Advanced usage of indexes in Oracle Coherence

Alexey Ragozin
alexey.ragozin@gmail.com
Nov 2011
• Structure of Coherence index
• How `IndexAwareFilter` works
• Multiple indexes in same query
• Custom index provider API (since 3.6)
• Embedding Apache Lucene into data grid
QueryMap.addIndex(
    ValueExtractor extractor, boolean ordered, Comparator comparator)

Attribute extractor, used to identify index later

Index configuration
public interface QueryMap extends Map {
    Set keySet(Filter f);
    Set entrySet(Filter f);
    Set entrySet(Filter f, Comparator c);
    ...
}

public interface InvocableMap extends Map {
    Map invokeAll(Filter f, EntryProcessor agent);
    Object aggregate(Filter f, EntryAggregator agent);
    ...
}
Indexes at storage node

Named cache backend

Backing map

Indexes

<table>
<thead>
<tr>
<th>extractor</th>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>extractor</td>
<td>index</td>
</tr>
<tr>
<td>extractor</td>
<td>index</td>
</tr>
</tbody>
</table>

SimpleMapIndex

Reverse map

val → key
val → key
val → key

Forward map

key → val
key → val
key → val
Indexes at storage node

- All indexes created on cache are stored in map

  Custom extractors should obey equals/hashCode contract!

- Reverse map is used to speed up filters
- Forward map is used to speed up aggregators

QueryMap.Entry.extract(...) is using index, if available
Indexes at storage node

- Index structures are stored in heap
  - and may consume a lot of memory
- For partitioned scheme
  - keys in index are **binary blobs**,  
  - **regular object**, otherwise
- Indexes will keep your key in heap even if you use off heap backing map
- Single index for all **primary partitions** of cache on single node
How filters use indexes?

```java
interface IndexAwareFilter extends EntryFilter {
    int calculateEffectiveness(Map im, Set keys);
    Filter applyIndex(Map im, Set keys);
}
```

- `applyIndex(...)` is called by cache service on top level filter
- `calculateEffectiveness(...)` may be called by compound filter on nested filters
- Each node executes index individually
- **For complex queries execution plan is calculated ad hoc, each compound filter calculates plan for nested filters**
Example: equalsFilter

Filter execution (call to applyIndex() )

- Lookup for matching index using extractor instance as key
- If index found,
  - lookup index reverse map for value
  - intersect provided candidate set with key set from reverse map
  - return null – candidate set is accurate, no object filtering required
- else (no index found)
  - return this – all entries from candidate set should be deserialized and evaluated by filter
Multiple indexes in same query

Example: `ticker=IBM & side=B`

```java
new AndFilter(
    new EqualsFilter("getTicker", "IBM"),
    new EqualsFilter("getSide", 'B'))
```

Execution plan

- call `applyIndex(...)` on first nested filter
  - only entries with ticker IBM are retained in candidate set
- call `applyIndex(...)` on second nested filter
  - only entries with side=B are retained in candidate set
- return candidate set
Index performance

**PROs**

- using of inverted index
- no deserialization overhead

**CONs**

- very simplistic cost model in index planner
- candidate set is stored in hash tables (intersections/unions may be expensive)
- high cardinality attributes may cause problems
Example: `ticker=IBM & side=B`

- Index per attribute
  ```java
  new AndFilter(
      new EqualsFilter("getTicker", "IBM"),
      new EqualsFilter("getSide", 'B'))
  ```

- Index for compound attribute
  ```java
  new EqualsFilter(
      new MultiExtractor("getTicker, getSide"),
      Arrays.asList(new Object[]{"IBM", 'B'}))
  ```

For index to be used, filter’s extractor should match extractor used to create index!
Ordered indexes vs. unordered

![Bar chart comparing ordered and unordered execution times for different term counts.](chart.png)

- **Term count = 100k**: Ordered (0.61 ms), Unordered (19.23 ms)
- **Term count = 10k**: Ordered (0.72 ms), Unordered (1.63 ms)
- **Term count = 2k**: Ordered (1.19 ms), Unordered (1.37 ms)
interface IndexAwareExtractor

    extends ValueExtractor {

        MapIndex createIndex(
            boolean ordered,
            Comparator comparator,
            Map indexMap,
            BackingMapContext bmc);

        MapIndex destroyIndex(Map indexMap);
    }
Ingredients of customs index

• Custom index extractor
• Custom index class (extends MapIndex)
• Custom filter, aware of custom index

+ Thread safe implementation
• Handle both binary and object keys gracefully
• Efficient insert (index is updates synchronously)
Custom index implementation is free to use any advanced data structure tailored for specific queries.

