The HSF Conditions Database Reference Implementation

Andrea Formica\textsuperscript{1}, Lino Gerlach\textsuperscript{2}, Giacomo Govi\textsuperscript{3}, Paul Laycock\textsuperscript{2}, Ruslan Mashinistov\textsuperscript{2}, Chris Pinkenburg\textsuperscript{2}

\textsuperscript{1}Université Paris-Saclay (IRFU/CEA, FR)  
\textsuperscript{2}Brookhaven National Lab (US)  
\textsuperscript{3}Fermilab (US)

9 May 2023
## Conditions Data – Introduction

“Conditions data is any additional data needed to process event data”

<table>
<thead>
<tr>
<th>Changes over time</th>
<th>High access rates</th>
<th>Heterogenous data</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Repeat detector calibration with larger cosmic dataset</td>
<td>• Distributed computing jobs access same conditions</td>
<td>• Granularity varies (time indexed, run-indexed,</td>
</tr>
<tr>
<td>• Improve calibration algorithms</td>
<td>data simultaneously</td>
<td>constant)</td>
</tr>
<tr>
<td></td>
<td>• Access rates up to ~kHz</td>
<td>• Structure of payload varies (3D map, time-indexed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>values, single number, …)</td>
</tr>
<tr>
<td>Versioning &amp; configuration</td>
<td>Fast DB queries &amp; effective caching</td>
<td>Payload agnostic by design</td>
</tr>
</tbody>
</table>

Similar challenges for various HEP experiments
Conditions Data – Recommendations

- HSF Conditions Databases activity: [https://hepsoftwarefoundation.org/activities/conditionsdb.html](https://hepsoftwarefoundation.org/activities/conditionsdb.html)
  - Discussions across various experiments
- Key recommendations for conditions data handling
  - Separation of payload queries from metadata queries
  - Schema below to organise payloads

‘Interval of Validity’: generalized concept of time (begin can be time stamp, run number, lumi block, …)

actual data (e.g. in a file)

configurations for each type of conditions data

top-level configuration of all conditions data

Kalpesh K. Desai — HEP Software Foundation
Community White Paper Working Group – Conditions Data
Conditions Data – Use Cases

• HSF Conditions Database meeting: use cases https://indico.cern.ch/event/1280790/
• Most can be realised w/ HSF Recomm.
• Exception: **Fast-Processing**. Goal:
  • Publish data for analysis fast
  • Maximize physics performance

<table>
<thead>
<tr>
<th>Use case</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online</td>
<td>• High Level Trigger</td>
</tr>
<tr>
<td>Reprocessing</td>
<td>• Run reco w/ improved calib.</td>
</tr>
<tr>
<td>Analysis</td>
<td>• High level physics analysis</td>
</tr>
<tr>
<td>Development</td>
<td>• Test new calib. within existing GT</td>
</tr>
<tr>
<td>Fast-processing</td>
<td>• Process data w/ just-in-time calib.</td>
</tr>
</tbody>
</table>

Need additional info: end of IOV
Implementation – Overview

nopayloadclient:
- Client-side stand-alone C++ tool
- Communicates with nopayloaddb (server)
- Local caching
- Handling of payloads

Experiment-agnostic lib
nopayloadclient

sPHENIX-specific lib
sphenixnpc

DUNE-specific lib
dunenpc

remote payload store

nopayloadclient

payload

client side

server side

REST*

nopayloaddb

*Example query (simplified)
curl http://<host>/api/payloadiovs/?gtName=test_gt&iovNum=42 -> {type_1: url_1, type_2: url_2, ...}
Implementation – Database Schema

<table>
<thead>
<tr>
<th>GlobalTag</th>
<th>PayloadList</th>
<th>GlobalTagStatus</th>
<th>PayloadType</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>id</td>
<td>id</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>author</td>
<td>description</td>
<td>description</td>
</tr>
<tr>
<td></td>
<td>description</td>
<td>created</td>
<td>created</td>
</tr>
<tr>
<td></td>
<td>status_id</td>
<td>updated</td>
<td>updated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Payloads are not stored in schema

Payloads are also not stored in schema

Major- & minor IOV for more flexibility

IOVs also have an end

Combination of major and minor IOV into single column for performance optimisation
Deployment on OKD

- Automated deployment on OKD (Helm chart)
- Horizontally scalable
- Open Source only

