CSE 681- Software Modelling and Analysis

Project #3
Remote Key/Value NoSQL Database
Operational Concept Document
Version 1.0

Achal Velani

SUID: 292190387

October 21, 2015

Instructor: Jim Fawcett

Table of Content

1. Executive Summary

“Remote NoSQL database system” proposes a system which overcomes the problems of conventional SQL database system and has more relaxed or extinct data model restriction, elastic scaling, better functioning with large datasets and easy remote access to this database using simple GUI at client side.

This document essentially presents an architecture for this system which contains a NoSQL database at the core of it. The main focus of the system is on storing huge sets of data, manipulating the data in the database, parsing and storing the data entries in XML and retrieval of the same when desired remotely. Moreover, this will also support different types of queries on the database from remote clients. To implement this functionality we are going to take different read and write clients which can access the database (server) through sender and receiver using WCF to communicate.

Critical issues faced during design are: constructing an API which exhibits all the requirement upon execution, data consistency, authentication, data privacy, locking issues, and performance testing and communication errors. Data consistency is not an issue specific to only this system but it is inherited from NoSQL database, as a partial solution to this we could persist the in-memory data and also use replication in big systems this would not however make system foolproof of this issue but make it “eventual consistent”. The issue of authentication comes into play when remote access is implemented and the solution to this could be carried out by proposing a small system which creates username and passcode for each remote server and explicitly provide access to them through a secured link. Locking is an issue which is related to simultaneous access to the database, when more than one clients are trying to access of modify the same data the database could become corrupted this could be resolved by providing token scheme to access the data. Latency is time lag between initiated request and its response, as far as software is concerned this critical issue can be addressed by implementing optimized message formation system and performance testing could be done on the system test different real world scenarios. We rely on some kind of network to make the communication possible in between client and server, this network can have some inherent issues which could be dealt by using checksum in the message. We explore all these issues along with the architecture in depth in this documentation.

2. Introduction

In this project we are creating a system of Remote Key/Value NoSQL database and propose functionalities such as performing manipulation, management and retrieval of requested data on remote client side from the server side which has NoSQL database. The main purpose of creating the NoSQL database is the shortcomings of conventional SQL database which has unacceptable performance issues while working with enormous data sizes and it could be only solved by proposing a database which has no defined structure and could respond the incoming queries and data retrieval requests with same response time without any constrains of the size of the database, and hence the NoSQL database comes in the play, and we are extending its usability by providing remote access to this database.

The major responsibilities of this project is to implement a generic key/value in-memory database with specified parameters under the metadata, support for different types of manipulations on this data such as addition and deletion of the key/value pair, editing of values of the parameters. Moreover this database provides functionality of storing and retrieving the in-mem data to and from an XML file and we are also hard coding a scheduler in the system which will store the in-mem data to the XML file after a fixed pre-defined time. As the main purpose of any database is to process queries and respond to the requests, this NoSQL database of ours also support different types of queries such as getting values or the children of a specified key, getting set of keys which matches specified patterns and also getting keys for a defined time stamp. Furthermore all these functionalities are extended to remote clients which uses all these implementations remotely through WCF message passing system.

The NoSQL database which is at the core of Remote Key/Value Database has higher performance, scalability and flexibility as it has some inherent characteristics unlike SQL database; for example scaling out in contrast to scaling up, distributed query support, easy load balancing by using sharding, clustering and replication. Replication is easy, as NoSQL database uses cheap commodity servers in comparison to high specification servers used by RDBMS. Also the licensing cost of NoSQL database is fairly low as they generally tend to be open source, and priced to function on cluster of server. All in all this makes this types of database achieve zero downtime with very less efforts. Sharding is also a very important functionality of the NoSQL database as this keeps server sizes comparatively small and encourages to use low priced servers for the same or better level of overall performance in contrast to sequential database.

Overall availability of the electronic devices, high speed and uninterrupted internet connectivity have exposed the information age to a new level of possibilities where very huge size data storing and processing could be achieved with use of NoSQL database. The major applications concerned with NoSQL database are Big Users, Big Data, Cloud Computing and Internet of Things. All of these applications take scalability, performance and distant accessibility advantage of this non-conventional database system.

3. Users/Actors of the NoSQL Database System

- **Developer:**
  A developer is an associate accountable for development of the desired application on NoSQL database system. He analyzes the requirements thoroughly and decides which aspects are critical for development and what are the dangers associated with the desired project. He uses the Remote Key/Value database to collaborate with other developers on the same project but different locations.
- **Team Leader:**
  This is the individual with power to deal with a task. This incorporates driving the arranging and the improvement of all undertaking deliverables. The undertaking administrator is in charge of dealing with the financial backing and timetable and all task administration systems (scope administration, issues administration, hazard administration, and so forth.). He is the person who doles out every one of the assignments and disperses the work to the colleagues to accomplish the objective of creating application using NoSQL database system. He will be benefited from this system as he will just access the database form his personal machine and access all the logs and keep a tab on ongoing activities of his project team.
- **Test executioner:**
  This person is responsible for running all the types of possible thorough tests on the system for testing the developed or under construction Remote NoSQL database. He is assigned test cases by the leader and is responsible to report the test results back to the system where team or analyst will check all the results of tests, also he makes log file of all the errors and exceptions handled by the system.
- **Project Team:**
  This group comprises of the full-time and low maintenance assets classified to deal with the venture's product. This incorporates the investigators, architects, software engineers, and so on.
  This is the group which works on creating application on Remote NoSQL database utilizing its framework.
- **Database Administrator:**
  He is an expert who models, plans and produces the required database used by the application development team. He also configures, monitors, maintains and secure the database in the project. He also decides the ownership of the data and sets different access permissions for users of database.
- **Teaching Assistants:**
  The TAs are kind of end users of the system who are going to run the command prompt command with command line arguments to setup the number of client and servers and run the batch files to access the user interface (UI), evaluate and grade the system based on the input (queries) they give to the UI and output displayed on the console display.
- **Instructor:**
  The instructor is also an end user who is going to check the system by running batch files, but he has more privileges than TAs. Instructor could check all the available source code and run some custom queries to check the robustness of the system. The instructor also has the power to change the grading of the system by overriding evaluation done by TAs.

