DESIGNING DISTRIBUTED SYSTEMS for: HEATHCARE and MEDICAL DBMS

BY: Rajat Bisht

INTRODUCTION

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- **Goal 1**: This project tries to replicate Facebook's (now Meta) approach in creating a cross country distributed system.

INTRODUCTION

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- **Goal 1**: This project tries to replicate Facebook's (now Meta) approach in creating a cross country distributed system.
- **Goal 2**: implement a read driven Memcache implementation.

INTRODUCTION

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- **Goal 1**: This project tries to replicate Facebook's (now Meta) approach in creating a cross country distributed system.
- **Goal 2**: implement a read driven Memcache implementation.
- **Goal 3**: implement a backend database for consistency.

INTRODUCTION

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- **Goal 1**: This project tries to replicate Facebook's (now Meta) approach in creating a cross country distributed system.
- **Goal 2**: implement a read driven Memcache implementation.
- **Goal 3**: implement a backend database for consistency.
- **Goal 4:** cache is cleared as soon as there is an update in the data base tables.

INTRODUCTION

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- **Goal 1**: This project tries to replicate Facebook's (now Meta) approach in creating a cross country distributed system.
- **Goal 2**: implement a read driven Memcache implementation.
- **Goal 3**: implement a backend database for consistency.
- **Goal 4:** cache is cleared as soon as there is an update in the data base tables.
- **Goal 5:** cache invalidation also takes place internally by using least frequent used (LFU).

INTRODUCTION

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- **Goal 1**: This project tries to replicate Facebook's (now Meta) approach in creating a cross country distributed system.
- **Goal 2**: implement a read driven Memcache implementation.
- **Goal 3**: implement a backend database for consistency.
- **Goal 4:** cache is cleared as soon as there is an update in the data base tables.
- **Goal 5:** cache invalidation also takes place internally by using least frequent used (LFU).
- **Goal 6:** implemented my own mc_router for implementing consensus data updating.

INTRODUCTION

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- **Goal 1**: This project tries to replicate Facebook's (now Meta) approach in creating a cross country distributed system.
- **Goal 2**: implement a read driven Memcache implementation.
- **Goal 3**: implement a backend database for consistency.
- **Goal 4:** cache is cleared as soon as there is an update in the data base tables.
- **Goal 5:** cache invalidation also takes place internally by using least frequent used (LFU).
- **Goal 6:** implemented my own mc_router for implementing consensus data updating.

DBMS & SQLITE3

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- DBMS: database management system is used in all the distributed system.

DBMS & SQLITE3

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- DBMS: database management system is used in all the distributed system.
- SQLITE3 is chosen because there isn’t much written data on its handling using C/CPP and also its free and open source.

DBMS & SQLITE3

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- DBMS: database management system is used in all the distributed system.
- SQLITE3 is chosen because there isn’t much written data on its handling using C/CPP and also its free and open source.
- Sqlite3 comes with a pretty redundant API library which provided no return value feature.

DBMS & SQLITE3

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- DBMS: database management system is used in all the distributed system.
- SQLITE3 is chosen because there isn't much written data on its handling using C/CPP and also its free and open source.
- Sqlite3 comes with a pretty redundant API library which provided no return value feature.
- To run Sqlite3, it is prerequisite to install its library file <sqlite3.h> to the "include" folder.

MEMCACHED

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Memcache implementation is done by using “std::map”

MEMCACHED

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Memcache implementation is done by using “std::map”
- After reading any table, its data is stored inside a map<int , table_type > map_LFU .

MEMCACHED

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Memcache implementation is done by using “std::map”
- After reading any table, its data is stored inside a map<int , table_type > map_LFU .
- LFU map holds all the recently read data from database.

MEMCACHED

- Memcache implementation is done by using “std::map”
- After reading any table, its data is stored inside a map<int , table_type > map_LFU .
- LFU map holds all the recently read data from database.
- When chache flag is updated a “particular” number of times, it becomes “Frequent” and is moved to “MFU” map.

