NoSQL Databases

Implemented in C++ and support C++ driver

Document: MongoDB

Key Value: Redis (unofficial driver), FoundationDB (C interface), BerkeleyDB

Wide Column: Hypertable

Key Value Store

Very simple and efficient and easy to scale. Uses a hash table in which there exists a unique key and a pointer to a particular item of data. Cache mechanisms enhances the performance a lot. But the data complexity is the limitation and when the volume of data increases, maintaining the hash table may be difficult.

Redis

1. Has complex data type (string, list, set, hase table)
2. Uses in memory store to speed up and has persistent disk storage. One million small pairs roughly use 100MB memory whose size can be configured. When runs out of memory errors may happen

Column Based

A row has many column families and each column family contains multiple columns. Storing data in cells of column has fast query in aggression.

Hypertable

1. An implementation of google bigtable. Uses HQL to query which is very similar to SQL.
2. Each cell has a timestamp to support different versions of table
3. For each column family different schema can be set like block size, how to build index, whether in memory
4. No support for join and transaction

Document Oriented

Quite similar to Key Value Store but the main difference is that embedded attribute metadata associate with content in the value section. Thus provide a way to query data based on content.

MongoDB

1. Support different storage allocation strategy, power of 2 or exact fit
2. Has no configurable cache meaning MongoDB use as much free memory as it can
3. MongoDB use the reader-writer lock to deal with the concurrent access and sometimes yield can happen.



1. Not support join. Use journal thus only support final consistency.

Graph Store, Neo4j

1. Real cool idea to handle real world data especially for highly connected data like social network.
2. Use a node as entity, an edge as relation. Thus the query is like traverse the graph whose model is really straightforward just like how to model the real world
3. No C++ implementation or driver. Only RESTful interface.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | MongoDB | BerkeleyDB | Redis | Hypertable |
| mem size | cannot customize.  use as much as possible | can be set  20MB when running 1000,000 record search with default setting | When using small key value pair(both integer), mem use is 70MB.  When use 200 length char string as value, mem use is 290 MB | 40MB but not too clear now |
| disk size | raw data 200M  storage size 430M | raw data 200M  storage size 310M | Compress are used.  Same value: 200MB(raw data)->18MB(disk)  Random value:200MB(raw data)->200MB(disk) | raw data 260M  storage size 510M |
| concurrency |  |  |  |  |
| performance (unique key search) | 0.0002s for each field | 0.0001s for key | 0.0001s | 0.0001s |
| different char set | UTF-8(default) | UTF-8, UTF-16BE or UTF-16-LE | UTF-8(default) | UTF-8(default) |
| ability to filter and query | No support join | No support join | No support join | No support join |

HyperdTable

In HQL there are three types of query sentence (select).

1. row predicate. Search by given row
2. cell predicate. Search by given row and column
3. column value predicate. Search by given column and value

Note.

1. Only row predicate support regular expression
2. Using column value predicate, given only column family won’t find the value under certain column. (e.g. a row ( ‘Key’, ‘ColumnFamily: Column’, ‘ Value’) , select \* from table where ‘ColumnFamily’ = ‘Value’ will find out nothing).
3. Using cell predicate given row and column family can find all the column within given row and column family.
4. Column value predicate does not support regular expression. Only support ‘start with’ and ‘less’ and ‘less or equal’ relation operation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **MongoDB - RegEx** | **MongoDB - Embedded** | **HyperTable - MultiVersion** | **HyperTable - MultiVersion** |
| **Query on “Title”**  Total Query Item: 5000  Total Result Item:11038 | normally stored as a field | normally stored as a field | stored in memory with index | stored in disk with index |
| Total Query Time | 0.398905 | 0.448331 | 0.475171 | 0.729707 |
| Each Query Time | 7.9781e-05 | 8.96662e-05 | 9.50342e-05 | 0.000145941 |
|  |  |  |  |  |
| **Query on “Genre”**  Total Query Item: 10  Total Result Item:1089 | Stored as a string  Query as RegEx | Stored as embedded structure  Query as embedded structure | stored in memory with index  as multiple version cell | stored in disk with index  as multiple version cell |
| Total Query Time | 0.001651 | 0.001582 | 0.00102 | 0.034811 |
| Each Query Time | 0.0001651 | 0.0001582 | 0.000102 | 0.0034811 |
|  |  |  |  |  |
| **Query on “Cast”** | Stored as a string  Query as RegEx  Total Query Item: 2684  Total Result Item:14821 | Stored as embedded structure  Query as embedded structure  Total Query Item: 2684  Total Result Item:8042 | stored in memory with index  as multiple version cell  Total Query Item: 2684  Total Result Item:8042 | stored in memory with index  as multiple version cell  same # |
| Total Query Time | 0.291555 | 0.219083 | 0.256505 | 0.605837 |
| Each Query Time | 0.000108627 | 8.16256e-05 | 9.55682e-05 | 0.000225722 |
|  |  |  |  |  |
| Raw Data Size | 2.3M | 2.3M | 2.3M | 2.3M |
| Storage Size | 10M | 10M | 5M | 5M |
| Memory Size | VIRT: 2459M  RES: 86M  MongoDB use as much free memory as it can | VIRT: 2459M  RES: 86M  MongoDB use as much free memory as it can | Multiple processes.. | Multiple processes. |