4. Uses of Remote Key/Value NoSQL Database System

- **BigData:**
  The main use of non-sequential database is to manage the BigData. The growth of BigData is due to the growth of internet and social applications on mobile devices and inter machine communication. This is creating a huge amount of data to be managed and accessed remotely and that is where the Remote NoSQL database comes in the picture. NoSQL database provides very flexible data structure which can accommodate any type data the application needs. This data is mainly consists of both unstructured and semi-structured data, and to store them there is nothing better than NoSQL database as it provides seamless interaction between application and the database. Which ultimately results in smaller codes to be developed, debugged and maintained. Remote access to this database opens doors for many possibilities of services and applications with greater productivity and without location barriers.
- **BigUsers:**
  In recent years a trend of unprecedented data spikes is recorded on specific days, time and after some events. This has led to problems of prediction in number of concurrent users but to serve this rapidly growing or sinking set of users it is very essential to have a database who could serve this type of dynamic number of users. As elasticity in data is supported by the Remote NoSQL database it can easily entertain this type of BigUsers situation. This is used mainly by e-commerce websites, as they do face massive data spikes during holidays and sales.
- **Metadata Storage:**
  The Remote NoSQL database system have linear response time for all type of queries. Now metadata storage application like web based leaning websites have huge collection of different type of data under one single platform which is accessed by many users simultaneously from different locations. For example of different data types it could have pictures, text files, presentations, data sheets and many more types of contents. Now for practical use of these websites needs very low response time or latency in accessing the metadata and flexibility in storage; the Remote NoSQL database provides just the same and is perfect for it.
- **Mobile Applications:**
  Time of digital age has made one thing clear and that is ever increasing mobile platforms and large number of applications for the same. Now every mobile application is created for the user base and to fulfill ever-changing and increasing feature support demand the application developers are required to update the functionality of their applications very frequently. Now downtime caused in application because the updating process is not acceptable when the users are of millions of numbers. So the developers uses the Remote NoSQL database to quickly modify or extend the existing database without any major structural changes in the database, and hence the application has zero downtime even when it is being upgraded.
- **Internet of things:**
  Internet of things is a cluster containing many different types of devices connected to the internet which keeps on interacting between themselves. Today because of widespread of technology 20 - 30 billion of devices are simultaneously connected to internet and they have data collected from more than 70-80 billion sensors. Managing each inter-connection and syncing the data of each device with other connected device is very difficult task even conceptually, but Remote NoSQL database provides functionality of syncing, storing and continuously scaling which makes the billions of devices to work seamlessly with each other.

5. Architecture of the System

5.1 Package Diagram: Client Side

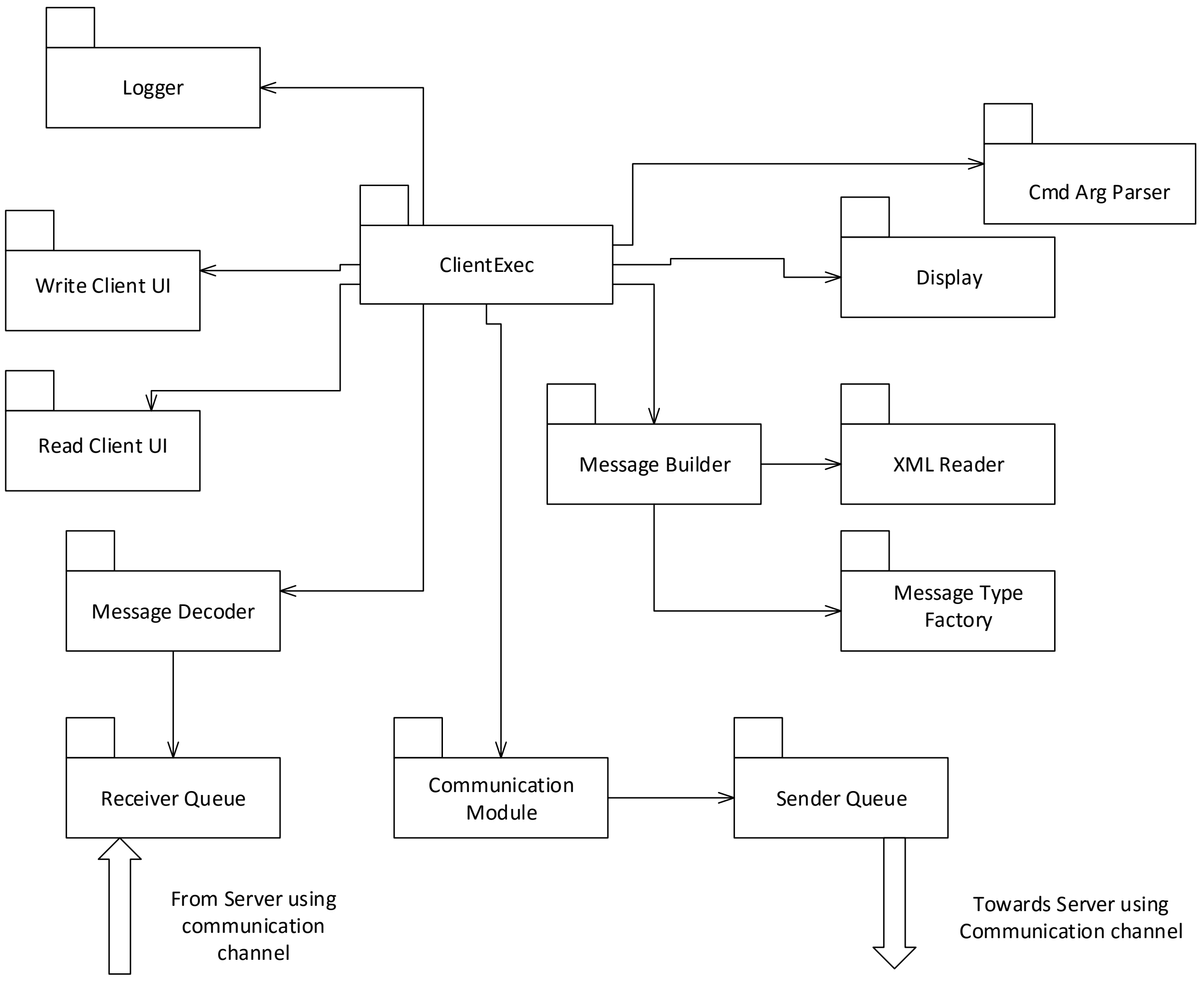


Fig 1: Package Diagram of Client side

This package diagram depicts the associations between the different types of packages which makes up the client side of the Remote Key/Value NoSQL database system.

The list of main packages and their description is traced below:

5.1.1 ClientExec:

ClientExec stands for client executive. This is the main module for Remote NoSQL database client side. It has fairly simple functionality to test the whole client side and it uses all the other packages in the client side to do so. It is connected to all the other packages through one and other package and it will have a main function which executes all the methods in client by loading a GUI and by giving usage functionality to the user.

5.1.2 Cmd Arg Parser:

Command-line argument parser is exactly what its name suggests. It takes command from the service initializer and initializes number of different clients i.e. write client and read client. For example, while initializing the server address and the number of read and write clients and to run the program we may write some command like this: "start .\Project4\bin\Debug\Project4.exe http://localhost:8080 10 10" where the localhost: 8080 defines the address of the server and "10 10" indicates numbers of read and write clients. We can also add "/log" at the end of this line to file every log on the console display.

5.1.3 Read Client UI:

Read client user interface provides interface for the user to make read queries on the NoSQL server database and read results of the queries. This will have association with both sender and receiver queues. As while submitting a query on the server it will use the sender queue and while the query result comes back to client it will use the receiver queue and fetch the data from it one by one.

5.1.4 Write Client UI:

Write client user interface provides interface for the user to make write queries on the NoSQL server database. This will also have association with both receiver and sender queues. It will use the sender queues to pass the data to the NoSQL database from the client and will use the receiver queue for the acknowledgement purposes.

