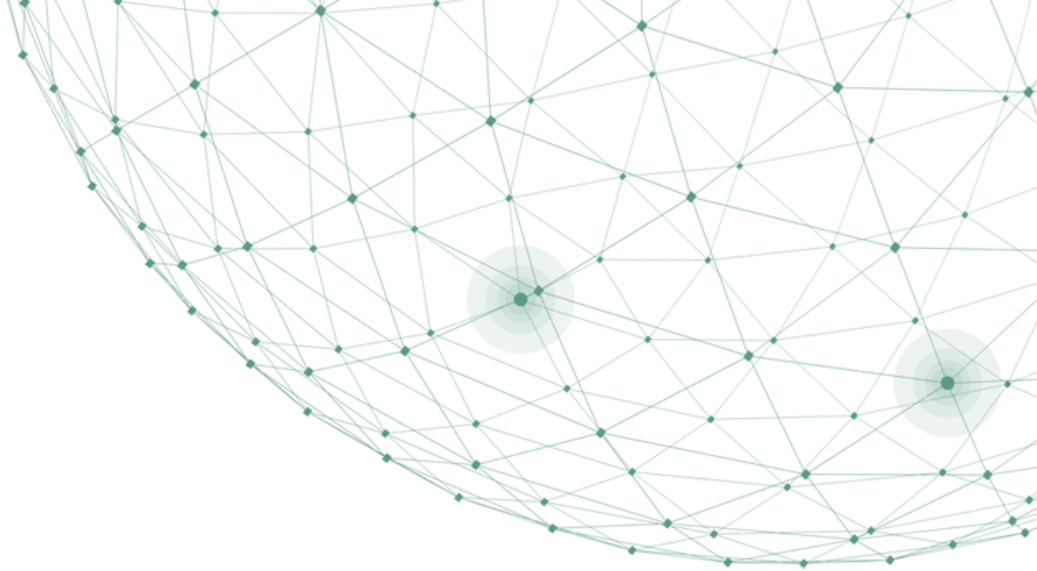
Memory Connector Improvements openLooKeng Metastore

Han Weng

openLooKeng

https://openlookeng.io

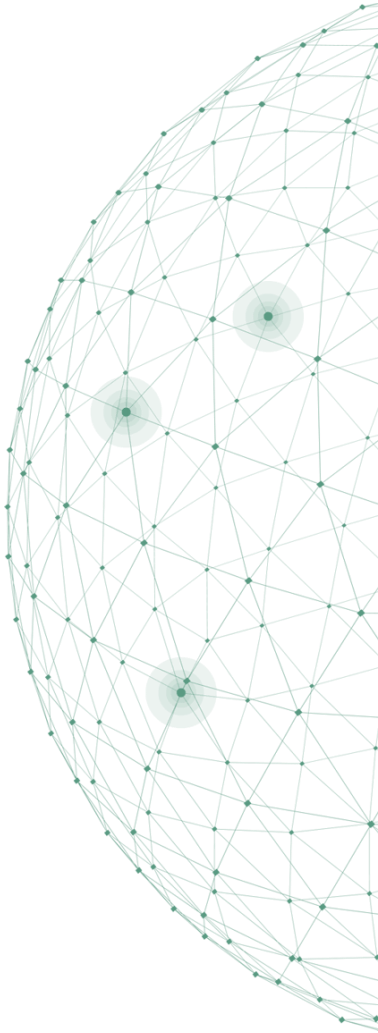


Part I.
Memory Connector Improvements

openLooKeng

https://openlookeng.io

Content

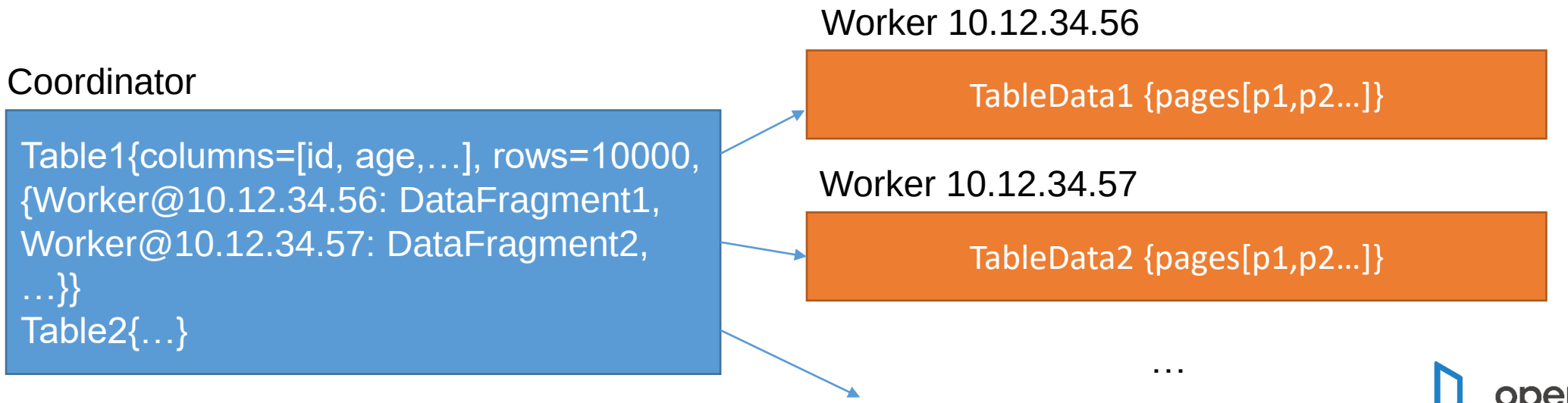


openLooKeng

https://openlookeng.io

1. Old memory connector (from Presto Community)

- Metadata stored in Java collections locally on (single) coordinator
  - Table list
  - Track data fragments on each worker
- Table data stored in simple Java objects on workers:



openLooKeng

https://openlookeng.io

1. Old memory connector (from Presto Community)

- Metadata stored in Java collections locally on (single) coordinator
  - Table list
  - Track data fragments on each worker

```
Map<HostAddress, MemoryDataFragment> dataFragments
```

- Table data stored in simple Java objects on workers:

```
Map<Long, TableData> tables
```

