# R&D Proposal from the Session “Persistence, Data Handling models and Databases”

# Topics for R&D in Databases

Alexandre Vaniachine

SuperB Computing R&D Workshop

March 9-12, 2010

IUSS, Ferrara, Italy

# Conditions DB: Requirements Gathering

- Are there Conditions DB use cases beyond these three?
  - Retrieve conditions (slow control data) for the study of the time evolution
    - for understanding the detector, not for event analysis
  - Retrieve predefined set of validated conditions/calibrations
    - aka user-defined set of tags
  - Retrieve the very latest conditions/calibrations
    - aka HEAD tag

- What database is required for conditions:
  - SQL RDBMS or key-value pair database?
  - Single database technology or hybrid?
    - e.g. plus data in ROOT files
  - Two-tier (client-server) model or data caching is required?

- What scalability limits are acceptable?
  - Should we put “database server” on every node?

# Conditions DB: Evaluation of Technologies

- How to prevent production/analysis from been affected by continuous updates to Conditions DB?
  - How user can get the consistent view of the "latest" conditions locked for reading at the moment when the analysis started
  - How reprocessing is protected from accidental updates of the "tagged" conditions for the duration of the campaign?
  - How reproducibility of the retrieved conditions is assured?
- Does Frontier/Squid technology for data caching satisfy the requirements?
  - Can Frontier/Squid be used having the "live" database as the source?
  - What is the scalability limit set by the (single-threaded) Squid server?
    - What prevented hierarchical deployment of Squids to achieve scalability?
  - Is extra latency due to Frontier/Squid cache consistency checking acceptable?
    - For each query there is a cache consistency checking query to remote master DB, e.g. Oracle
- Does COOL/CORAL Conditions DB implementation satisfy the requirements?
- Binding of conditions to datasets
  - Can ATLAS approach for Conditions DB "slicing" be extended to satisfy the requirements of SuperB analysis?
    - "Slicing" is similar to the software release build, with user analysis code is build on top of the SW Release
    - Similarly, official Conditions DB "slice" build and validated centrally, then updated with extras for user analysis
  - Datasets with rare events will have sparse conditions
    - Could that present a bottleneck?
  - How to use event timestamps known from the first-pass processing?
    - Can SuperB benefit from the SciDB array model?

# “Databases” R&D topics overlapping with “Persistence and data handling models”

- What are the requirements for “bookkeeping”?
  - Collections and provenance
  - Dataset selection
  - Coherence with conditions

- Event metadata vs. conditions
  - Does ATLAS in-file metadata approach satisfies the requirements?

# “Databases” R&D topics overlapping with “Persistence and data handling models”

- Top-down model for data analysis: requirements
  - Can SuperB learn useful things from looking at the SciDB?
    - SciDB is being designed for analysis of petascale datasets

THE SMARTEST PEOPLE ON THE PLANET ARE PUTTING THEIR HEADS TOGETHER.