5.1.5 Message Builder:

Message builder is an important package of whole package diagram, it will create the requested query messages for the read clients or makes appropriate data in XML format for intended data to be sent over the communication channel. It uses all the predefined XML formats from the XML reader. For the creating huge number of random data it will use message type factory. It will also add a client tag for its identification and return address purposes.

5.1.6 XML Reader:

XML reader will provide functionality to read the XML file at the start of the every read and write client initialization this will load all the pre-defined XMLs from memory which contains type of formats of messages to use or is being used while communicating with the server. The XML reader will then have all the type of XML message formats in its local memory which is then taken by message type factory during message creation.

5.1.7 Message Type Factory:

This factory would be given an unstructured message and a format of message to work with, it will process them and return a structured message to be converted to WCF. Also while generating large number of messages of random format which is required by writer client this message factory will be generating random messages.

5.1.8 Display:

Display is used to display logger event if specified by the user while giving command line arguments and to show the acknowledgements of the requests to the server. Also it shows the output of the high-resolution timer while the user gives command to measure the performance of the server.

5.1.9 Logger:

In every professional system a logger is included as a package. What it does is logs each and every activity of the system and keeps a track of all the important events in the system. It is very useful for developer while debugging and analyzing the system as he can run multiple tests on the system and a well-designed logger can record all the warnings and errors during the execution. Also this is very useful for post-selling maintenance of the system for obvious reasons. It also has a high-resolution timer which is initiated on command to measure performance and is stopped when all the requested entries are serviced (determined by looking at the acknowledgements the client gets from the server) by the server.

5.1.10 Message Decoder:

The messages from the server side will be in windows communication foundation format and the client requires these messages to be in XML format, hence this package will essentially work as a translator between received message and type of message supported by the client. It will take messages from the receiver blocking queue and provide it to required UI via ClientExec. Furthermore, this will also decide the type of message received, i.e. whether the received message is acknowledgement or data, if it is a data then what type of data it is. This will make easy for the ClientExec to redirect the data to correct UI.

5.1.11 Communication Module:

Communication module is a package which is mainly used to interact with communication channel for sending requests or data using sender queue. This package understands the requirements of the communication channel and formats the message to required format and adds it to sender queue. For our architecture this will convert the XML type message to WCF format.

5.1.12 Sender and Receiver Queues:

There are mainly two types of blocking queues one is sender queue and other is receiver queue. *A blocking queue is a queue that blocks when you try to dequeue from it and the queue is empty, or if you try to enqueue items to it and the queue is already full. A thread trying to dequeue from an empty queue is blocked until some other thread inserts an item into the queue*. [Ref: "tutorials.jenkov.com/java-concurrency/**blocking-queues**.html"]. We use blocking queue to avoid exceptions during run time.

All the messages dispatched from the client or received by the client are not sent/received in random fashion but in First in Fist out (FIFO) manner. This will prioritize messages by their timestamps. Receiver queue will have data ready to be fetched by decoder and the sender queue will have the data ready to be sent in WCF format in communication channel.

5.2 Package Diagram: Server Side:

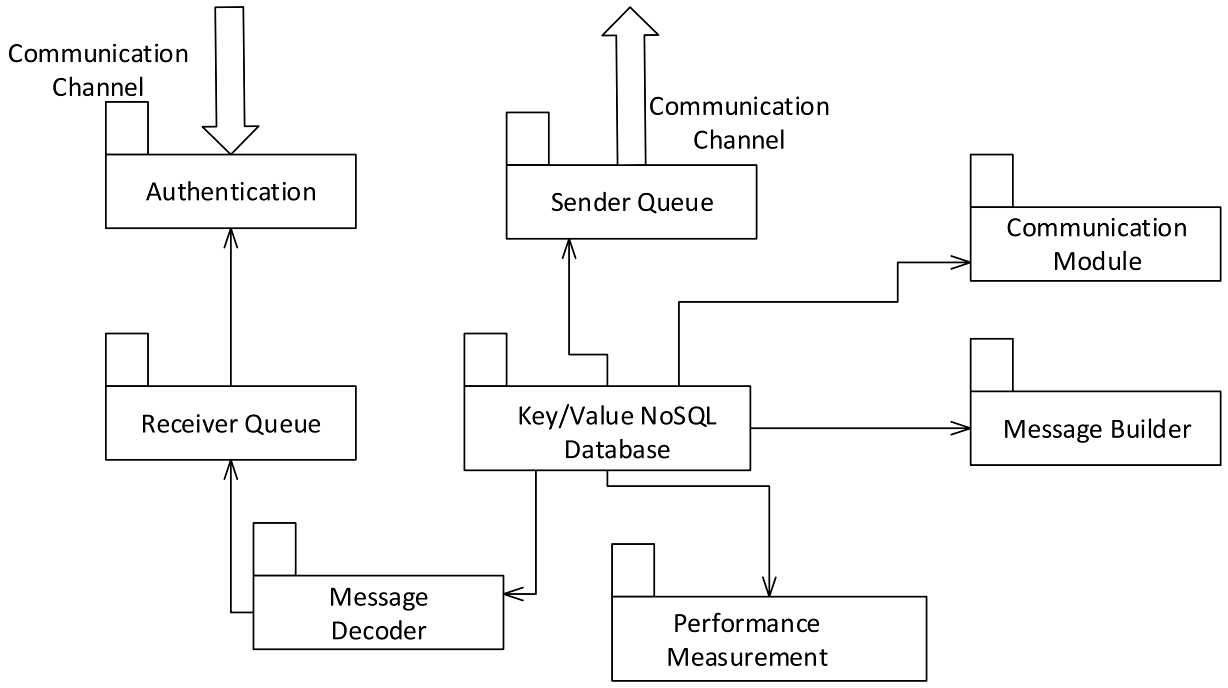


Fig 2: Package Diagram of Receiver side

This package diagram depicts the associations between the different types of packages which makes up the server side of the Remote Key/Value NoSQL database system.

The list of main packages and their description is traced below:

5.2.1 Authentication:

This is very important package in the receiver side as it checks the client ids before letting them added to the receiver queue. It will already have inbuilt subscribed user base ids stored and it will compare the unique id of the client with its inbuilt database. The requests from unknown clients would be discarded and matched data with correct ids would be added to receiver queue on request by the receiver queue.

5.2.2 Receiver Queue:

Receiver blocking queue will store all the authenticated requests FIFO fashion.

5.2.3 Message Decoder:

Message decoder will translate the WCF format messages to XML format and this would be fetched by the query manager of Key/Value NoSQL Database to process upon.

5.2.4 Key/Value NoSQL Database:

This package is the heart of the server side, it has a NoSQL database with all the necessary functionalities which we implemented in Project#1. All data will be stored in database using this package, also the message decoder will use inbuilt query manager of this package to serve the requests.