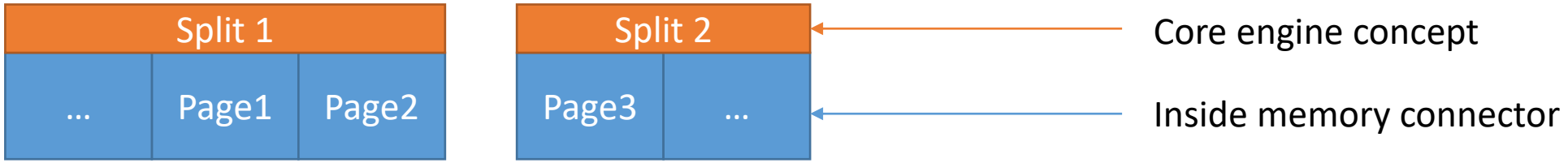
```
private static final class TableData
{
    private final List<Page> pages = new ArrayList<>();
    private long rows;

    public void add(Page page)
    {
        pages.add(page);
        rows += page.getPositionCount();
    }

    private List<Page> getPages() { return pages; }

    private long getRows() { return rows; }
}
```

openLooKeng

https://openlookeng.io

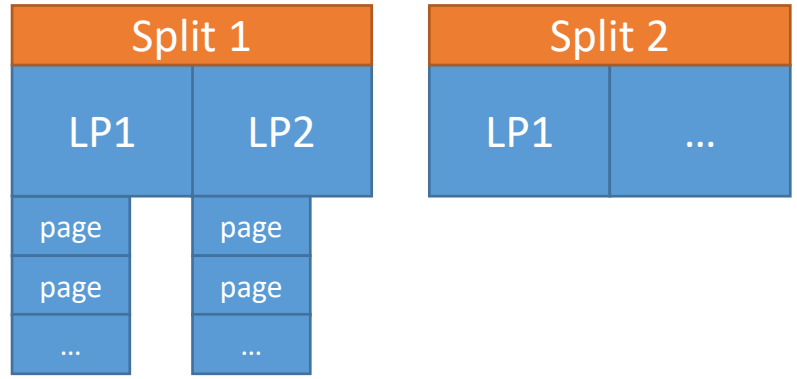
Issues with old memory connector

- Information stored in local objects -> data loss after server restart
- Data split scheduled at openLooKeng level -> no customized parallelization
- No index supported -> full scan of pages list always required
- No predicate pushdown -> always return all pages to Filter Operator
- Bad query performance



2. New design overview

- Use HetuMetastore to persist table metadata
  - Metadata won’t be lost after server restart
  - Metadata can be shared across cluster (multiple coordinator, workers)
- Introduce **LogicalParts** (LP) under **splits**
  - Splits are further organized into LogicalParts
  - LogicalParts contain **index** and data
- Table data spilled to disk after creation/insertion
  - Data won’t be lost after restarting server
  - Data can be offloaded when node runs out of memory (LRU offload)
  - Separate index and actual data: pre-filter, no need to load all data
- LogicalPart processing and data spilling are async.