Easily adoptable for various HEP experiments

**Diagram:**

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nginx</td>
<td>WSGI</td>
<td>pgBouncer</td>
<td>Postgres</td>
</tr>
<tr>
<td>/mnt</td>
<td>/mnt</td>
<td>/mnt</td>
<td>/var/lib/pgsql/data</td>
</tr>
<tr>
<td>logs</td>
<td>logs</td>
<td>log</td>
<td></td>
</tr>
<tr>
<td>/current_logs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cron</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pg_dump</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>db_dump_26.20.2022.tgz</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>/data</td>
</tr>
</tbody>
</table>

```

Powered by nfS (persistent storage)
Performance Testing – Strategy

- Simulate expected DB occupancy
- Simulate access patterns
  - Query read API for payload URL
  - Parallel requests via HTC or MT

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Payload Types</th>
<th>Payload IOVs (per type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tiny</td>
<td>10</td>
<td>100 (10)</td>
</tr>
<tr>
<td>tiny-moderate</td>
<td>10</td>
<td>2000 (200)</td>
</tr>
<tr>
<td>moderate</td>
<td>100</td>
<td>20000 (200)</td>
</tr>
<tr>
<td>heavy-usage</td>
<td>100</td>
<td>500000 (5000)</td>
</tr>
<tr>
<td>worst-case</td>
<td>200</td>
<td>5200000 (26000)</td>
</tr>
</tbody>
</table>

All following tests:
- Random major- and minor IOV, no caching
- Query metadata only, no payloads

![Graphs showing mean response time and frequency with annotations](image)
Performance Testing – ORM vs Raw SQL

- High frequency read API workflow:
  - Filter on global tag, major- and minor IOV *
  - Find ‘latest’ IOV for each payload type **
  - Return payload type, file URL, IOV
- Django’s ORM writes query for user
- Optimized raw SQL query
  - Covering index (index-only scan)
  - Combined IOV column <major.minor>
  - Lateral join operation

 Resp. freq. vs size of queried GT

*: my_major<major_iov OR (my_major=major_iov AND my_minor<=minor_iov) **: for max major_iov, find max minor_iov
Performance Testing – Scaling

- Investigate scaling w/ size of queried GT
  - Content of DB remains constant
- Measure mean response frequencies
  - Scales with number of payload types
    - More data to sort and return
  - Almost flat vs number of IOVs
    - Index scan (covering index)
- Also tested scaling w.r.t. size of DB
  - No dependence, plot in backup

Resp. freq. vs size of queried GT

![Graph showing response frequency vs size of queried GT]

1M IOVs
Conclusion

• HSF recommends how to handle conditions data

• Presented experiment-agnostic reference implementation
  • Django REST API: nopayloaddb
    • Automized deployment on OKD
  • C++ client-side client: nopayloadclient

• Performance tests show solid results
  • After raw SQL query optimisation

• Already in use at sPHENIX
• ProtoDUNE planning to deploy as well
Thank you for your Attention!
Backup
Performance Testing – Scaling

- Scales with number of payload types
- Almost flat w.r.t. number of IOVs

- Performance depends on size of queried GT
- Additional ‘stuff’ in DB has no significant impact
Performance Testing – High Frequency

- Simulate offline reco use case
  - Many jobs launched at the same time
- Cooperative multithreading (asynchio)
  - Send requests firsts
  - Process responses later
- Allows very high peak request frequency
- Server-side queuing of requests works

10k requests sent within ~1.2 secs
received all responses within ~55 sec
PostgreSQL High-Availability Cluster

- Consider DB cluster for high-availability and higher performance

- **CloudNativePG:**
  - Open source operator (Kubernetes) for PostgreSQL
  - Primary / Standby architecture
  - Native support for pgBouncer connection pooling
PayloadIOV Read API – Raw SQL Query