5.2.5 Performance Measurement:

This package will measure the performance of the server when it gets requests from the user. When user sends a performance measurement request along with random data to perform test, this package will start a high resolution timer when first entry of the random data is enqueued in the receiver queue of the server, and turns it off when the last data entry is serviced and enqueued back to sender queue to be sent. This will solely measure the performance of how the server performs without the consideration of latencies in the communication channel.

5.2.6 Communication Module:

Communication module in the server side is same as communication module as explained in the client side. The database package will use this module to convert its XML formatted responses to WCF format.

5.2.7 Message Builder:

This package will prepare the response of the required data in appropriate XML form using predefined in-memory templates and will have appropriate tags to define the type of message response. Moreover it will also use the request message's id tag to define the return address. This package will be called by the NoSQL database package, when it is done processing the query and wants to transmit the response.

5.2.8 Sender Queue:

Sender blocking queue will hold the results or acknowledgements created by the server to be dispatched towards the clients.

6. Chief Activities

Under this section we depict two prime activities with their respective activity diagrams.

6.1 General Activity Diagram Client Side:

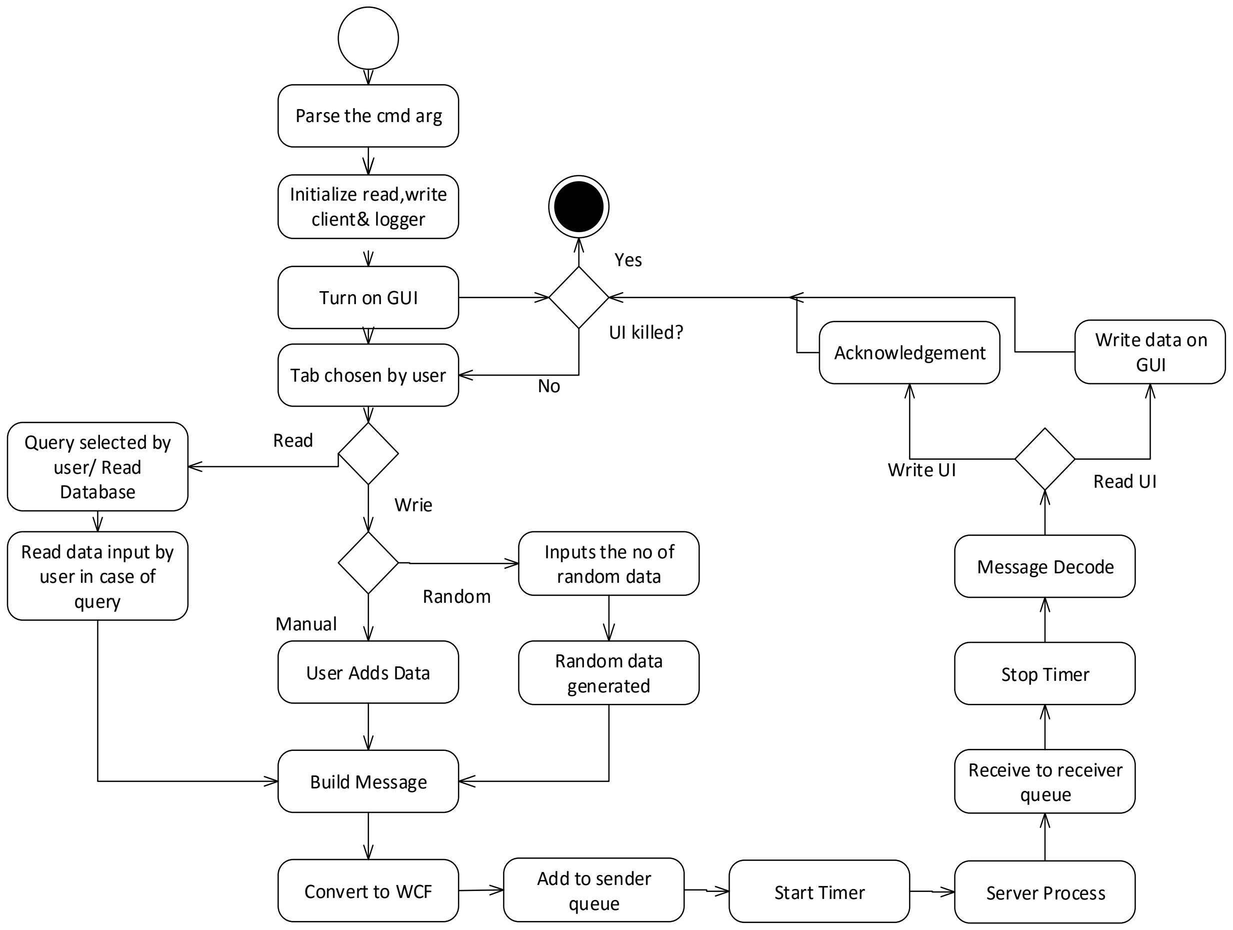


Fig 3: General Activity Diagram Client

The above figure shows the general activity flow of the client side and is explained below:

- The user will enter the command in command prompt to run this program with command line arguments, for example: “start .\Project4\bin\Debug\Project4.exe http://localhost:8080 10 10 /log”.
- According to this read and write clients will be initialized and server address is taken as first argument and the last tag defines that every activity will be logged on the console.
- After that the GUI will be turned on and user will be presented with tabbed window.
- User will choose one tab among given tabs.
- According to the chosen tab read client or write client is decided. For example, Add, edit, delete, persist database all those things will fall under write client and different types of queries and show database etc will fall under read client.
- For read operations, user will be presented with different queries to choose from and the data is read by the system.

- For write operations, there are mainly two types procedures one has random data and one has user defined data.
- Random data insertion is used for performance testing and it is described in detail in 6.3.
- Manual data is taken from the user and passed to next flow.
- For both read and write clients the message is built by message builder using XML reader and Type message factory.
- This generated message will carry tag for message type and client id for its authentication purposes.
- This created message is converted to WCF by using communication module and this converted data is added to the sender queue.
- After the enqueue the high resolution timer in logger is turned on.
- Now the WCF formatted messages will travel by communication channel towards server and the process of server is explained in section 6.2.
- After the request messages are processed by the server it will send the results back to the client and these messages will be enqueued to the client's receiver queue.
- The timer is stopped and the total time is logged in the logger.
- After that one by one messages are taken from the receiver queue and will be decoded by the message decoder.
- There could be two types of responses: one related to write client which is generally an acknowledgement or error, and second could be related to read client which could contain the data in form of XML.
- This both types of responses would be transferred to the respective UI to show output the user.
- Until the UI is explicitly killed by the user it will continue responding to user's input on the user interface.