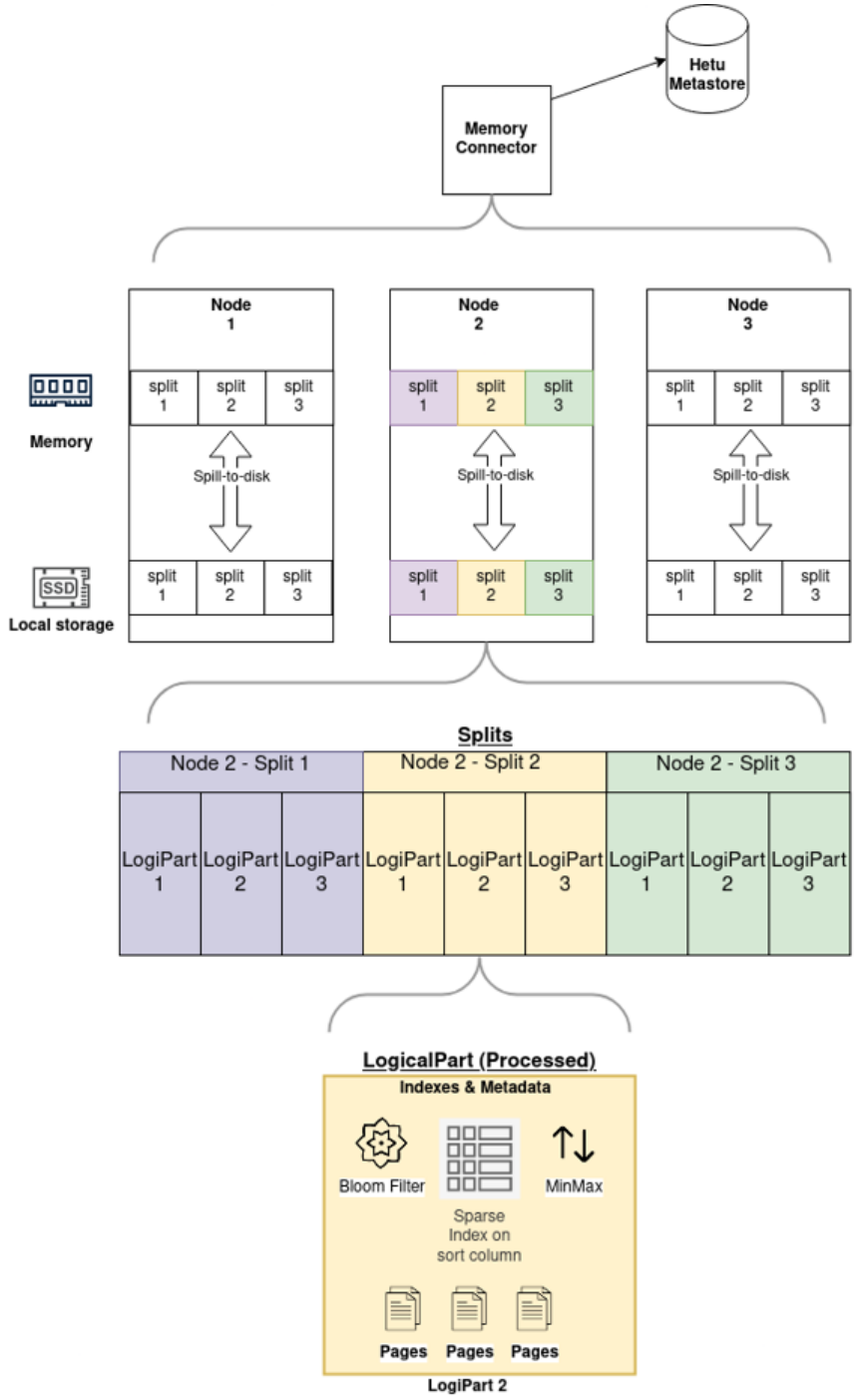


openLooKeng

https://openlookeng.io

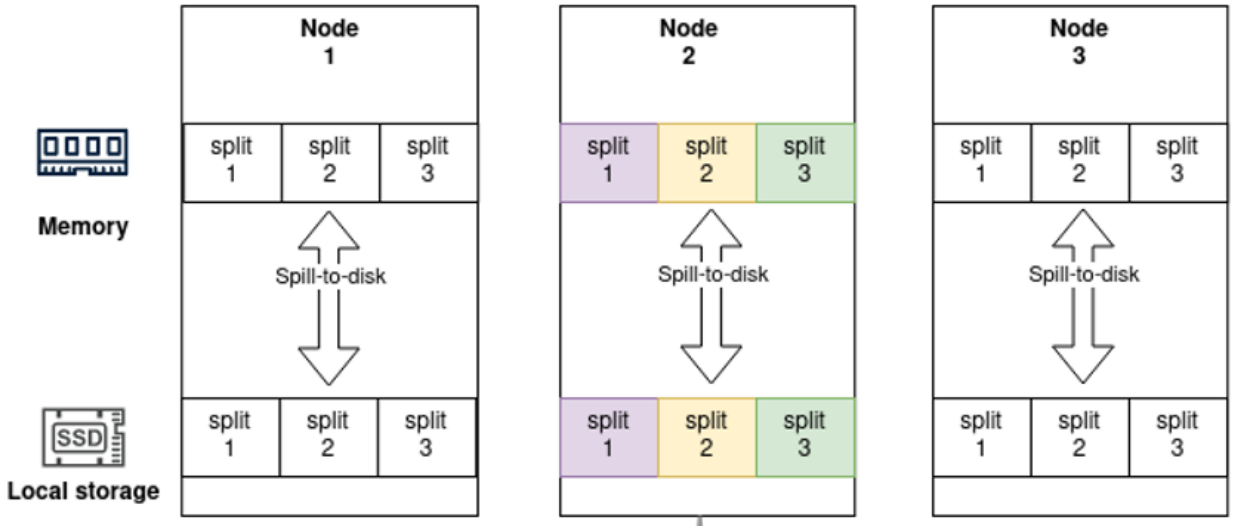
Architecture and usage

```
CREATE TABLE memory.test.table
WITH (sorted_by=array['phone1']),
      index_columns=array['phone2'],
      spill_compression=true)
AS SELECT * FROM hive.schema.table;
```



Splits

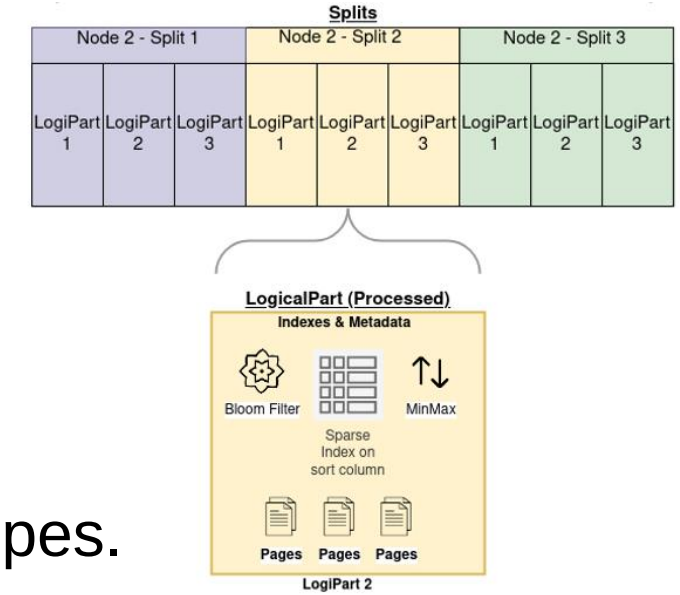
- **Goal: maximize parallelism**
- During table creation, pages are distributed to each of the workers
- Each of the workers will have n splits. n will be determined by number of logical CPU cores on the machine.
- When TableScan is scheduled, n splits will be scheduled to maximize parallelism.



openLooKeng

https://openlookeng.io

LogicalPart (LP)



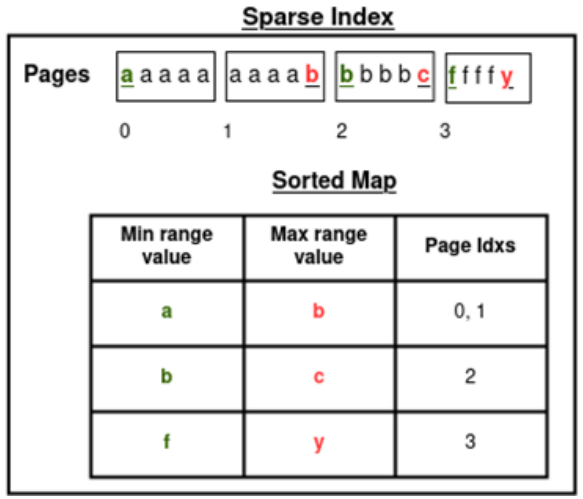
- **Goal: Reduce data that is read.**
- Each split contains multiple LogicalParts.
- LP size configurable (default 256MB). Similar to ORC stripes.
- LPs are immutable. New LP created when previous one is filled.
- Each LP is FSM. (Accepting pages, processing/ed, spilled, etc)
- As part of background processing after table creation, indexes are created:
  WITH(sorted_by=[…]): bloom, sparse, minmax
  WITH(index_columns=[…]): bloom, minmax
- Based on pushed down predicate, entire LogiParts can be filtered out using Bloom Filter and MinMax index
- Further filtering is done using Sparse index

openLooKeng

https://openlookeng.io

Sparse Index

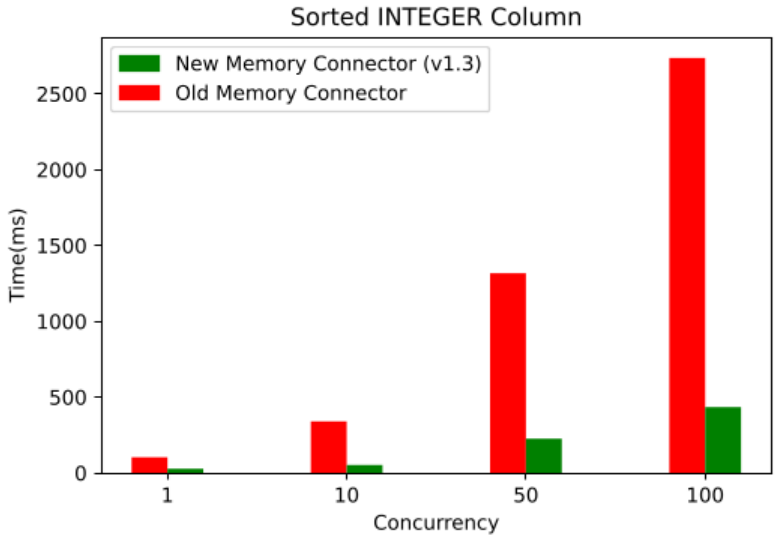
- **Goal: Reduce input Pages**
- BTree index internally
- Pages are first sorted, optimized and Sparse Index is created
- Allows for smaller index size since not all unique values need to be stored
- Sparse index helps reduce input rows but does not perform perfect filtering; further filtering is done by openLooKeng's Filter Operator
- Example:
  - column=a -> 0,1
  - column=b -> 1,2
  - column=c -> 2
  - column=d -> No pages returned