MEMCACHED

- Memcache implementation is done by using “std::map”
- After reading any table, its data is stored inside a map<int , table_type > map_LFU .
- LFU map holds all the recently read data from database.
- When chache flag is updated a “particular” number of times, it becomes “Frequent” and is moved to “MFU” map.
- Invalidation is again based on update or system generated internal cleanup.

MEMCACHED

- Memcache implementation is done by using “std::map”
- After reading any table, its data is stored inside a map<int , table_type > map_LFU .
- LFU map holds all the recently read data from database.
- When chache flag is updated a “particular” number of times, it becomes “Frequent” and is moved to “MFU” map.
- Invalidation is again based on update or system generated internal cleanup.
- When a value is updated to the MFU map, it is removed from LFU map.

THREADING for SERVER

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Threading is implemented for handling multiple requests.

THREADING for SERVER

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Threading is implemented for handling multiple requests.
- Reading from theSqlite3 database does not require lock.

THREADING for SERVER

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Threading is implemented for handling multiple requests.
- Reading from theSqlite3 database does not require lock.
- But any write operation requires a lock to be implemented.

THREADING for SERVER

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Threading is implemented for handling multiple requests.
- Reading from theSqlite3 database does not require lock.
- But any write operation requires a lock to be implemented.
- Thus I used one mutex lock thread for updating so that there are no race conditions.

THREADING for SERVER

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Threading is implemented for handling multiple requests.
- Reading from theSqlite3 database does not require lock.
- But any write operation requires a lock to be implemented.
- Thus I used one mutex lock thread for updating so that there are no race conditions.
- Peer implementation using a thread for constant acknowledgement felt like a waste of resource. So there is an internal update call implemented. See file “ServerThread.cpp” for more clarity.

READ DATA

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Reading data is fairly simple.

READ DATA

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Reading data is fairly simple.
- Each request is opened up in its own thread.

READ DATA

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Reading data is fairly simple.
- Each request is opened up in its own thread.
- Each thread processes the request type and the data type requested from the server.

READ DATA

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Reading data is fairly simple.
- Each request is opened up in its own thread.
- Each thread processes the request type and the data type requested from the server.
- After this information flows like this:

READ DATA

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Reading data is fairly simple.
- Each request is opened up in its own thread.
- Each thread processes the request type and the data type requested from the server.
- After this information flows like this:
  check if it is already a MFU_map value?

READ DATA

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Reading data is fairly simple.
- Each request is opened up in its own thread.
- Each thread processes the request type and the data type requested from the server.
- After this information flows like this:
  check if it is already a MFU_map value?
  check if it a LFU_map value?

READ DATA

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Reading data is fairly simple.
- Each request is opened up in its own thread.
- Each thread processes the request type and the data type requested from the server.
- After this information flows like this:
  check if it is already a MFU_map value?
  check if it a LFU_map value?
  if no then Read the value from database

READ DATA

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Reading data is fairly simple.
- Each request is opened up in its own thread.
- Each thread processes the request type and the data type requested from the server.
- After this information flows like this:

  check if it is already a MFU_map value?
  check if it a LFU_map value?
  if no then Read the value from database
  update it in LFU_map

READ DATA

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Reading data is fairly simple.
- Each request is opened up in its own thread.
- Each thread processes the request type and the data type requested from the server.
- After this information flows like this:

  check if it is already a MFU_map value?
  check if it a LFU_map value?
  if no then Read the value from database
  update it in LFU_map
  if (this value pinged frequently)

READ DATA

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Reading data is fairly simple.
- Each request is opened up in its own thread.
- Each thread processes the request type and the data type requested from the server.
- After this information flows like this:

  check if it is already a MFU_map value?
  check if it a LFU_map value?
  if no then Read the value from database
  update it in LFU_map
  if (this value pinged frequently)
  update it to the MFU_map

UPDATE DATA

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Updating data is done a way very similar to Facebook's Memcached and mc_router implementation.

UPDATE DATA

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Updating data is done a way very similar to Facebook's Memcached and mc_router implementation.
- When a client makes an update request it is routed directly to the main server.