6.2 General Activity Diagram Server Side:

The figure below shows the general activity flow of the client side and is explained as follow:

- The server is defined in the command line argument when user gives the address of the client.
- Once initialized the server will stay online until it is explicitly killed by the user.
- The received data is authenticated by using authentication module. This will check for client's id tag and will try to match that with its own client id database.
- If the client id doesn't match with the database it will be discarded.
- The authenticated data would be added to the server's receiver queue.
- The local timer of server would be turned on. The reason behind this is explained in the section 6.3.
- One by one data will be picked from this queue to the message decoder and the message would be converted to the XML format and the type of request would be understood by the server.
- The Key/Value NoSQL database's query manager will use this decoded message to process on it.
- The processed data is converted first to appropriate XML format using predefined templates and then converted to WCF format to be transmitted on the communication channel. This will contain a client id to decide towards which client the message should go.
- This converted message is then added to the server's sender queue and after all the responses are recorded to the queue the timer is turned off.

- After servicing one client the server would wait for other client requests.

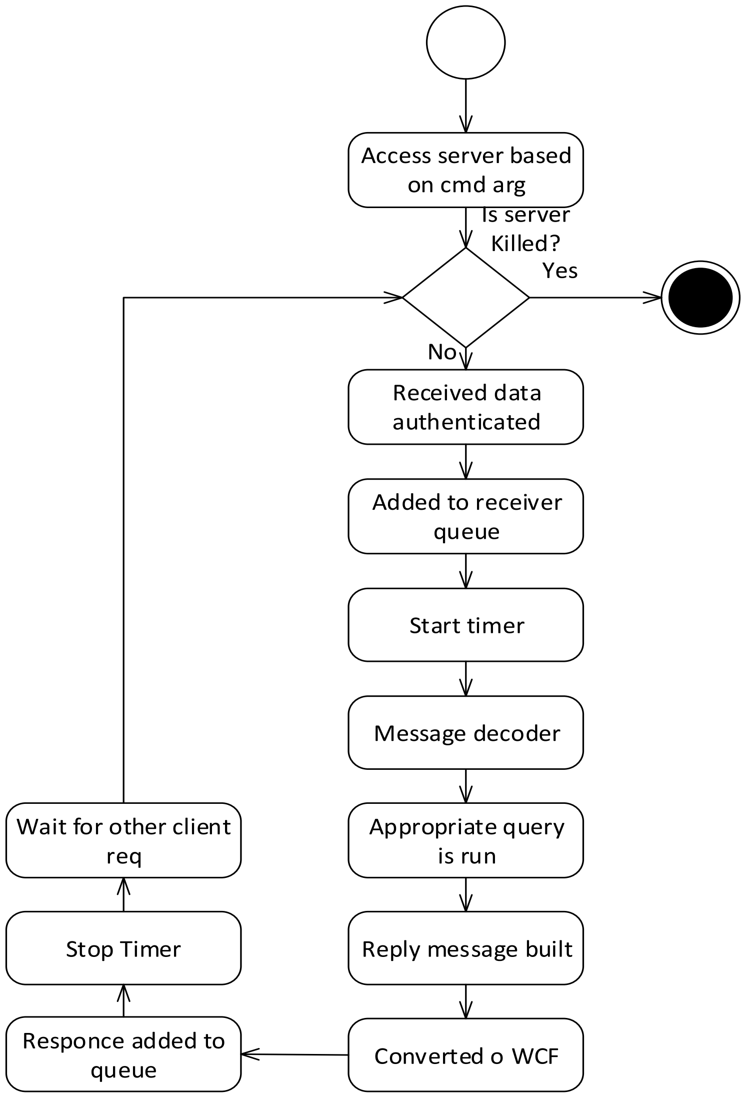


Fig 4: General Activity Diagram Server

6.3: Performance Measurement:

Performance measurement is an important activity performed in remote key/value NoSQL database. It is essential as it shows how the system will perform under different types of critical loads. There are mainly two types of performance measure here:

- Pure Server Performance Measurement
- Overall Performance Measurement

We define Pure Server Performance Measurement by only considering the time taken by the server to process all the requests in its receiver queue and add them back to its own sender queue. We can observe that this measurement will not consider any time latencies taken by communication channel.

In contrast to this Overall Performance Measurement will count the time from when the data is added to the sender's queue to the time when the client receives all the responses back to its receiver queue. This will take in account all the communication channel related latencies and server side processing overhead. By processor overhead at server side it is meant that the server doesn't have only one client, it could have multiple, and at a given time when one client enqueues requests in server it is not necessary that the server would be free; it will take some time to process the older requests and then service the newer requests an d this will add to overall waiting time.

The latter performance measurement is more critical for a system as the first one only measures the raw processing power of the system, while the latter measure overall system load distribution and scaling which is critical for the NoSQL database.

For this system's performance measurement we have created different templates to measure different types of performances for example there are templates for enqueues, certain set of queries, persistence etc. Now we will ask user to enter number of inputs he wants to test for different types of performance tests and will generate that number of random data for the test case.

The generalized diagram of performance measurement is as follows:

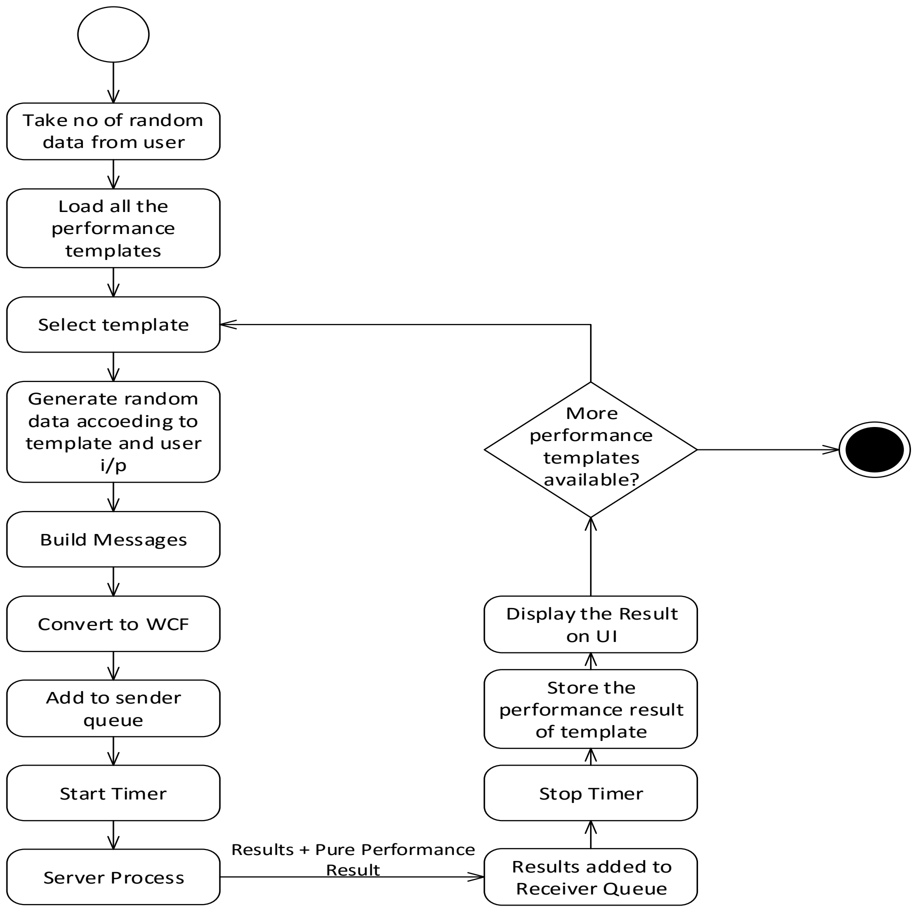

Take no of random data from user
Load all the performance templates
Select template
Generate random data accoeding to template and user i/p
Build Messages
Convert to WCF
Add to sender queue
Start Timer
Server Process
Results + Pure Performance Result
Results added to Receiver Queue
Stop Timer
Store the performance result of template
Display the Result on UI
More performance templates available?