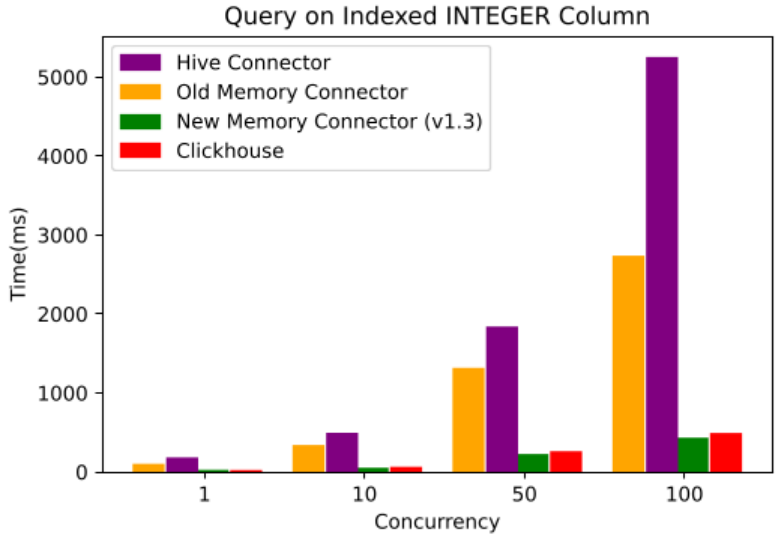


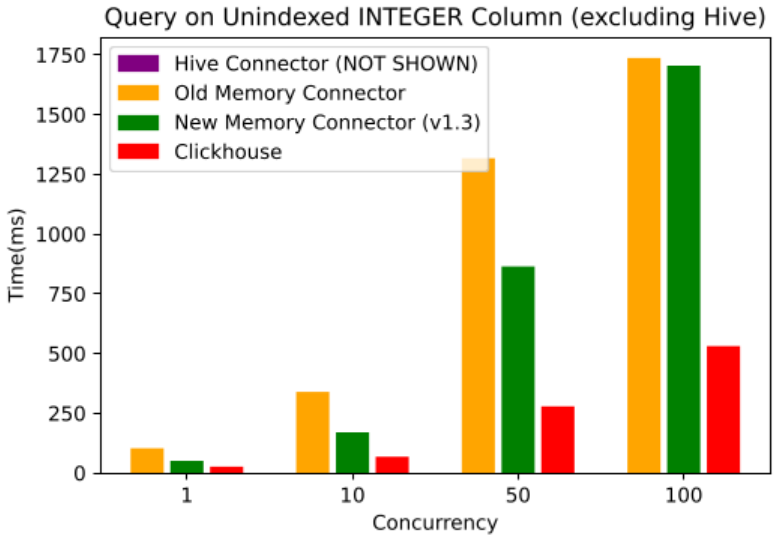
openLooKeng

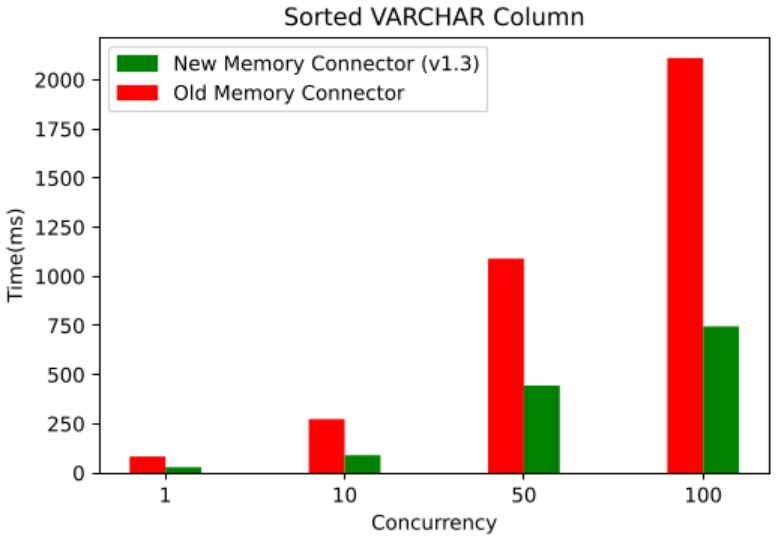
https://openlookeng.io

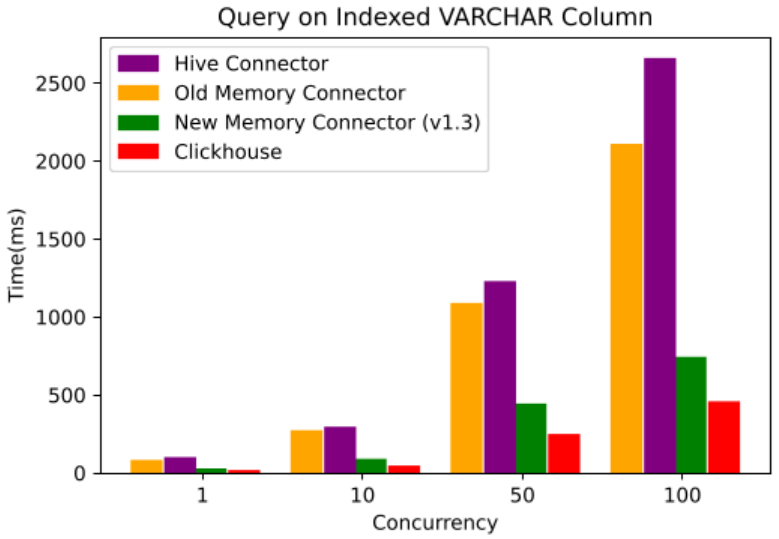
3. Performance improvement

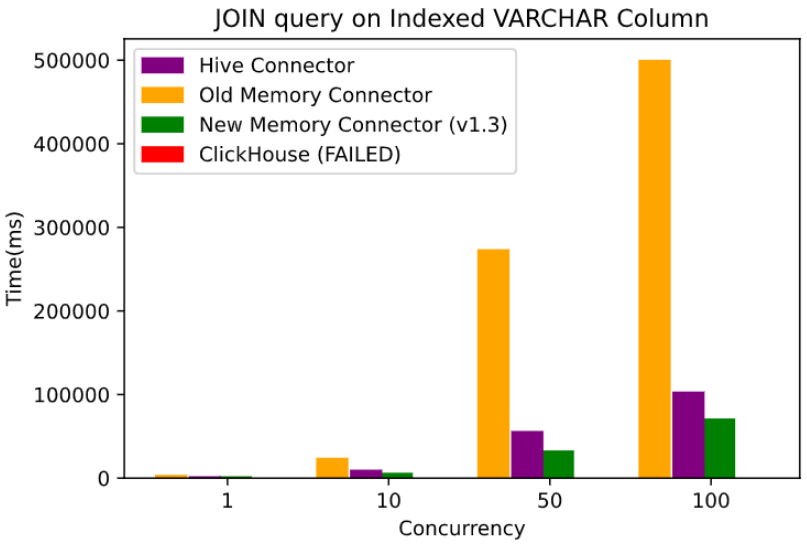












4. Future enhancements

- Support partitioning
  - allows filtering splits at schedule time
- Support bucketing
- Support additional pushdown operations
  - currently only predicate pushdown is supported, add support for aggregation pushdown, count pushdown, etc.
- Optimize split count dynamically
  - Small tables won’t end up in too many splits

5. Demo

- Configure memory connector

```
memory.properties:
    connector.name=memory
    memory.max-data-per-node=120MB
    memory.splits-per-node=6
    memory.spill-path=/tmp/mem
```

- Table creation with LP indexing

```
create table l with (sorted_by=array['orderkey']) as select * from tpch.tiny.lineitem;
