A Git-based Conditions Database backend for LHCb

M. Clemencic *on behalf of the LHCb Collaboration*

July 10, 2018

CERN - LHCb

slides

Introduction

- LHCb has been using COOL/SQLite based CondDB for 12 years
- We wanted to investigate alternative technologies for LHC Run 3
- A Git based implementation was developed and commissioned

Table of contents

The Conditions Database

Definitions

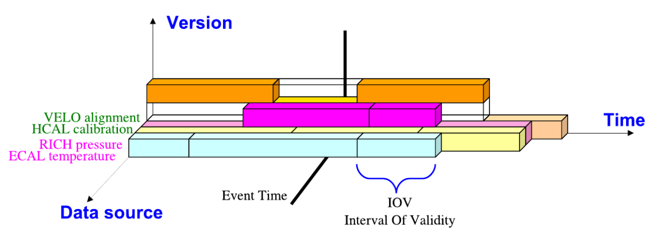
Condition:

Time-varying non-event data required for the correct reconstruction of event data.

Conditions Database (CondDB):

A database for recording and retrieving conditions.

Conditions Database



- 3 dimensions of condition values
  - source/id
  - version
  - time evolution (IOVs)

COOL

- CERN-IT developed Conditions Database library
    - API matching our CondDB model
    - optimized for standard access to conditions
    - multiple backends (via CORAL)
    - Python bindings (via ROOT)

LHCb Conditions Databases

- Same format for detector description and condition data
  - currently XML files
- Small footprint
  - full history in a few GBs
  - deployed as files on CVMFS
- Storage is partitioned in different SQLite files by type of data
  - detector description
    (2 dimensions, no time evolution)
  - alignment and calibrations
    (all 3 dimensions)
  - environment information
    (2 dimensions, no versions)

Git CondDB Design and Implementation

Choosing the Technology

- Old system had limits
  - not ready for multi-threaded applications
  - support of COOL/CORAL limited to bugfixes
  - clumsy data management
- Main requirements
  - file based
  - filesystem-like hierarchy
  - simple management of contributions

Among alternative backends, Git looked promising

Git

Git is a *Distributed Version Control System* with interesting features:

- filesystem structure with versions
- each clone contains all versions
- tags and branches
- built-in incremental synchronization
- data compression and deduplication

but

- no support for the 3^{rd} dimension (IOVs)

The First 2 Dimensions

Git main goal is to track changes to a filesystem hierarchy.

The Detector Description partition of LHCb CondDB is just XML files, with multiple revisions.

Porting the Detector Description data to a Git database is natural.

Adding the 3^{rd} Dimension

Alignments and Calibrations are recorded in LHCb CondDB as XML files with an attached IOV.

Adding the 3rd Dimension

Alignments and Calibrations are recorded in LHCb CondDB as XML files with an attached IOV.

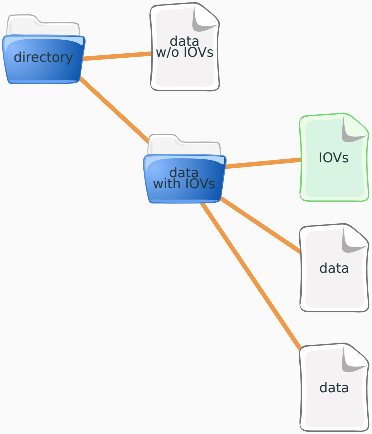
Conditions time evolution must be mapped to a simple filesystem layout.

Adding the 3^{rd} Dimension

Alignments and Calibrations are recorded in LHCb CondDB as XML files with an attached IOV.

Conditions time evolution must be mapped to a simple filesystem layout.

One possibility:
  use directories and metadata files



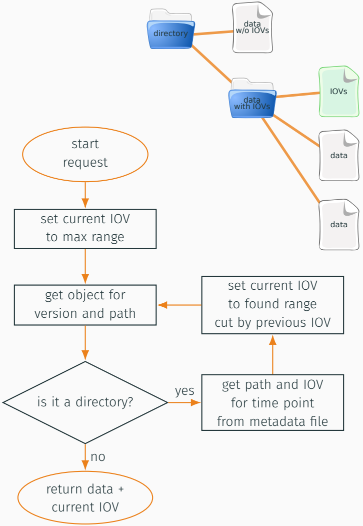
Accessing the Data

A condition payload is identified by:

- version (tag)
- string id (path)
- event time

Git API allow retrieval of payloads by version and path.