Fig 5: Performance Measurement

The logical steps of above activity diagram can be described in following steps:

- At the beginning of the performance testing, no of random data input is taken from the user.
- All the performance test templates are loaded in the memory.
- One of the templates is taken into consideration and according to that and user input data, random messages are created with the help of Message Type Factory using Message builder.
- These XML structured messages are then converted into WCF format and added to the sender queue.
- At the same time when the data is added to the sender queue of the client a high resolution timer is turned on. Here we are using high resolution timer as it has resolution of 1ms and that type of granularity is essential to measure the performance precisely.
- These messages are passed to server and the server will respond with all the request results and its Pure Server Performance measurement.
- All these results are added to the client's receiver queue and when the last result is added to the queue, the timer is stopped and the result is recorded.
- Now this recorded time and the received time of pure server performance measurement are forwarded to the user interface.
- After this this whole process is repeated for all the available performance templates.

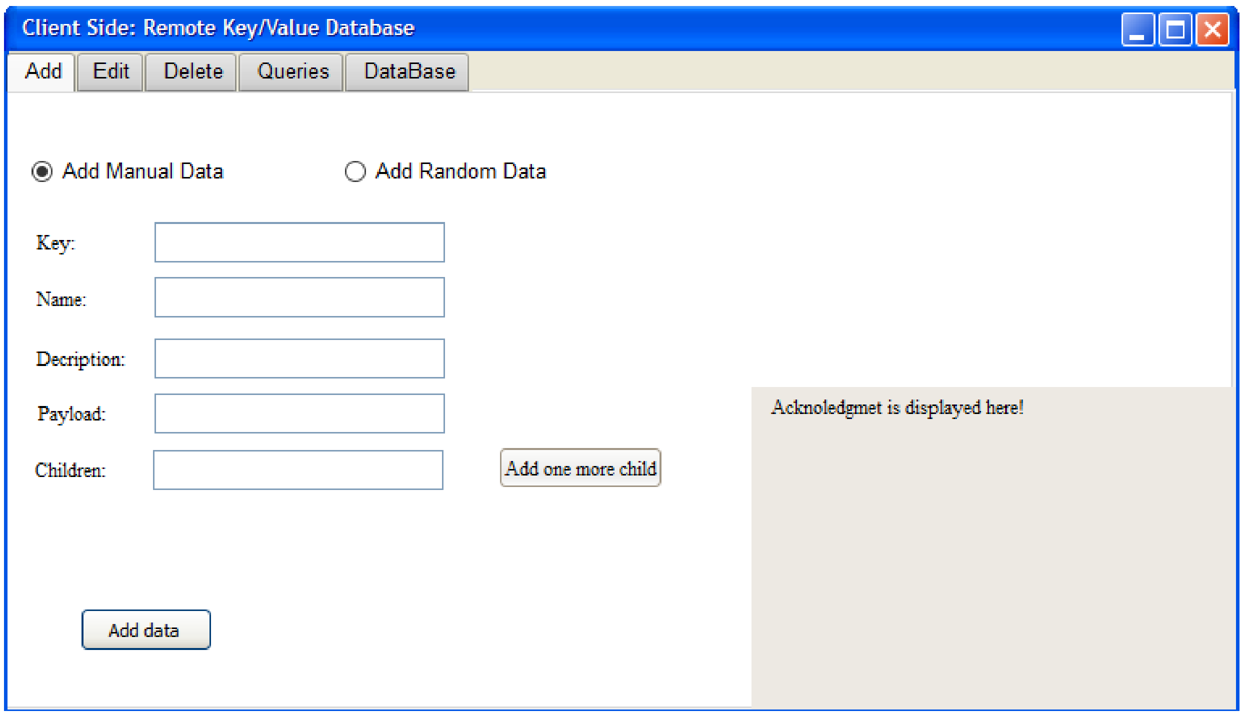
7. Views

Remote Key/Value database provides graphical user interface for the users to interact with the system. The client side of the system, gives the user ability to manipulate the database remotely and provides functionalities as listed below:

- Add Data
- Edit Data
- Delete Data
- Perform Queries on the NoSQL database
- Database Operations such as Retrieval and Persistence

Below is the description of all the different views with their screenshots and description. As it can be clearly seen the GUI has different tabs for the user to perform different functionalities.

7.1 Add Data:



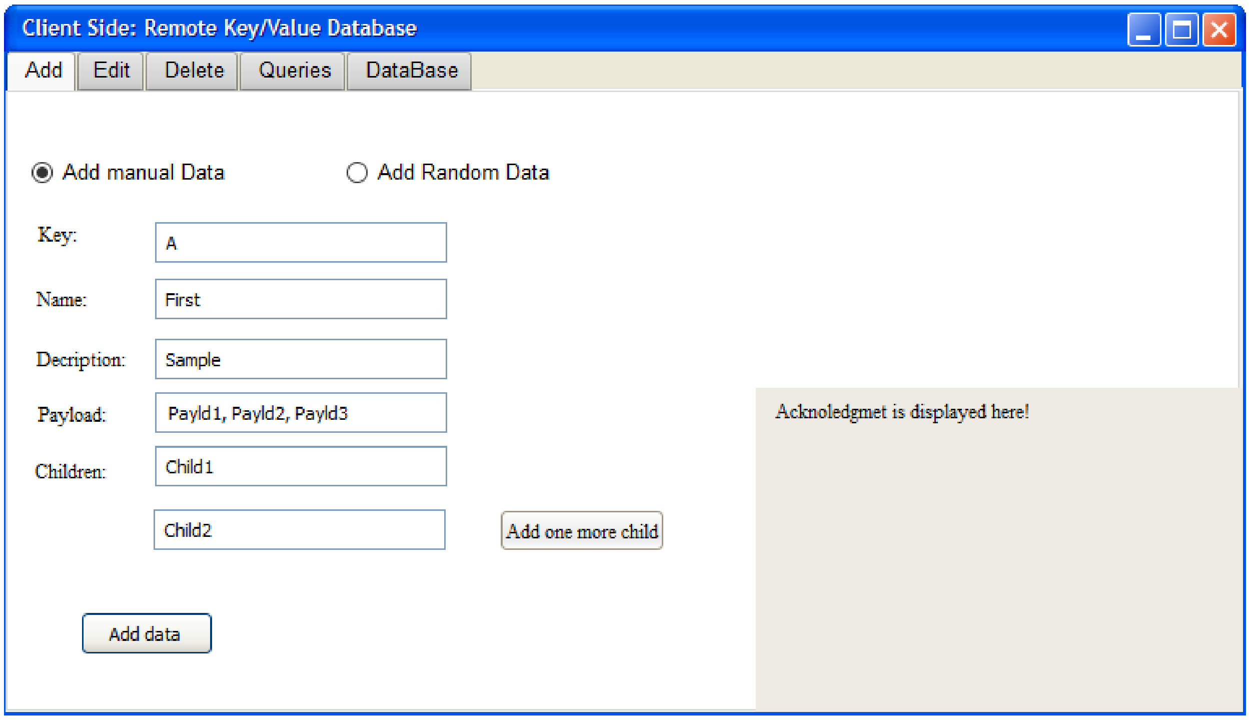
Acknoledgmet is displayed here!

