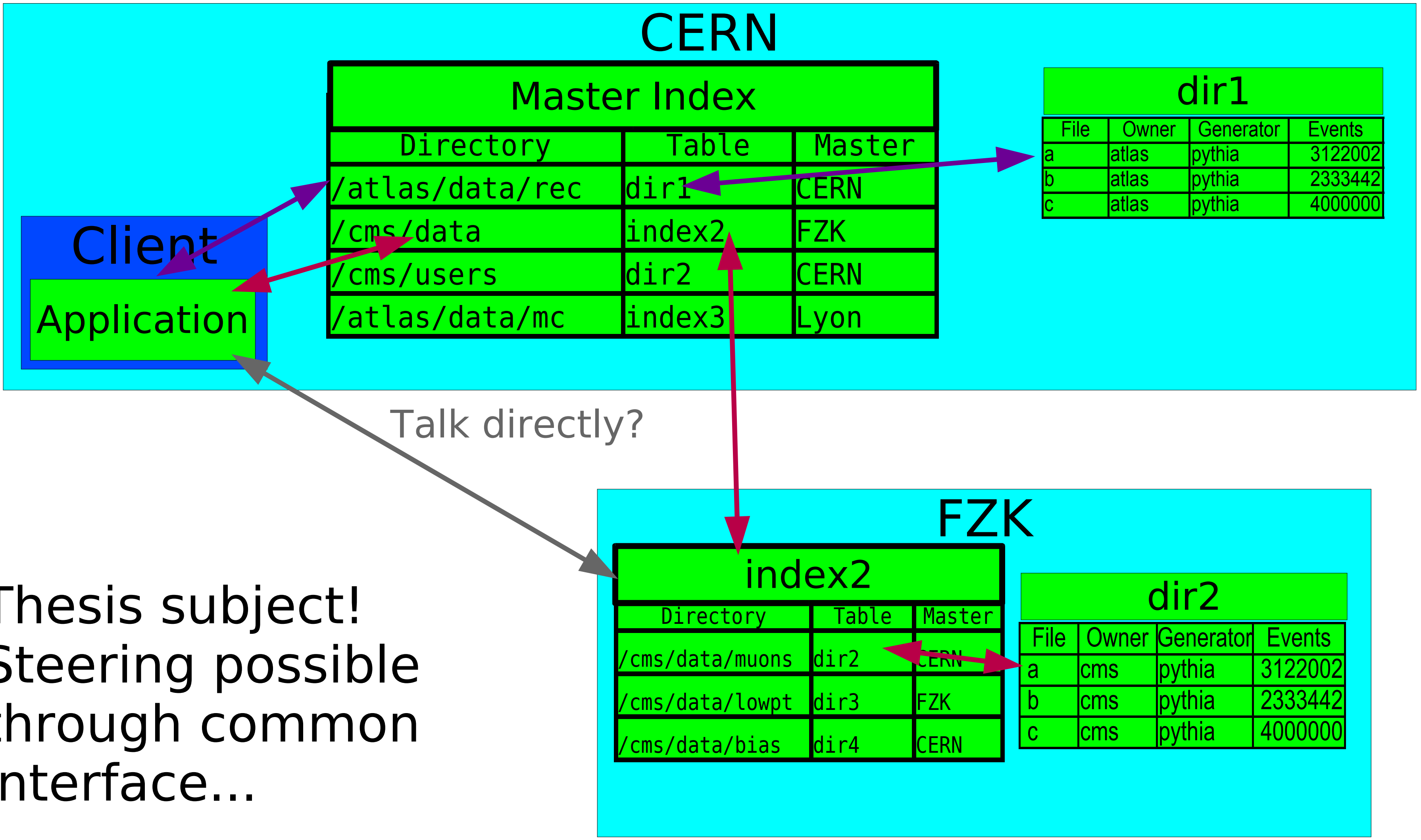




ARDA prototype even faster than AliEn!
Now very stable after tuning.
→High performance seems to require streaming
Possible with iterators?

Distributed Metadata Ideas

Use PostgreSQL/Oracle per-table replication with different masters, make use of hierarchy:



Master Index

Directory	Table	Master
/atlas/data/rec	dir1	CERN
/cms/data	index2	FZK
/cms/users	dir2	CERN
/atlas/data/mc	index3	Lyon

dir1

File	Owner	Generator	Events
a	atlas	pythia	3122002
b	atlas	pythia	2333442
c	atlas	pythia	4000000

index2

Directory	Table	Master
/cms/data/muons	dir2	CERN
/cms/data/lowpt	dir3	FZK
/cms/data/bias	dir4	CERN

dir2

File	Owner	Generator	Events
a	cms	pythia	3122002
b	cms	pythia	2333442
c	cms	pythia	4000000

Thesis subject!
Steering possible through common interface...

More Ideas

To be more generally useful:

- Create user indices with views:

```
create view dirs as select generator, file from dir1
    union
select generator, file from dir2;
CREATE INDEX gen_index ON dirs(generator);
```

- Or via inheritance:

```
createdir “/atlas/data/2008” INHERITS “/atlas/data/”;
```

Copies data schema, select on “/atlas/data” gives also 2008 data.

(Both features available in PostgreSQL/Oracle)

Implementation done, now experimenting

Conclusions

- Many problems understood studying metadata implementations of experiments
- Common requirements exist
- ARDA proposes generic interface to metadata:
  - Retrieving/Updating of data
  - Hierarchical view
  - Schema discovery and menagement
- Seems possible to create generic metadata catalogue suitable for very different metadata
- But room for special database solutions exist
- Design and Implementation certainly challenging
  ➔Metadata experts need to work together
- Started Collaboration with LHCb, large scale test till January 2005

http://project-arda-dev.web.cern.ch/project-arda-dev/metadata/