create table l_not_ordered as select * from tpch.tiny.lineitem;
```

- Query data: applying index

```
select count(*) from l where orderkey < 5;
select count(*) from l_not_ordered where orderkey < 5;
```

- Query data: restore table after table is restarted
- LRU offload (creation of customer table takes ~120MB)

```
create table c as select * from tpch.sf1.customer;
```

openLooKeng

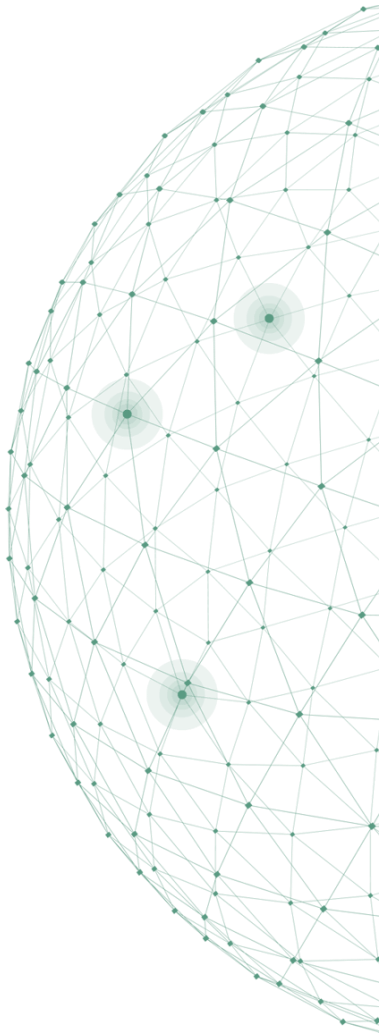
https://openlookeng.io

Part II.
openLooKeng Metastore

openLooKeng

https://openlookeng.io

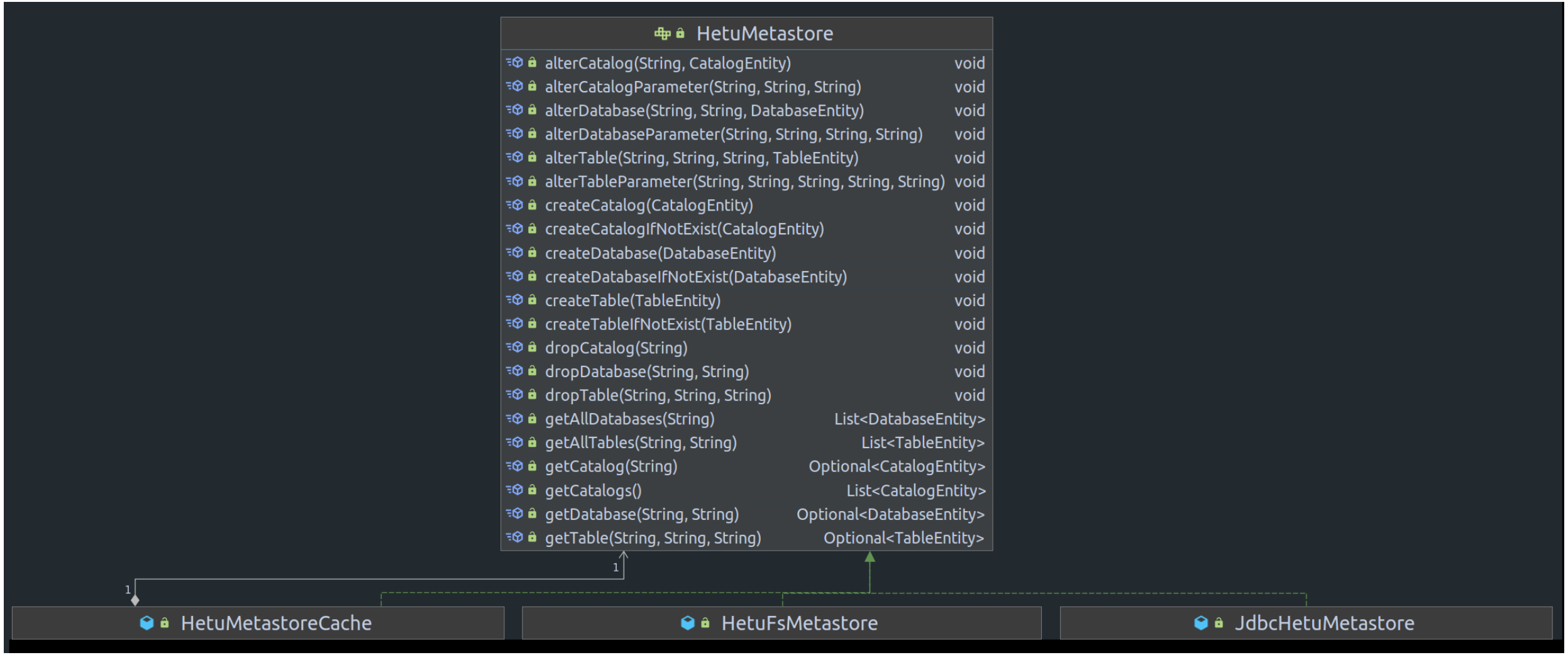
Content



openLooKeng

https://openlookeng.io

1. Design of hetu-metastore module



FSMetastore

- Dir structure storing catalogs/schemas/tables/columns
- Documental JSON files stores parameter and other metadata info

```
{
  "name" : “catalog1",
  "createTime" : 0,
  "comment" : "Hetu xxx connector",
  "parameters" : { }
}
```