Fig 6: Add Data

In this window the user can add data to the remote database easily without doing any actual formatting of the data. The user have two options:

- Add Manual Data
- Add Random Data

In option of adding *manual data* the user will be able to add data manually of his desired choice. There is also an option to add more than one child to the database. This is shown in figure below:



[● Add manual Data] ○ Add Random Data

Key: A

Name: First

Decription: Sample

Payload: Payld1, Payld2, Payld3

Children: Child1

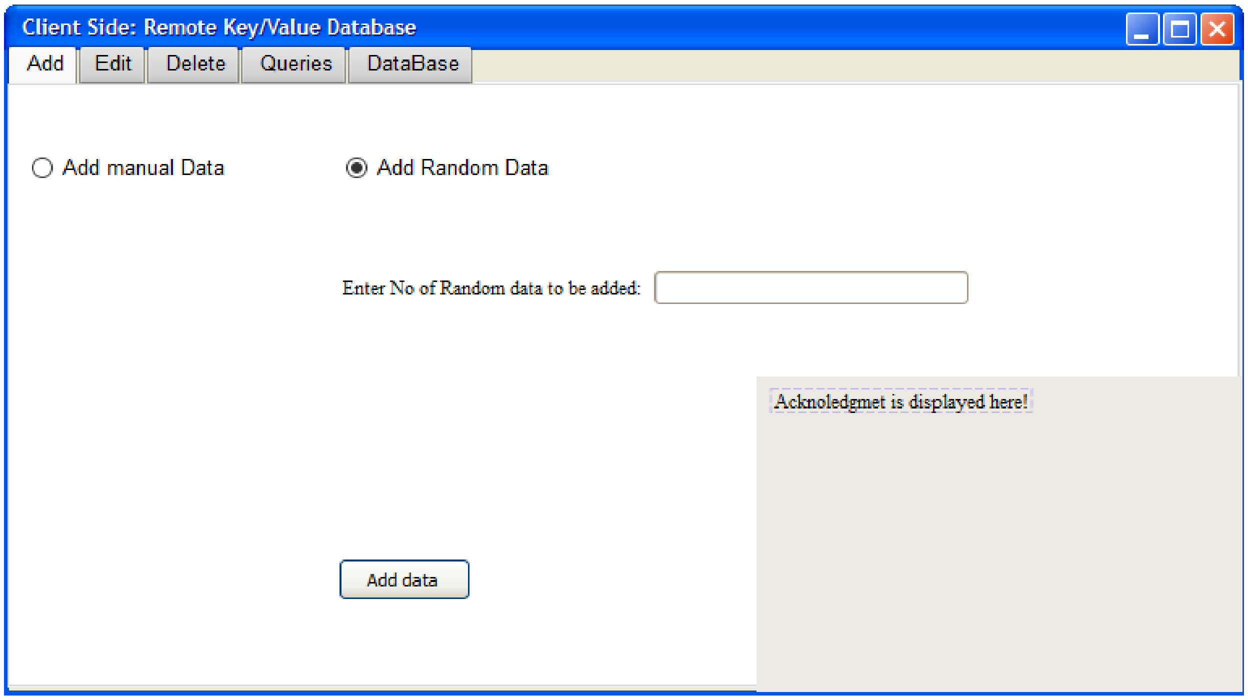
Child2

Acknoledgmet is displayed here!

Fig 7: Add Data (More than one child)

An ambiguity could arise when the user is adding the key which already exists in the database, for this situation we have designed the system to over-write the given key data in the database.

In the option of adding *random data* the user will be able to enter desired no of data in random fashion.



○ Add manual Data [● Add Random Data]

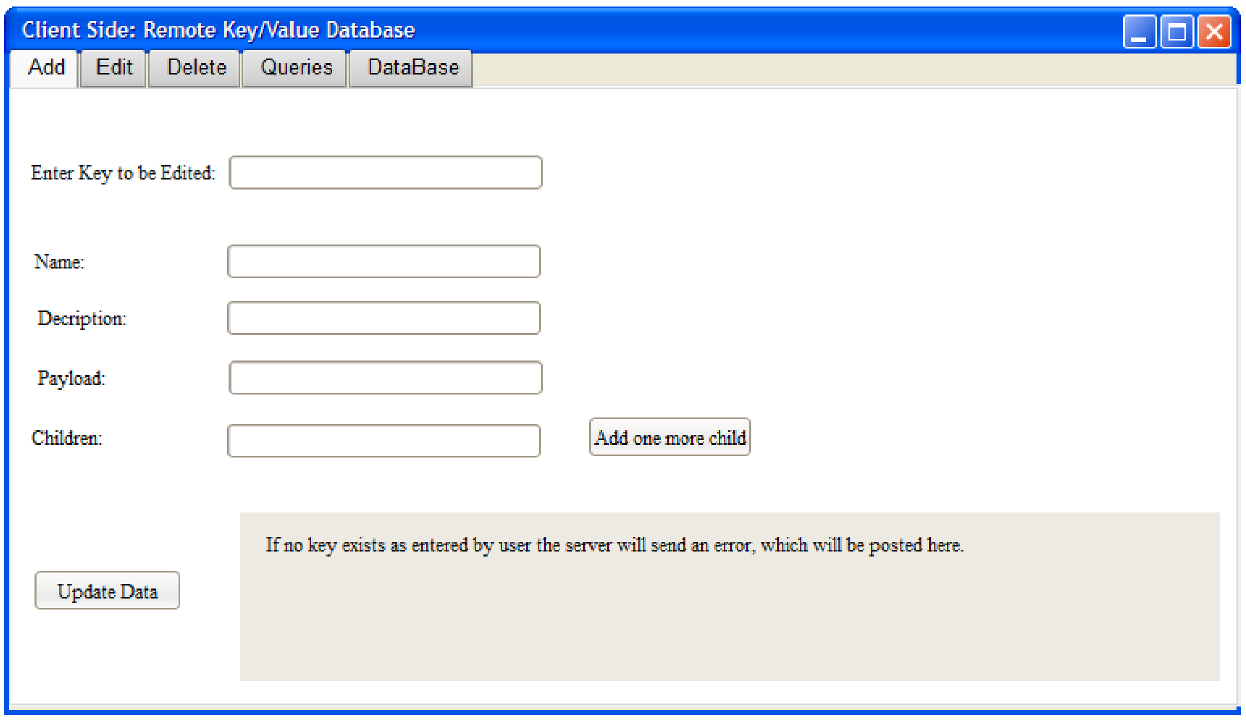
Enter No of Random data to be added:

Acknoledgmet is displayed here!

