東京キャビネット  京都キャビネット
Tokyo Cabinet  Kyoto Cabinet

Katie Bambino
Marcelo Martins
CSCI2270
Tokyo Cabinet
Tokyo Family

- Tokyo Cabinet
  - Core DB library
- Tokyo Tyrant
  - Network accessible
- Tokyo Dystopia
  - Full Text Indexing/Search
- Tokyo Promenade
  - CMS
Tokyo Cabinet

- Modern implementation of DBM
  - e.g., NDBM, GDBM, TDBM, CDB, Berkeley DB, QDBM
- Library for managing key/value-type store
- High performance, efficient use of space
- C99 and POSIX compatible
- 64-bit architecture support
- Database size limit is 8EB
- LGPL
The High Points

- Multiple data storage options
  - Hash tables, B+-tree tables, fixed-length arrays
- Offers breadth of functionality
- Interfaces for several languages
  - Ruby, Java, Lua, and Perl
History

- 2001: Development of Estrainer using GDBM
- 2003: Development of QDBM, applied to Estrainer
- 2004: Development of Hyper Estrainer
- 2006: Joins Mixi.jp, production run of Hyper Estrainer
- 2007: Tokyo Cabinet development
- 2008: Tokyo Tyrant and Tokyo Distopia development
- 2010: Leaves Mixi.jp, founds FAL Labs
  - Releases Kyoto Cabinet
Features

- High concurrency
  - Multi-thread safe
  - read/write locking by records
- High scalability
  - Hash and B+-tree structures = $O(1)$ and $O(\log n)$
- Transactions
  - Write ahead logging and shadow paging
  - ACID properties (atomicity and durability)
- Various APIs
  - On-memory list/hash/tree
  - File hash/B+ tree/array/table
Data Storage Options – Hash Table

- Standard hash semantics
- Permits insert/lookup/delete and traversal of keys
- Unordered
- Fast operations
- $O(1)$ for retrieval, store and deletion
- Collision managed by separate chaining
Hash Table - Optimizations

- Chains are built from binary search trees
- Bucket array is mmap’ed
- Three modes for store:
  - Insert
  - Replace
  - Concatenate
- How to deal with fragmentation
  - Padding
  - Free block pool
Hash Table – Typical Use Cases

- Job/message queue
- Sub-index of relational database
- Dictionary of words
- Inverted index for full-text search
- Temporary storage for map-reduce
- Archive of many small files
Hash Table – Tuning

- **bnum** - Specifies the number of elements to use in the bucket array.
- **rcnum** - Specifies the maximum number of records to be cached.
Data Storage Options – B+ Tree

- Keys can be duplicated
- Records stored in order
- Same operations of HT, plus range queries
- Inserts are fast, but lookup is slower than HT
- More space-efficient than HT
require "rubygems"
require "tokyocabinet"

include TokyoCabinet

bdb = BDB::new  # B-Tree database; keys may have multiple values
bdb.open("casket.bdb", BDB::OWRITER | BDB::OCREAT)

# store records in the database, allowing duplicates
bdb.putdup("key1", "value1")
bdb.putdup("key1", "value2")
bdb.put("key2", "value3")
bdb.put("key3", "value4")

# retrieve all values
p bdb.getlist("key1")
# => ["value1", "value2"]

# range query, find all matching keys
p bdb.range("key1", true, "key3", true)
B+ Tree - Optimizations

• Records are stored and arranged in nodes
• Sparse index for accessing nodes in memory
• Each leaf node is stored on disk as a hash table record
• Nodes can be compressed using ZLIB or BZIP2
  • Size can be reduced to about 25%
B+ Tree – Typical Use Cases

- Session management for a web service
- User account database
- Document database
- Access counter
- Cache of CMS
- Graph/text mining
B+ Tree – Tuning