- NGram index – fast substring based lookup
- Apache Lucene index – full text search
- Time series index – managing versioned data
Using Apache Lucene in grid

Why?
• Full text search / rich queries
• Zero index maintenance

PROs
• Index partitioning by Coherence
• Faster execution of many complex queries

CONs
• Slower updates
• Text centric
Step 1. Create document extractor

// First, we need to define how our object will map
// to field in Lucene document
LuceneDocumentExtractor extractor = new LuceneDocumentExtractor();
extractor.addText("title", new ReflectionExtractor("getTitle"));
extractor.addText("author", new ReflectionExtractor("getAuthor"));
extractor.addText("content", new ReflectionExtractor("getContent"));
extractor.addText("tags", new ReflectionExtractor("getSearchableTags"));

Step 2. Create index on cache

// next create LuceneSearchFactory helper class
LuceneSearchFactory searchFactory = new LuceneSearchFactory(extractor);
// initialize index for cache, this operation actually tells coherence
// to create index structures on all storage enabled nodes
searchFactory.createIndex(cache);
Now you can use Lucene queries

// now index is ready and we can search Coherence cache
// using Lucene queries
PhraseQuery pq = new PhraseQuery();
pq.add(new Term("content", "Coherence"));
pq.add(new Term("content", "search"));
// Lucene filter is converted to Coherence filter
// by search factory
cache.keySet(searchFactory.createFilter(pq));
You can even combine it with normal filters

// You can also combine normal Coherence filters
// with Lucene queries
long startDate = System.currentTimeMillis() - 1000 * 60 * 60 * 24;
// last day
long endDate = System.currentTimeMillis();
BetweenFilter dateFilter = new BetweenFilter("getDateTime", startDate, endDate);
Filter pqFilter = searchFactory.createFilter(pq);
// Now we are selecting objects by Lucene query and apply
// standard Coherence filter over Lucene result set
cache.keySet(new AndFilter(pqFilter, dateFilter));
Lucene search performance

Filter execution time (ms)

- A1=x & E1=y: Coherence 0.72, Lucene 0.67
- E1=x & A1=y: Coherence 0.71, Lucene 0.67
- D1=x & E1=y: Coherence 1.10, Lucene 1.09
- E1=x & D1=y: Coherence 1.09, Lucene 1.10
- E1=x & E2=y: Coherence 3.30, Lucene 1.16
- E1=x & E2=Y & E3=z: Coherence 1.80, Lucene 7.23
- D1=w & E1=x & E2=Y & E3=z: Coherence 1.16, Lucene 1.49
- E1=x & E2=Y & E3=z & D1=w: Coherence 1.18, Lucene 1.49
- A2 in [n..m] & E1=x & E2=Y & E3=z: Coherence 11.15, Lucene 15.96
- E1=x & E2=Y & E3=z & A2 in [n..m]: Coherence 11.12, Lucene 15.96
- D1 in [v1..., v10] & E1=x & E2=Y & E3=z: Coherence 11.15, Lucene 15.96
- E1=x & E2=Y & E3=z & D1 in [v1..., v10]: Coherence 11.12, Lucene 15.96
- H1=a & E1=x & E2=Y & E3=z: Coherence 2.38, Lucene 2.38
- E1=x & E2=Y & E3=z & H1=a: Coherence 52.59, Lucene 52.59
Special index for managing versioned data

Entry key

Entry value

Series key Entry id Timestamp Payload

Cache entry

Getting last version for series $k$

```
select * from versions where series=k and version =
(select max(version) from versions where key=k)
```
Time series index

Series inverted index

HASHTABLE

Series key

Series key

Series key

Series key

Timestamp inverted subindex

ORDER

Timestamp → Entry ref

Timestamp → Entry ref

Timestamp → Entry ref
Thank you

http://aragozin.blogspot.com
- my articles
http://code.google.com/p/gridkit
- my open source code

Alexey Ragozin
alexey.ragozin@gmail.com