LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Belle II Conditions Database Overview

Martin Ritter[1], Lynn Wood[2],
Server: Todd Elsethagen[2], Kevin Fox[2], Jeter Hall[2], Bibi Raju[2], Malachi Schram[2], Eric Stephan[2]
Client: Thomas Kuhr[1], Christian Pulvermacher[3]
[1]Ludwig-Maximilians-University (LMU), Munich
[2]Pacific Northwest National Laboratory (PNNL), Richland
[3]High Energy Accelerator Research Organization (KEK), Japan

18th International Workshop on Advanced Computing and Analysis Techniques in Physics Research
August 22, 2017

TUM LMU Excellence Cluster Universe

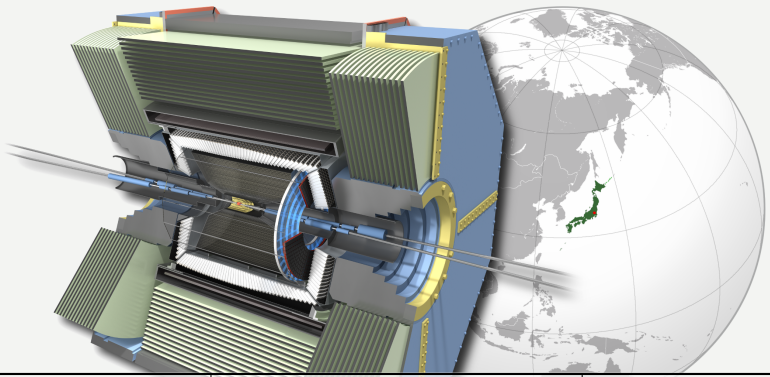
Bundesministerium für Bildung und Forschung

Pacific Northwest NATIONAL LABORATORY

Belle/Belle II Experiment

Asymmetric e^+e^- experiment mainly at the $\Upsilon(4S)$ resonance (10.58 GeV)

Focus on B, charm and τ physics



	KEKB/Belle	SuperKEKB/Belle II
operation	1999–2010	start 2018
peak luminosity	$2.11 \times 10^{34}\,\mathrm{cm^{-2}s^{-1}}$	$8 \times 10^{35}\,\mathrm{cm^{-2}s^{-1}}$
integrated luminosity	$1023\,\mathrm{fb^{-1}}$ (772 million $B\overline{B}$ pairs)	$50\,\mathrm{ab^{-1}}$

Conditions Data

Conditions Data: changes over time, not part of event

- luminosity
- detector status
- calibrations
- reconstruction settings

Challenges:

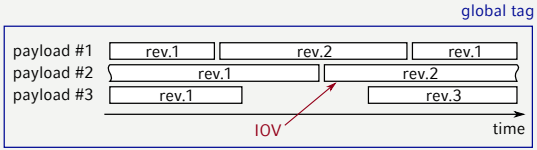
- required for data taking
- required worldwide for grid/cloud processing
- vastly different lifetimes/sizes
- different requirements by different sub-detectors

Terms

payload is one atom of conditions data (e.g. alignment constants)

IOV is short for "interval of validity", the time interval in which a payload is valid

global tag is an immutable set of payloads and their IOVs.

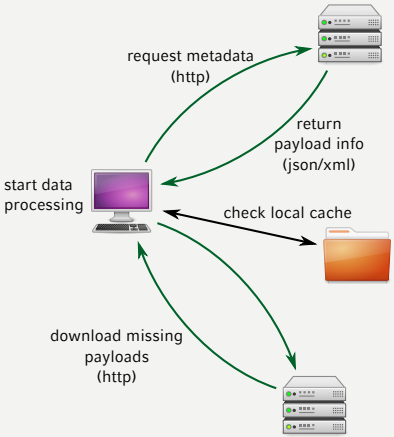


Design Overview

- use industry standard tools where possible
- decouple metadata from content
- use REST service for metadata
- use files for payloads
- smallest granularity: 1 run
  (uninterrupted period of data taking,
  up to a few hours)

➡ REST interface greatly decouples server/client development

➡ Very low requirements, only connect to http(s)



LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Pacific Northwest NATIONAL LABORATORY

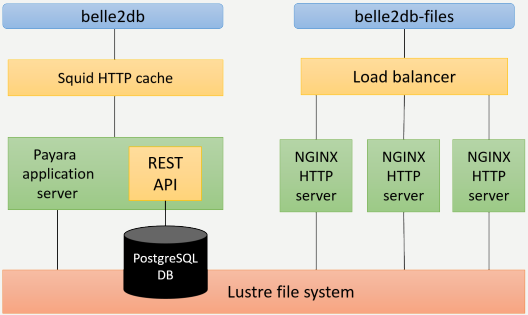
Server Side

Lynn Wood, Todd Elsethagen, Kevin Fox, Jeter Hall,
Bibi Raju, Malachi Schram, Eric Stephan

Current Back-End Configuration

two separate services:

- left: DB access/file upload
- right: payload file download



Database Server

- Squid cache in front of REST application server to reduce load
- Payara Micro Java server
- Hazelcast in-memory data grid platform for caching and stability

Payload server

- Load-balanced NGINX high performance HTTP servers

➡ Each component is implemented as a Docker container managed using Kubernetes – provides modularity and auto-restart

Performance of Current System

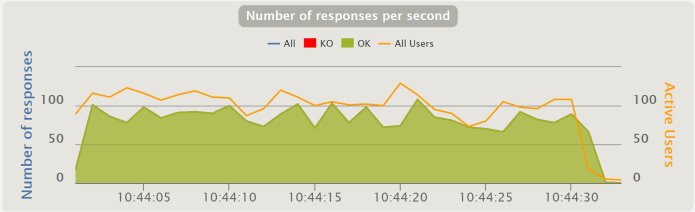
Directed testing with Gatling, a HTTP load stress tool

- Scala scripting for custom test design

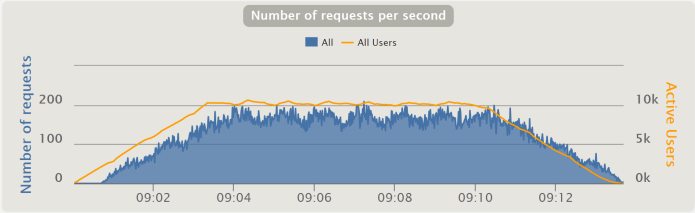
Database server performance dependent on Squid, Java config

Payload file server (load-balanced x3) very stable

Current performance about half of needed levels at full expected Belle II processing.