```
SELECT pi.payload_url, pi.major_iov, pi.minor_iov, pt.name, ...
FROM "PayloadList" pl
JOIN "GlobalTag" gt ON pl.global_tag_id = gt.id AND
gt.name = %(my_gt)s
JOIN LATERAL
    (SELECT payload_url, major_iov, minor_iov, ...
     FROM "PayloadIOV" pi
     WHERE pi.payload_list_id = pl.id
     AND pi.comb_iov <= CAST(%(my_major_iov)s +
     CAST(%(my_minor_iov)s AS DECIMAL(19,0)) / 10E18 AS
     DECIMAL(38,19))
     ORDER BY pi.comb_iov DESC
     LIMIT 1
    ) pi ON true
JOIN "PayloadType" pt ON pl.payload_type_id = pt.id;
```

- LATERAL joining. Without LATERAL, each sub-SELECT is evaluated independently and so cannot cross-reference any other FROM item
- Covering index on Payload table including combined IOV and reference to the PayloadList

For each PayloadList (Type)

Get Payloads descending ordered by combined IOV

Limit return to 1 line - latest Payload for a given IOVs

And then append the results of each subquery to create the final output
Hash Join (cost=7.23..410.15 rows=86 width=70) (actual time=6.111..365.158 rows=200 loops=1)
Hash Cond: (pl.payload_type_id = pt.id)
  -> Nested Loop (cost=0.71..403.40 rows=86 width=69) (actual time=6.017..364.977 rows=200 loops=1)
    -> Nested Loop (cost=0.15..11.70 rows=86 width=16) (actual time=0.048..0.133 rows=201 loops=1)
      -> Seq Scan on "GlobalTag" gt (cost=0.00..1.09 rows=1 width=8) (actual time=0.023..0.025 rows=1 loops=1)
        Filter: (name)::text = 'worst-case':text
        Rows Removed by Filter: 6
      -> Index Scan using "PayloadList_global_tag_id_2b35c85f" on "PayloadList" pl (cost=0.15..9.75 rows=86 width=24) (actual time=0.022..0.083 rows=201 loops=1)
        Index Cond: (global_tag_id = gt.id)
      -> Limit (cost=0.56..4.53 rows=1 width=61) (actual time=1.815..1.815 rows=1 loops=1)
        -> Index Only Scan using "PayloadIOV" pi (cost=0.56..3484.55 rows=876 width=61) (actual time=1.815..1.815 rows=1 loops=201)
          Index Cond: (payload_list_id = pl.id)
          Filter: (major_iov < 100000000 OR (major_iov = 100000000 AND minor_iov <= 100000000))
          Rows Removed by Filter: 24669
          Heap Fetches: 0
    -> Hash (cost=4.01..4.01 rows=201 width=17) (actual time=0.078..0.078 rows=201 loops=1)
      Buckets: 1024 Batches: 1 Memory Usage: 19kB
      -> Seq Scan on "PayloadType" pt (cost=0.00..4.01 rows=201 width=17) (actual time=0.018..0.043 rows=201 loops=1)
Planning Time: 0.996 ms
Execution Time: 365.221 ms

Hash Join (cost=7.23..90.89 rows=86 width=70) (actual time=6.111..365.158 rows=200 loops=1)
Hash Cond: (pl.payload_type_id = pt.id)
  -> Nested Loop (cost=0.71..403.40 rows=86 width=69) (actual time=6.017..364.977 rows=200 loops=1)
    -> Nested Loop (cost=0.15..11.70 rows=86 width=16) (actual time=0.048..0.133 rows=201 loops=1)
      -> Seq Scan on "GlobalTag" gt (cost=0.00..1.09 rows=1 width=8) (actual time=0.023..0.025 rows=1 loops=1)
        Filter: (name)::text = 'worst-case':text
        Rows Removed by Filter: 6
      -> Index Scan using "PayloadList_global_tag_id_2b35c85f" on "PayloadList" pl (cost=0.15..9.75 rows=86 width=24) (actual time=0.022..0.083 rows=201 loops=1)
        Index Cond: (global_tag_id = gt.id)
      -> Limit (cost=0.56..4.53 rows=1 width=61) (actual time=1.815..1.815 rows=1 loops=1)
        -> Index Only Scan using "PayloadIOV" pi (cost=0.56..3484.55 rows=876 width=61) (actual time=1.815..1.815 rows=1 loops=201)
          Index Cond: (payload_list_id = pl.id)
          Filter: (major_iov < 100000000 OR (major_iov = 100000000 AND minor_iov <= 100000000))
          Rows Removed by Filter: 24669
          Heap Fetches: 0
    -> Hash (cost=4.01..4.01 rows=201 width=17) (actual time=0.078..0.078 rows=201 loops=1)
      Buckets: 1024 Batches: 1 Memory Usage: 19kB
      -> Seq Scan on "PayloadType" pt (cost=0.00..4.01 rows=201 width=17) (actual time=0.018..0.043 rows=201 loops=1)
Planning Time: 0.996 ms
Execution Time: 365.221 ms
Investigating Query Plans - II

```
-> Limit (cost=0.56..4.53 rows=1 width=61) (actual time=1.815..1.815 rows=1 loops=201)
  -> Index Only Scan using combo_covering_idx on "PayloadIOV" pi
     (cost=0.56..3484.55 rows=876 width=61) (actual time=1.815..1.815 rows=1 loops=201)
       Index Cond: (payload_list_id = pl.id)
       Filter: ((major_iov < 100000000) OR ((major_iov = 100000000) AND (minor_iov <= 100000000)))
       Rows Removed by Filter: 24669
       Heap Fetches: 0
```

Index Condition & Filter

```
-> Limit (cost=0.56..0.82 rows=1 width=61) (actual time=0.014..0.014 rows=1 loops=201)
  -> Index Only Scan using combo_covering_idx on "PayloadIOV" pi
     (cost=0.56..232.55 rows=876 width=61) (actual time=0.014..0.014 rows=1 loops=201)
       Index Cond: (payload_list_id = pl.id)
       (major_iov < 100000000))
       Heap Fetches: 0
```

Index Condition Only
Raw SQL - Combined IOV Column

- Preselection on major- & minor IOV (AND / OR)
  - Scales with entries to consider
  - Query uses ‘Filter’
- Preselection on single column (<=)
  - Constant time
  - Query uses ‘Index Condition’

- Combine major- and minor IOV into single column:

<table>
<thead>
<tr>
<th>major_iov</th>
<th>minor_iov</th>
<th>comb_iov</th>
</tr>
</thead>
<tbody>
<tr>
<td>477658914</td>
<td>1001747433</td>
<td>477658914.00000001001747433</td>
</tr>
<tr>
<td>23283443</td>
<td>1525747152</td>
<td>23283443.00000001525747152</td>
</tr>
<tr>
<td>1834979804</td>
<td>648013294</td>
<td>1834979804.0000000648013294</td>
</tr>
</tbody>
</table>

- Fast across all values while selecting on both