- **bnum** - Specifies the number of elements to use in the bucket array.
- **cmpfunc** - Specifies the comparison function used to order B+Tree Databases.
- **lmemb (nmemb)** - Specifies the number of members in each leaf (non-leaf) page.
- **lcnum (ncnum)** - Specifies the maximum number of leaf (non-leaf) nodes to be cached.
Data Storage Options – Fixed-Length Array

- Keyed by unique integers
- Fixed record size – limited length for each value
- Fastest insert/lookup
- Uses mmap() to reduce file I/O overhead
  - Multiple processes share same memory space
• **width** - Specifies the width of values (255 by default).
  - Anything beyond specified length will be silently discarded.
• **limsiz** - Specifies the limit on database file size in bytes (268435456 by default).
  - Setting width = 1024 and limsiz = 1024 * 4, will produce a database that holds only 4 keys.
Data Storage Options – Table Database

- Built out of other table types
- Free form-schema, resembles document-oriented DB
- Permits sophisticated querying
- Arbitrary indexes on columns
- Slower, but easy to use
require "rubygems"
require "rufus/tokyo/cabinet/table"

t = Rufus::Tokyo::Table.new('table.tdb', :create, :write)

# populate table with arbitrary data (no schema!)
t['pk0'] = { 'name' => 'alfred', 'age' => '22', 'sex' => 'male' }
t['pk1'] = { 'name' => 'bob', 'age' => '18' }
t['pk2'] = { 'name' => 'charly', 'age' => '45', 'nickname' => 'charlie' }
t['pk3'] = { 'name' => 'doug', 'age' => '77' }
t['pk4'] = { 'name' => 'ephrem', 'age' => '32' }

# query table for age >= 32
p t.query { |q|
  q.add_condition 'age', :numge, '32'
  q.order_by 'age'
}

# => [ {"name":"ephrem", :pk=>"pk4", "age"=>"32"},
#       {"name":"charly", :pk=>"pk2", "nickname"=>"charlie",
#        "age"=>"45"},
#       {"name":"doug", :pk=>"pk3", "age"=>"77"} ]
ACID Properties: Atomicity

- Transactions
- Isolation levels:
  - Serializable
  - Read uncommitted
- Locking granularity
  - Per record
    - Hash database
    - Fixed-length database
  - Per file
    - others
ACID Properties: Durability

Shadow paging (COW)

Write-ahead logging

Space-saving
Point-in-time,
Copy-on-write methodology

Only block-level differentials
no full-copy
Tokyo Tyrant

- Network interface for Tokyo Cabinet DB
- Turns TC into a database server
- Client/server model
- Multiple applications can access one database
TT Features

- High concurrency via thread pool
- Speaks three different protocols: binary, memcached, and HTTP
- Uses abstract API to converse with internal storage
- Embedded Lua scripts
Master-slave(s) topology

- All participants must record the update log
- Each server must have a unique ID
Replication II

Dual master

- Reciprocal replication
- May cause inconsistencies
Replication II

Diagram showing a client connecting to a master server (active) and a master server (standby), with slave servers and databases. The diagram illustrates load balancing, two-way replication (dual-master), and one-way replication (master-slave).
Tokyo vs. DBM Family (time)

- Write Time (s)
- Read Time (s)
Tokyo vs. DBM Family (file size)

File size (bytes)
Tokyo vs. NoSQL (qualitative)

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Tokyo Cabinet + Tokyo Tyrant</th>
<th>Berkeley DB + MemcacheDB</th>
<th>Voldemort + BDBJE</th>
<th>Redis</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion (small data set)</td>
<td>🌟🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Insertion (large data set)</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Random Read (small data set)</td>
<td>🌟🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Random Read (large data set)</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Speed Consistence</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Storage Efficiency</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Horizontal Scalability</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Manageability</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Stability</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Feature Set</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Project Activeness and Community Support</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
<td>🌟🌟🌟🌟</td>
</tr>
</tbody>
</table>

http://perfectmarket.com/blog/not_only_nosql_review_solution_evaluation_guide_chart
Tokyo vs. NoSQL (Small data)