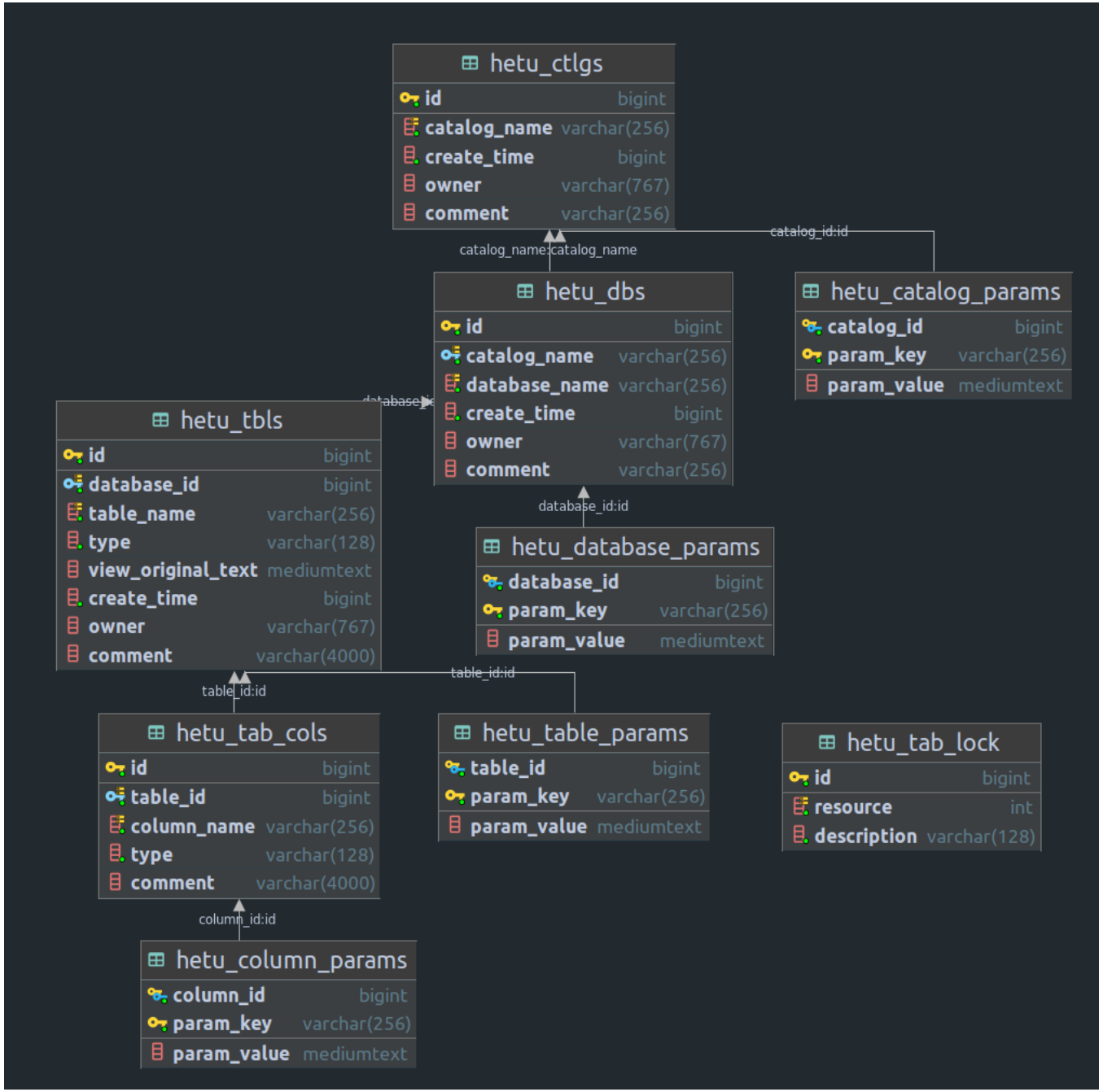
```
metastore-root
    catalog1.metadata
    catalog1/
        schema1.metadata
        schema1/
            table1.metadata
            table1/
                col1.metadata
                col1/
        schema2.metadata
        schema2/
            ...
    catalog2/
    ...
```

openLooKeng

https://openlookeng.io

JdbcMetastore

- Entity tables
  - Catalog
  - Schema
  - Table
  - Column
- Parameter tables
  - Catalog
  - Schema
  - Table
  - Column
- High concurrency and transaction handling



2. Usage

`etc/hetu-metastore.properties`

FSMetastore

```
hetu.metastore.type=hetufilesystem
hetu.metastore.hetufilesystem.profile-name=local-config-default
hetu.metastore.hetufilesystem.path=/tmp/openlookeng/metastore
```

JdbcMetastore

```
hetu.metastore.type=jdbc
hetu.metastore.db.url=jdbc:mysql://localhost:3306/hetu
hetu.metastore.db.user=root
hetu.metastore.db.password=mysql
```