SciDB Open Source DBMS for Scientific Research

**HEART BEAT:** Jan 6, 2010: Open Letter to the SciDB Community

Nov 3, 2009: First public release expected in March 2010

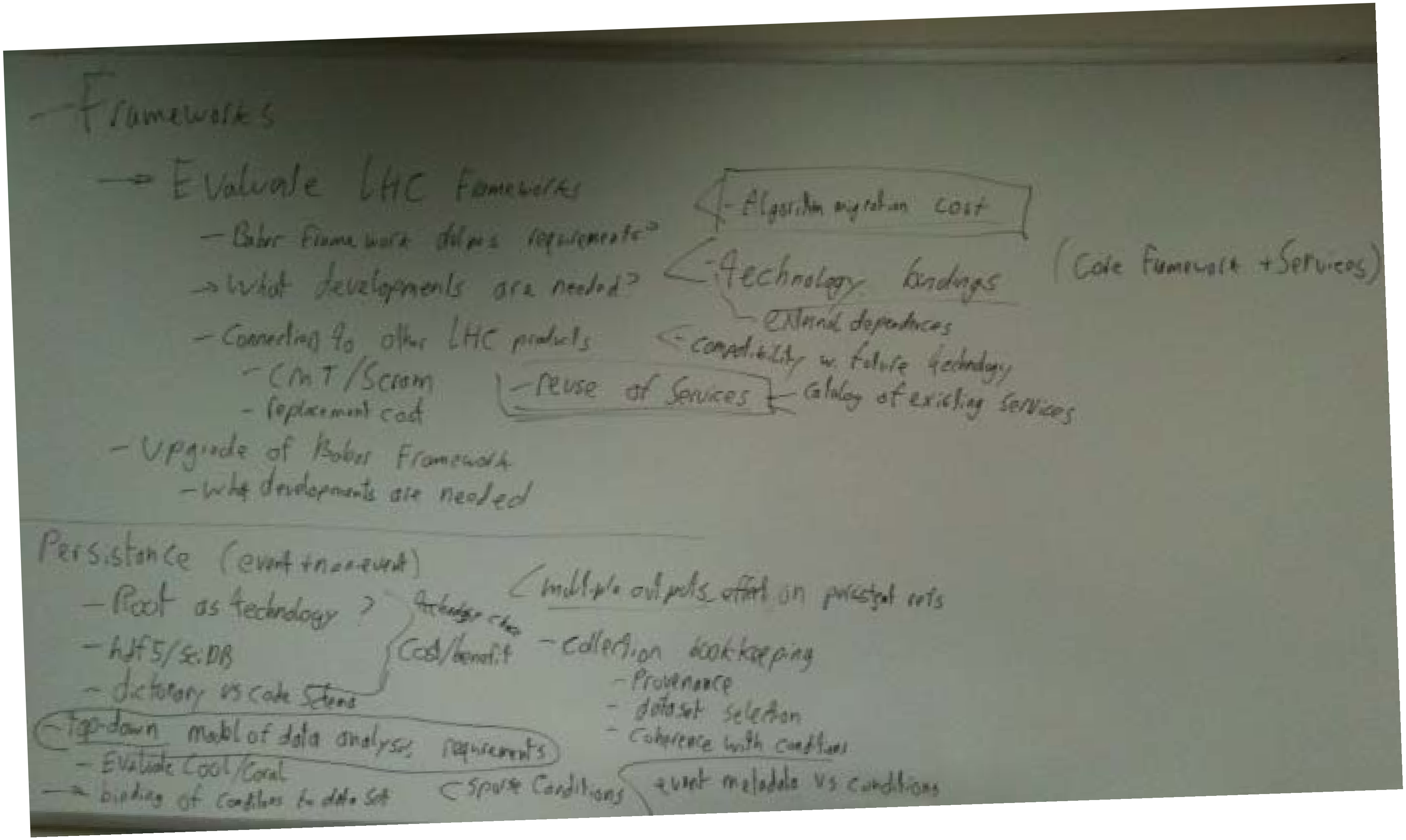
Aug 24, 2009: First public SciDB demo at VLDB'09 (poster, paper)

# Supporting Materials

# Details on ATLAS Conditions DB “Slicing”

- A 1.4 GB “slice” of Conditions DB covers the data taking period of $0.23 \cdot 10^7$ s, which is about one quarter of the nominal LHC year
  - We are not expecting multi-TB “slices” any time soon
- ATLAS “slicing” is done using dozens of processes run in parallel
- “Slicing” (per multi-hour run) is much more efficient then retrieval of the conditions for each event, which is done from the jobs in ~10-min intervals

# Notes from Break-out Discussions



- Frameworks
  - → Evaluate LHC Frameworks
    - Babar Framework fulfills requirements?
    - → What developments are needed?
    - Connecting to other LHC products
      - CMT/Scram
      - replacement cost
  - Upgrade of Babar Framework
    - what developments are needed

< - Algorithm migration cost

< - technology bindings
- External dependences

< - compatibility w. future technology

- reuse of Services ← Catalog of existing services

(Core Framework + Services)

Persistence (event + non-event)

- Root as technology ?
- hdf5/SciDB
- dictionary vs code Schema

Technology choice
Cost/benefit

< multiple outputs effect on persistent sets

- collection bookkeeping
  - Provenance
  - dataset selection
  - Coherence with conditions