- “2.8 million records (6GB) were loaded, and then a half million records were retrieved from the database”

<table>
<thead>
<tr>
<th>Database</th>
<th>Load time</th>
<th>Retrieval time</th>
<th>File size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokyo Cabinet/Tyrant</td>
<td>12 minutes</td>
<td>3 1/2 minutes</td>
<td>24MB</td>
</tr>
<tr>
<td>CouchDB</td>
<td>22 hours</td>
<td>14 1/2 minutes</td>
<td>236MB</td>
</tr>
<tr>
<td>MongoDB</td>
<td>3 minutes</td>
<td>4 minutes</td>
<td>192-960MB</td>
</tr>
</tbody>
</table>
Case Study: Storage cache at mixi.jp

- Work as proxy
  - Mediate insert/search
- Lua extension
  - Atomic access per record
  - Uses LuaSocket to access storage
- Proper DB scheme
  - On-memory hash: suitable for generic cache
  - File hash table: suitable for large records, e.g., images
  - File fixed array: suitable for small, fixed-length records, e.g., timestamps
Case Study: Ravelry

- Online knit and crochet community
  - Organizational tool
  - Yarn/pattern database
  - Social site: forums, groups, friend-related features
- Ruby on Rails
- 70,000 DAU (2009)
- 3.6 million pageviews per day (2009)

- Uses Tokyo Cabinet/Tyrant to cache larger objects
  - Tons of rendering Markdown into HTML
  - Too large to store in memcached
Casey Forbes, Ravelry’s only developer, on TC/TT:

- “I think it is a very nice solution for storing large chunks of HTML, etc. — MySQL is not a very good solution for this (waste of InnoDB buffer pools, lots of growth in database files, less than ideal performance when dealing with large tables of blobs) and memcached can become full very fast depending on how much memory you have to devote to caches.”

- “We’ve stored up to 25 GB but we are currently storing 10 GB of data. Performance is so close to memcached (even though it hits the disk) that speed is really a non-issue.”
Kyoto Cabinet
Kyoto Cabinet

- Very similar to Tokyo Cabinet
- Dropped support for fixed-length and table databases
- Support for external compression
  - LZO and LZMA
- Support for atomic-increment and CAS
- Supports Win32
- License: GPLv3 (TC: LGPL)
Kyoto Cabinet vs. Tokyo Cabinet

- Better performance and concurrency
  - Parallelism in a multithreaded environment
  - Decreased efficiency per thread due to grained locking
  - User-land locking by CAS
- Space efficiency
  - 16B footprint/record (vs. TC’s 22B)
- Robustness
  - Auto-transaction
  - Auto-recovery
- New database types
  - Four new on-memory databases
  - Two new file-based databases
<table>
<thead>
<tr>
<th>Name</th>
<th>Data structure</th>
<th>Complexity</th>
<th>Ordering</th>
<th>Locking</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proto HashDB</td>
<td>Hash table</td>
<td>O(1)</td>
<td>None</td>
<td>File (rwlock)</td>
<td>None (testing)</td>
</tr>
<tr>
<td>Proto TreeDB</td>
<td>Red black tree</td>
<td>O(log n)</td>
<td>Lexical</td>
<td>File (rwlock)</td>
<td>Ordered records</td>
</tr>
<tr>
<td>StashDB</td>
<td>Hash table</td>
<td>O(1)</td>
<td>None</td>
<td>Record (rwlock)</td>
<td></td>
</tr>
<tr>
<td>CacheDB</td>
<td>Hash table</td>
<td>O(1)</td>
<td>None</td>
<td>Record (mutex)</td>
<td>General caching</td>
</tr>
<tr>
<td>GrassDB</td>
<td>B+ tree</td>
<td>O(log n)</td>
<td>Custom</td>
<td>Page (rwlock)</td>
<td></td>
</tr>
</tbody>
</table>
Choosing the Right Tool