2. Usage

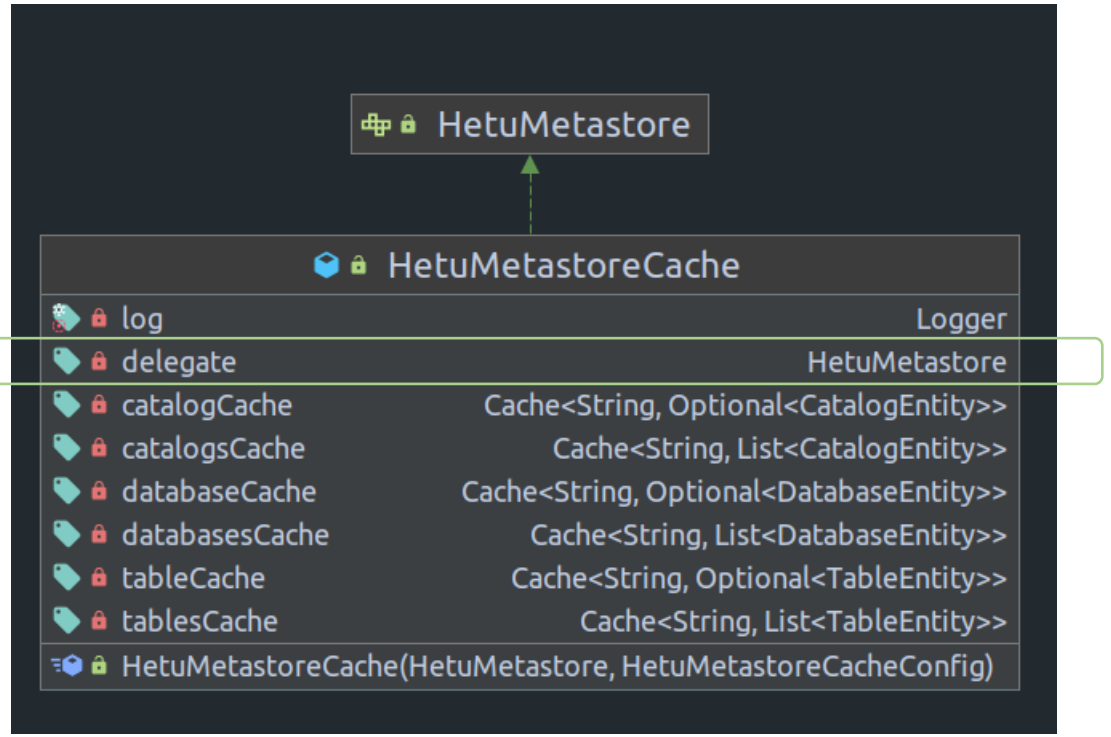
- Currently used by multiple modules:
  - hetu-heuristic-index
  - hetu-cube (star tree index)
  - hetu-vdm (dynamic catalog)
  - presto-memory
  - …

3. Improvement in v1.3.0: distributed cache

- Previous design: local delegate cache
- Only invalidates on modification

```java
@Override
public List<DatabaseEntity> getAllDatabases(String catalogName)
{
    try {
        return databasesCache.get(catalogName, () -> delegate.getAllDatabases(catalogName));
    }
    catch (ExecutionException executionException) {
        log.debug(executionException.getCause(),
                String.format("Error while caching all databases metadata " +
                        "in catalog[%s]. Falling back to default flow", catalogName));
        return delegate.getAllDatabases(catalogName);
    }
}
```

```java
@Override
public void createCatalog(CatalogEntity catalog)
{
    try {
        delegate.createCatalog(catalog);
    }
    finally {
        catalogsCache.invalidateAll();
        catalogCache.invalidate(catalog.getName());
    }
}
```

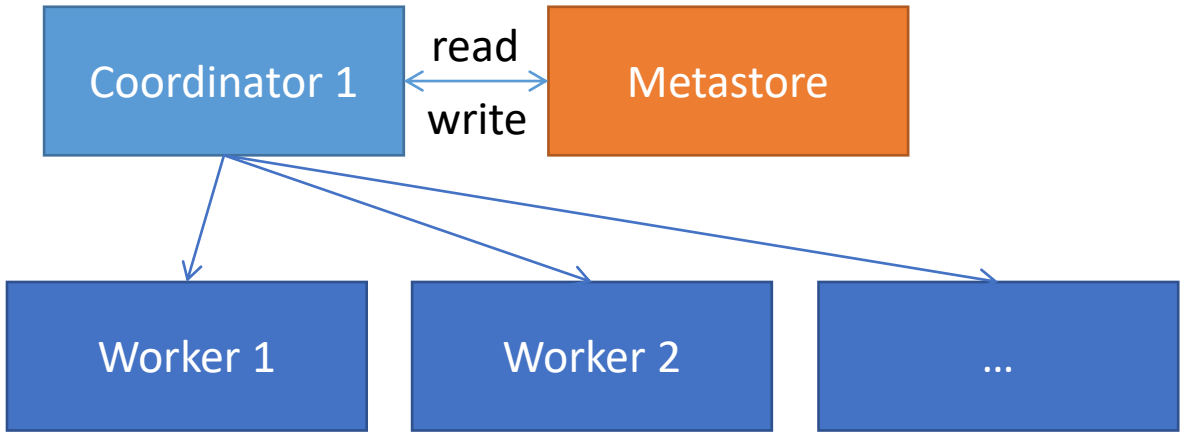


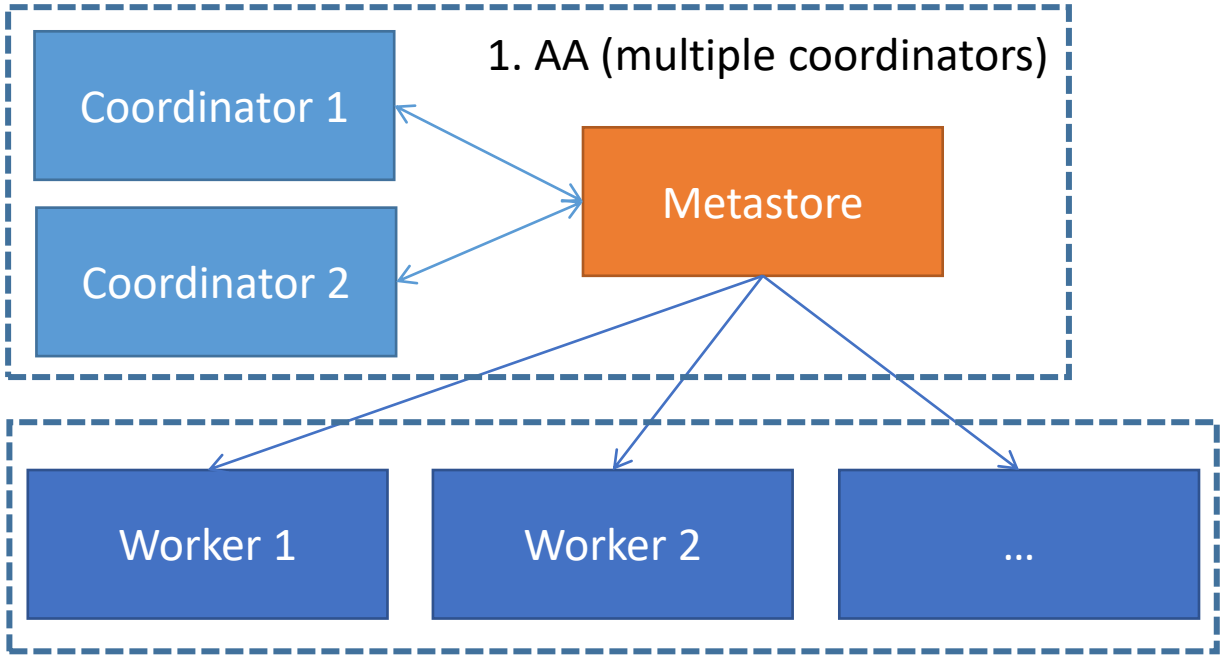
openLooKeng

https://openlookeng.io

3. Improvement in v1.3.0: distributed cache

- Metadata is stored across the cluster (HDFS/DB)
- Previously used local Guava cache
  - Previously metadata only used on coordinator
  - Cache not properly invalidated when another node makes changes