- top-down model of data analysis requirements
- Evaluate Cool/Coral
- → binding of conditions to data set

< Sparse Conditions

event metadata vs conditions

# IDEAS FOR SUPER-B

# What can be inherited?

Very conservative strategy

- **Unfortunately very little because:**
  - None of the databases mentioned in this talk was designed/implemented as an independent or portable product
    - This has never been encouraged/discussed in BABAR
  - Very little documentation
  - Dissipation of the expertise with specific designs/implementations (and getting worse each year)
- **For things which can be reused the waiting time matters:**
  - Waiting for a couple more years won't make it easier
  - If anything has to be inherited directly – it must be done today, not tomorrow
- **What to consider:**
  - Experience, models, approaches (ROOT as a data modeling language?)
  - Probably Conditions/DB design, interfaces and some implementations (MySQL+ROOT); probably Config/DB
  - Other code (yet to be identified by contacting developers)

# What can be done better?

Moderate strategy

- **R&D topic: "Distributed Databases"**
  - Better integration of databases into the Computer Model; provisions for consistency and data integrity in a distributed environment; Data Provenance "light"?
  - Design database applications (conceptual models, interfaces, protocols) as if they were to be used in a distributed environment, even if that's not a requirement now.
  - Investigate design scenarios to increase **mobility** of applications so that they would less depend on a specific database environment
  - Consider specialized servers/services (possibly Web based) in front of databases (see next slides)
  - Consider cooperative data caching on many core architectures (see next slides)

- **R&D topic: "Abstraction layers, Interfaces, Tools"**
  - Study various options for decoupling database applications from being directly dependant (by a design) on underlying persistent technologies; proper abstraction layers and interfaces.
  - Investigate a possibility of using standard portable data formats in the interfaces, such as XML, JSON

# Extreme ideas (1)

Progressive strategy

This is inspired by a success of XROOTD

- **Investigate a possibility of developing a specialized server (service architecture) for the Conditions/DB (and alike):**
  - Basically, this is the same concept as "..*extra level of indirection solves all problems..*"; this is how most Web users interact with back-end databases behind Web servers; consider this as a model
  - Decouple client code from any technology-specific libraries (except those which are distributed along with the application); less requirements for an exec environment
  - Redefine the "**atomicity**" of database operations at the domain level, not at the level of a persistent technology
  - Implement authorization models which would suit specific database application domains better (not trying to fit into the low-level ones enforced by technologies);
  - Optimize operations with the "backend" databases; more options to implement better caching of results (see an example of caching interval requests in MySQL)
  - Correct blocking of clients during data distribution/synchronization on the server's backend; no service shutdown during the distributed data synchronization operations
  - Dynamic load balancing, cleints' redirection and more…

# Extreme ideas (2)

Progressive strategy

- **An interesting variation of the idea from the previous page would be to use Web services as an indirection layer between client applications and databases:**
  - Should work for both reading and updating databases
  - Will increase the **mobility** of applications in a distributed environment
  - Leverage of certain interesting Web technologies, such as portable object serialization using XML and JSON, caching, etc.

- **The idea was inspired by the US/DOE/SBIR proposal:**
  - "Customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases", Tech X Corporation
  - http://www.sc.doe.gov/sbir/awards_abstracts/sbirsttr/cycle25/phase2/087.htm

# Extreme ideas (3)

Targeting an explosion of parallelism

- **Investigate a possibility of implementing a dynamic cooperative caching of the read-only data by a group of application processes run on a multi- or many-core system:**
  - Consider a scenario of x1000 processes run on a many-core system and processing in parallel different events of the same event collection; each process would probably need to bring the same data from some remote database server (x1000 similar requests to that server); what would happen to the server?
  - The processed could cooperate to cache (on a local disk or in memory) the data read from a remote database server either by delegating one of them as a caching server, or launching a dedicated server, or employing a distributed logic to use a shared disk as a local data cache
  - Will decrease the load onto database servers
  - Will decrease (due to caching) the service latency for applications in a distributed environment
  - Should be easy to do for read-only access to databases