- No persistence required? On-memory DB
- If order is important, use cache tree DB (GrassDB)
- Time efficiency: CacheDB > StashDB > ProtoHashDB > ProtoTreeDB > GrassDB
- Space efficiency: GrassDB > StashDB > CacheDB > ProtoHashDB > ProtoTreeDB
Auto Snapshot

- Similar to the one in Redis
- Periodically saves on-memory records into files
- Thanks to COW, each snapshot operation is performed atomically

<table>
<thead>
<tr>
<th>Format</th>
<th>Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw</td>
<td>22.888MB</td>
<td>0.322s</td>
</tr>
<tr>
<td>LZO</td>
<td>10.215MB</td>
<td>0.411s</td>
</tr>
<tr>
<td>ZLIB</td>
<td>6.367MB</td>
<td>2.010s</td>
</tr>
<tr>
<td>LZMA</td>
<td>2.787MB</td>
<td>17.619s</td>
</tr>
</tbody>
</table>
## Storage Options - File

<table>
<thead>
<tr>
<th>Name</th>
<th>Data structure</th>
<th>Complexity</th>
<th>Ordering</th>
<th>Locking</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>HashDB</td>
<td>Hash table</td>
<td>O(1)</td>
<td>None</td>
<td>Record (rwlock)</td>
<td>Small, but numerous metadata</td>
</tr>
<tr>
<td>TreeDB</td>
<td>B+ tree</td>
<td>O(log (n))</td>
<td>Custom</td>
<td>Page (rwlock)</td>
<td>Small, but numerous metadata, ordered</td>
</tr>
<tr>
<td>DirDB</td>
<td>Undefined</td>
<td>Undefined</td>
<td>None</td>
<td>Record (rwlock)</td>
<td>Large but few data</td>
</tr>
<tr>
<td>ForestDB</td>
<td>B+ tree</td>
<td>O(log (n))</td>
<td>Custom</td>
<td>Page (rwlock)</td>
<td>Large and many data, ordered</td>
</tr>
</tbody>
</table>
Kyoto Tycoon

- Persistent cache server
- High concurrency
  - 1M queries / 25 sec = 40,000 QPS or more
- Supports auto-expiration mechanism
- Discarded replication mechanism
- Like TT and memcached, no data sharding
- Usage: large web services
  - Access counters
  - Time stamp trackers
  - User account managers
  - Session data
Auto Expiration

- Expiration time given to each record
- Records removed timer expiration
- "GC cursor" eliminates expired regions gradually
Primitive Map-Reduce with Tokyo Tyrant

- Lua Extension
- Defines DB operations as Lua functions
- Client sends function name, plus key/value
- Server returns function result
function wordcount()
    function mapper(key, value, mapemit)
        for word in string.gmatch(string.lower(value), "%w+")do
            mapemit(word, 1)
        end
        return true
    end
    local res = ""
    function reducer(key, values)
        res = res .. key .. "\t" .. #values .. "\n"
        return true
    end
    if not _mapreduce(mapper, reducer) then
        res = nil
    end
    return res
end
function wordcount()
    function mapper(key, value, mapemit)
        for word in string.gmatch(string.lower(value), "\%w+\")do
            mapemit(word, 1)
        end
        return true
    end

    local res = ""
    function reducer(key, values)
        res = res .. key .. "\t" .. #values .. "\n"
        return true
    end

    if not _mapreduce(mapper, reducer) then
        res = nil
    end

    return res
end
Map-Reduce with Tokyo Tyrant (III)
Kyoto Cabinet - Visitor Pattern

1: defines arbitrary operation

```cpp
class MyVisitor : public DB::Visitor {
    char* visit_full(const char* kbuf, size_t ksiz,
                     const char* vbuf, size_t vsiz,
                     size_t sp) { ... }
    char* visit_empty(const char* kbuf, size_t ksiz,
                      size_t sp) { ... }
};
```

2: gives the visitor to the DB

3: call the record operation atomically

Application

Database