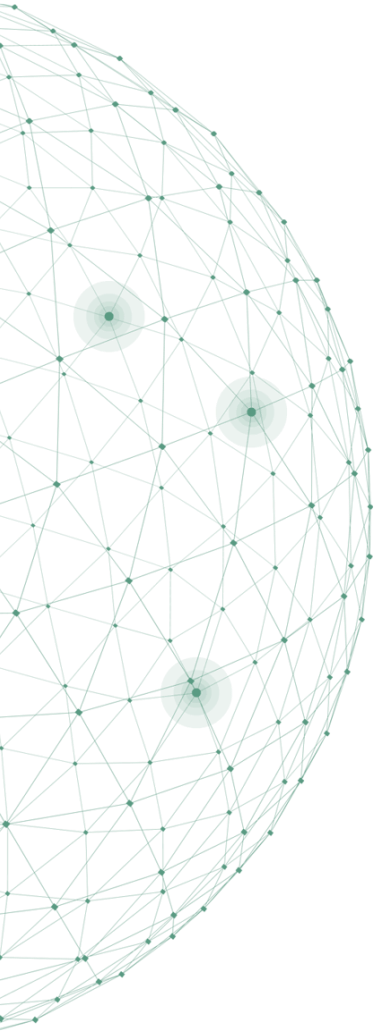


2. If worker needs to read from metastore as well

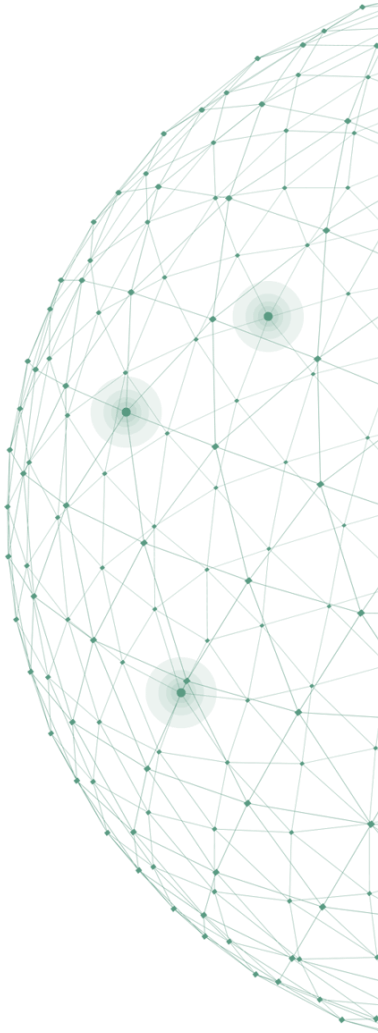
3. Improvement in v1.3.0: distributed cache

- Integrate hetu-state-store: cross-cluster sharing infrastructure
- Internally uses a Hazelcast cluster to maintain distributed cache
- Supports following use cases:
  - 1. Active-Active cluster: Multiple coordinators use metastore
  - 2. When worker needs to fetch information from metastore

https://docs.hazelcast.com/hazelcast/5.0-beta-1/index.html

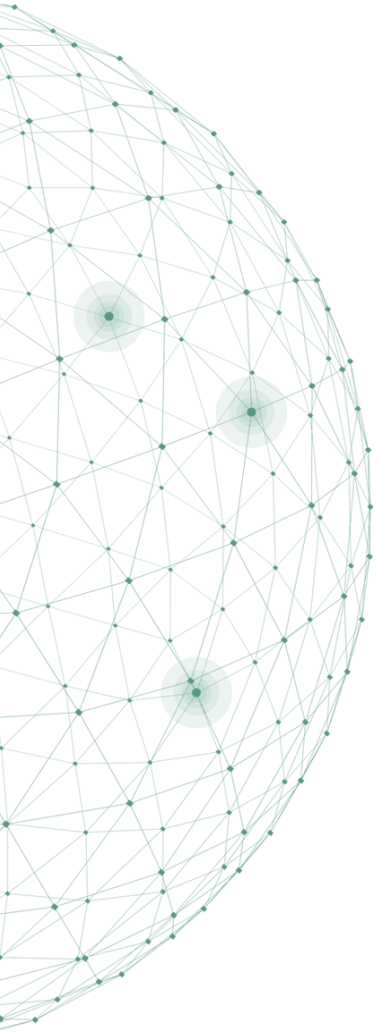


Thank you!

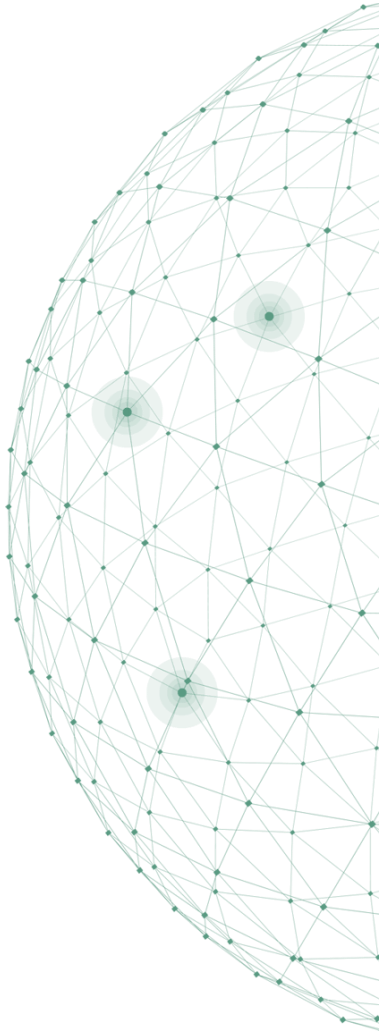


openLooKeng

https://openlookeng.io



Q & A



openLooKeng

https://openlookeng.io