UPDATE DATA

- Updating data is done a way very similar to Facebook's Memcached and mc_router implementation.
- When a client makes an update request it is routed directly to the main server.
- Then main server checks if the update request is valid and updates its own database.

UPDATE DATA

- Updating data is done a way very similar to Facebook's Memcached and mc_router implementation.
- When a client makes an update request it is routed directly to the main server.
- Then main server checks if the update request is valid and updates its own database.
- After updating its own , it sends a update request from its end directly to both the backup servers and waits for them to send acknowledgement of the update.

UPDATE DATA

- Updating data is done a way very similar to Facebook's Memcached and mc_router implementation.
- When a client makes an update request it is routed directly to the main server.
- Then main server checks if the update request is valid and updates its own database.
- After updating its own , it sends a update request from its end directly to both the backup servers and waits for them to send acknowledgement of the update.
- If both the updates are acknowledged back; a final acknowledgement is sent to the client making request.

LESSONS LEARNED

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Sqlite3 DB consumed most of the time.

LESSONS LEARNED

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- Sqlite3 DB consumed most of the time.
- **Integrating** it with a object oriented programming architecture becomes limiting.

LESSONS LEARNED

- Sqlite3 DB consumed most of the time.
- **Integrating** it with a object oriented programming architecture becomes limiting.
- **Callback function**: this callback function is a static int type and is called repeatedly for each row read from the database.

LESSONS LEARNED

- Sqlite3 DB consumed most of the time.
- **Integrating** it with a object oriented programming architecture becomes limiting.
- **Callback function**: this callback function is a static int type and is called repeatedly for each row read from the database.
- In all the documentations found over the internet for implementing it were **redundant** and **vague**. I am planning to update my work on GitHub and provide with solution to the problems I faced on various discussion channels.

LESSONS LEARNED

- Sqlite3 DB consumed most of the time.
- **Integrating** it with a object oriented programming architecture becomes limiting.
- **Callback function**: this callback function is a static int type and is called repeatedly for each row read from the database.
- In all the documentations found over the internet for implementing it were **redundant** and **vague**. I am planning to update my work on GitHub and provide with solution to the problems I faced on various discussion channels.
- **Fascinating** , the READ table is race condition proof. This meant that since my system is basically read driven, it is fast.

ASPIRATIONS

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- This project is currently a proof of concept for a global deployable system.

ASPIRATIONS

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- This project is currently a proof of concept for a global deployable system.
- Future plans involve implementing a python based Flask client for general deplorability and testing over various inter device communication channels.

ASPIRATIONS

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- This project is currently a proof of concept for a global deployable system.
- Future plans involve implementing a python based Flask client for general deplorability and testing over various inter device communication channels.
- Write(update) and read are included in the distributed code.

ASPIRATIONS

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- This project is currently a proof of concept for a global deployable system.
- Future plans involve implementing a python based Flask client for general deplorability and testing over various inter device communication channels.
- Write(update) and read are included in the distributed code.
- Inserting and deleting data from the database is similar to implementing above calls.

ASPIRATIONS

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- This project is currently a proof of concept for a global deployable system.
- Future plans involve implementing a python based Flask client for general deplorability and testing over various inter device communication channels.
- Write(update) and read are included in the distributed code.
- Inserting and deleting data from the database is similar to implementing above calls.
- Classes and functions are already defined in the source code if anyone wants to test and prod.

ASPIRATIONS

1. Introduction
2. DBMS & SQLITE3
3. Memcached
4. Threading for server
5. Read data
6. Update data
7. Lessons learned
8. Aspirations

- This project is currently a proof of concept for a global deployable system.
- Future plans involve implementing a python based Flask client for general deplorability and testing over various inter device communication channels.
- Write(update) and read are included in the distributed code.
- Inserting and deleting data from the database is similar to implementing above calls.
- Classes and functions are already defined in the source code if anyone wants to test and prod.
- Suggestions are accepted. Please do share your reviews on Microsoft teams.

Thank You