Metadata files are simple mappings from IOV to payload.



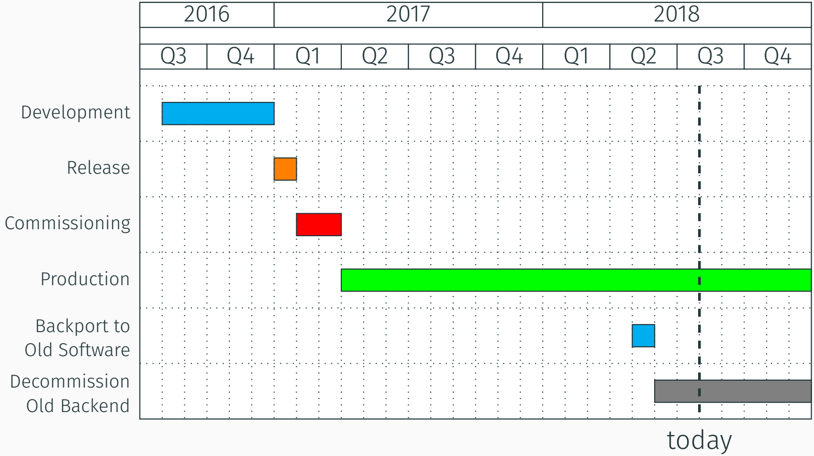
LHCb CondDB Ecosystem Changes

Not just the CondDB backend

- data management
  - custom tools for CondDB I/O → text editors
- contributions management
  - JIRA + custom tools → Gitlab + merge requests
- database files deployment
  - custom web based sync tools → Git native sync protcol

Git CondDB in Production

Commissioning



From conception to production in less than one year!

Performance: Time

Time to load all objects for a given version and event time:

Backend	Total Time
COOL/SQLite	8.1 s
Git	3.5 s

- Host
  - Intel(R) Core(TM) i7-7560U CPU @ 2.40GHz
  - Data from CVMFS with SSD local disk
- $\sim$ 13000 queries to CondDB
- XML parsing + objects creation $\rightarrow \sim$ 20 s

Performance: Size

Space on Disk (as of May 2017) in MB

	SQLite	Git bare
Det. Desc.	34	3
Align. + Calib.	730	285
Environment	2300	357
Simulation	25	9

average $\sim 5\times$ reduction

(Git CondDB includes LHCb Upgrade Detector Description)

Summary

Summary

- Other technologies are available
  - see HSF CWP on Conditions Data
- Git fits our main requirements and more
  - faster, smaller, large community support, tools, ...
- Current implementation very simple
  - not much time spent in optimization (e.g. IOV lookup)
- Git CondDB has been used in production since 2017 data taking
  - although intended as R&D for Run 3
- Changes are planned for Run 3
  - investigating DD4hep Detector Description framework (poster #111)
  - new data format for Detector Description and Conditions
  - but we are planning to keep Git as backend

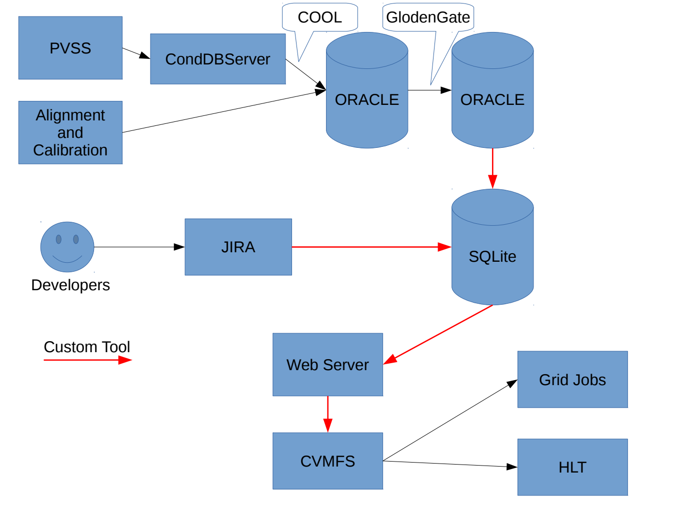
Back Up Slides

Why a Recursive Algorithm?

Why use recursion instead of simple branching?

1. call to object retrieval Git API in only one place
2. allow partitioning of metadata files for faster lookup
   $\mathcal{O}(\log n)$ instead of $\mathcal{O}(n)$

Infrastructure: COOL



Infrastructure: Git