Conditions Database Code Review

- Expected attendees
- Documents
- Q&A

Expected attendees

- PNNL: Lynn Wood, Malachi Schram, Todd Elsethagen
- BNL: Carlos Fernando Gamboa, Dmitry Arhipkin
- LMU-Munich: Thomas Kuhr, Martin Ritter
- Irfu, CEA Saclay: Andrea Formica
- CERN: Misha Borodin
- JSI-Ljubljana: Marko Bracko

Documents

- **Conditions DB Status Update** (Oct 2017 Belle II General Meeting)
- b2s architecture overview (including general architecture and component diagrams)
- PayloadResource.java (code for the primary /iovPayloads service)
- PayloadResource_annotated.java (version with additional comments)
- PayloadiovRpt.java (entity class for the payload_iov_rpt table inpostgresql and JPQL query for the /iovPayloads service)
- Source code (older, latest will be posted soon)
- REST API (via swagger)
- Schema (corrected)

Q&A

- *Please show access patterns from production vs. “user” accesses and how they were behaving differently (even up to SQL-like descriptions) – to better understand why the cache was not efficient in some cases*

  The majority of READ service requests are for the .iovPayloads service. This service returns all IOV payload information for a global tag and experiment/run. Response size for production requests are in the 2-3 MB range. So far, user request responses are in the K range.

  The primary production access pattern is to call the /iovPayloads service for an experiment/run. Experiment/run combinations being called are few, meaning the web-proxy cache is very efficient with a high hit/miss ratio.

  A secondary production and user access pattern exists, where for a global tag and experiment a sequence of n run numbers will be cycled through resulting in n calls to the /iovPayloads service. Depending on the size of n this can result in a sustained heavy load bypassing the web-proxy cache.

- *Have you already tested after correction of the hibernate eager / lazy fetching? This on is a killer for query performance...*

  Yes, this has been tested and resolved. It is a killer...

- *Questions about the internal cache (hibernate + Hazelcast):*
  - *Why do you use this kind of cache at the server level? If the objects I want to access should be cached by hibernate/hazelcast, then why shouldn’t they be cached by the web proxy as well?*

    Yes, responses are cached at the web-proxy level and are retained for 10 minutes (this is configurable). There are two primary reasons for server level or application caching in addition to web-proxy caching:
    1. Improve performance for web-proxy low hit/miss ratio use cases and in turn minimize database access.
    2. To allow clustering/synchronization of service installations. Application cache is local to an installation and is pre-fetched from database at startup. Service WRITEs (i.e. PUT/POST/DELETE) are communicated as cache invalidation messages to the other cluster members. (see https://github.com/hazelcast/hazelcast-hibernate#hazelcastlocalcacheregionfactory)

    *did you see big differences when activating these internal caches with respect to no-cache at all at server level?*

    Yes, local application caching does improve performance, especially under web-proxy low hit/miss ratio use cases. We have also introduced compression, which has helped to improve performance as well.