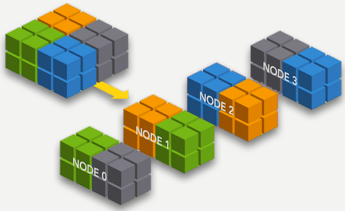
Stress test of the database server, showing a sustained rate of ~ 80 requests/second



Stress test of the payload file server, showing a sustained rate of ~ 180 requests/second and support of 10,000 simultaneous connections

Multi-Site Caching via Hazelcast

- Currently all Conditions DB services are installed on a **single** PNNL-managed host
- Hazelcast provides standalone caching store to improve service performance
  - Data is evenly distributed among the nodes
    ⇨ **horizontal scaling** of processing and storage
  - Backups also distributed among nodes
    ⇨ protect against **single-node failure**
  - Each site also supports a dedicated "local cache" for commonly requested items



- Evaluating multi-site system at PNNL now
  - Hazelcast would auto-cluster sites as they come online
  - Cache distributed/partitioned across the memory resources (Java heap) contributed by each site
  - Access to partitioned cache would be **transparent** to the application: Hazelcast manages routing, no code changes for distributed access
  - Frequently accessed remote cache entries will be stored in a "Near Cache"

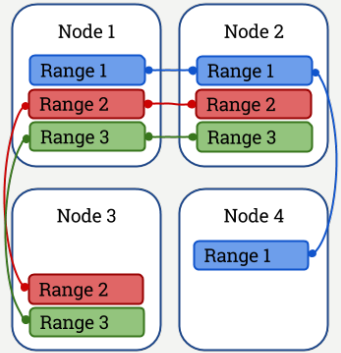
Database Replication and Authentication

The PostgreSQL database also not currently scalable

- Investigating distributed database options such as CockroachDB
  - Symmetric node architecture, horizontal scalability per site
  - Distributed transactions, majority consensus for consistent replication
  - Automated repair after failure

Authentication currently not implemented

- Expectation is to leave read operations open, but require authentication for write operations
- Considering leveraging the X.509 authentication already present in the Belle II Grid computing interface
- Create new roles for database



Range Replication in CockroachDB

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Client Side

Martin Ritter, Thomas Kuhr, Christian Pulvermacher

Client Side

Belle II Software Framework

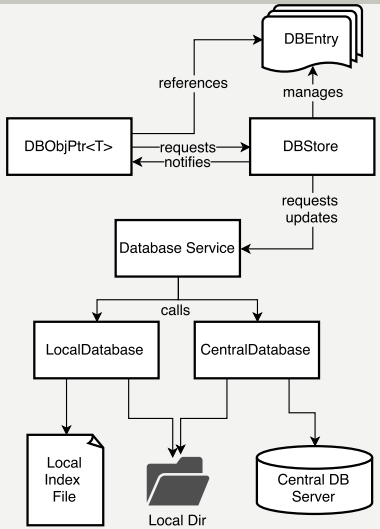
- C++14, ROOT 6
- Python 3 configuration/scripting
- Multi processing capabilities

➡ use ROOT files for conditions payloads

- users obtain reference to payload by name
- framework will obtain payload information
- handle updates transparently
- users can check/be notified on changes

➡ allow operation without connectivity

- read payload information from file.
- allow downloading of (partial) database



Payload Information and Distribution

Allow cascade of payload information providers

- testing of new payloads locally
- additional analysis-specific payloads (e.g. training-data)

Payloads as files allows for flexible payload delivery

- trivial caching in file system
- various distribution possibilities could be investigated (cvmfs, key-value stores, pack-files, …)
- hybrid solutions possible (e.g. only some payloads on cvmfs)
- http as reliable fallback

Intra Run Dependency

Some conditions data might change more frequently than per run

- payload will contain multiple objects
- handled transparently on client side

Command Line Interface

REST interface makes implementation of clients very easy

- libcurl for C/C++ client in the software framework
- requests library for Python
- large amount of standard tools

User friendly command line interface

- pure python
- inspect/modify database contents
- e.g. compare global tags
  `b2conditionsdb diff tag1 tag2`

Python Requests

```
#!/usr/bin/env python3.6
import requests
BASE_URL = "http://..."
globalTag = "development"
r = requests.get(f"{BASE_URL}/globalTag/"
                 "{globalTag}/payloads")
r.raise_for_status()
for payload in r.json():
    print(payload["checksum"])
```

Conclusions

Belle II Conditions Database

- leverage existing tools where possible
- REST: easy, well defined interface between client and server

Server

- payload content agnostic web service
- implemented using industry tools
- single server setup at half the expected performance

Client

- use ROOT files as payloads
- automatic updates, “offline” mode
- independent command line client

Additional Details

Two Posters today

- Implementing the Belle II Conditions Database using Industry-Standard Tools (L. Wood et al.)
- Belle II Conditions Database Interface (M. Ritter et al.)

Thank you
for your attention