Fig 8: Add Random Data

7.2 Edit Data:

Using this option, the user will be able to edit an already existing data in the database. For editing purpose the pre-requisite for the user is that he must be knowing the key of the data he wants to edit. If there is no data in the key/value database for that particular key, the database will send an error in the acknowledgment to the client to show it on the UI.

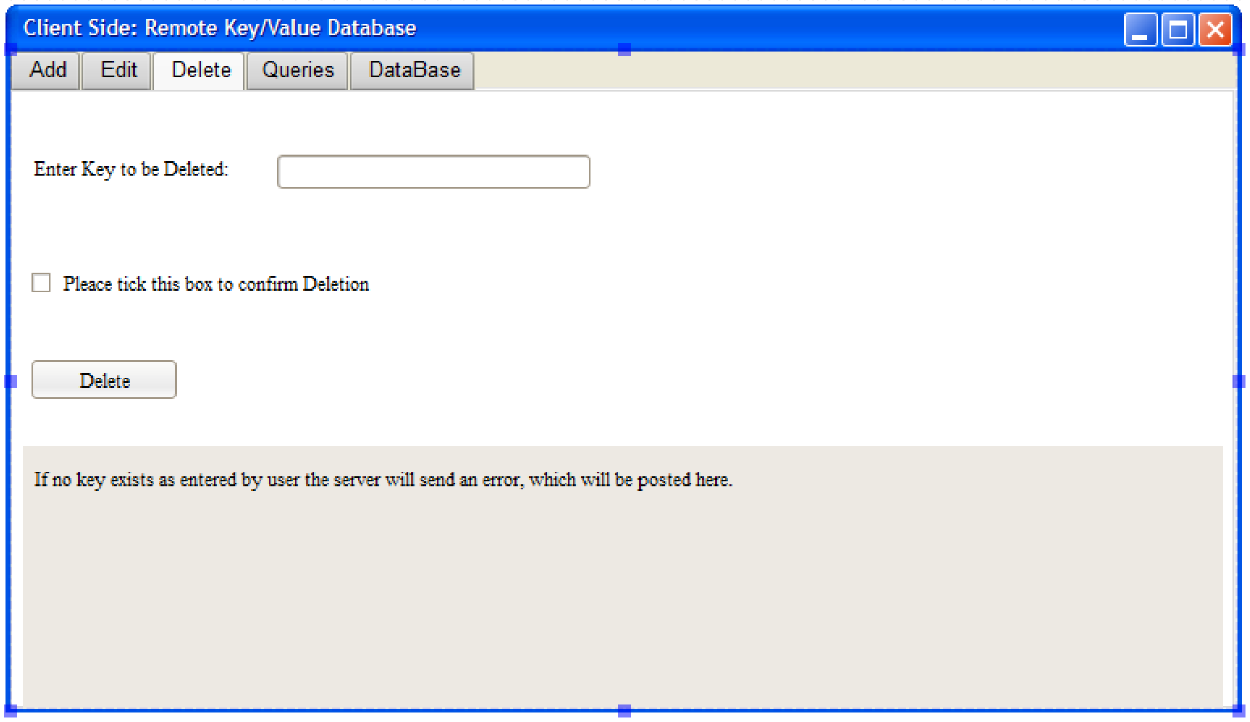


If no key exists as entered by user the server will send an error, which will be posted here.

Fig 9: Edit Data

7.3 Delete Data:

To delete some data from the database the user will insert specific key to delete the whole entry from the database. If user enters the wrong key to be deleted the server will notify the user with an error message. For extra safety there is also one checkbox to be ticked by the user before he deletes an entry by pressing the “Delete” button.



Enter Key to be Deleted:

☐ Pleace tick this box to confirm Deletion

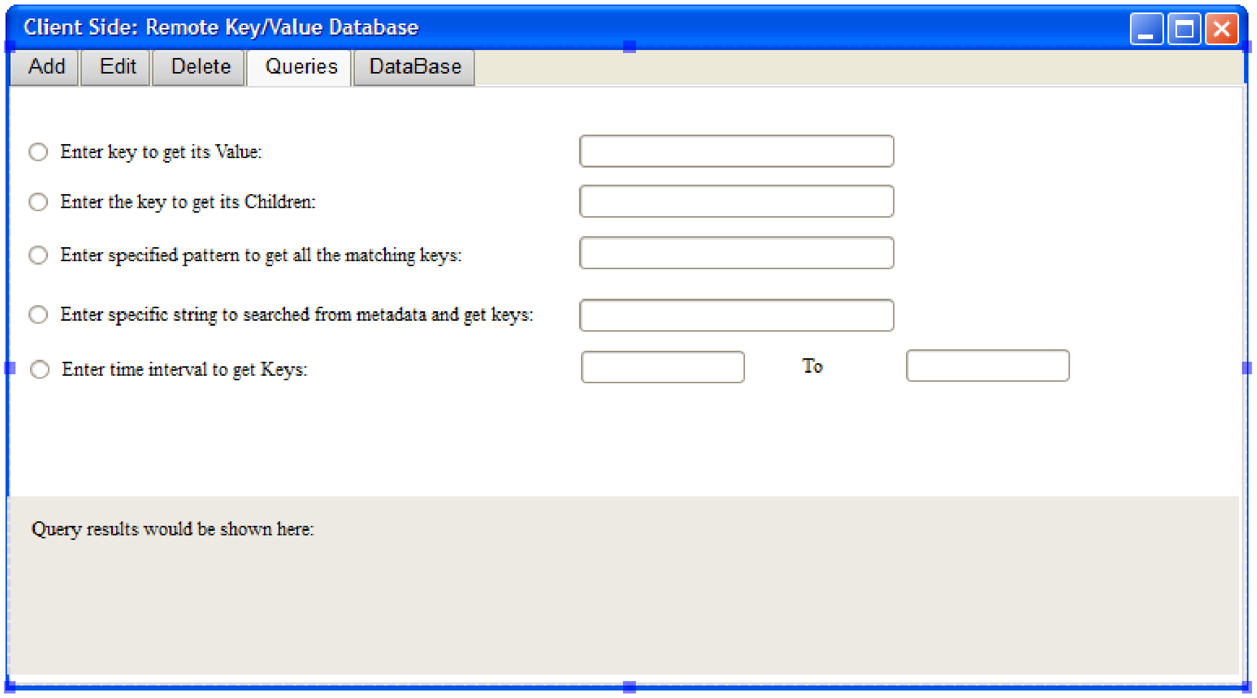
Delete

If no key exists as entered by user the server will send an error, which will be posted here.

Fig 10: Delete Data

7.4 <u>Queries:</u>

In this window the user can run different types of queries on the database. User could search for specific key data, get children of specific key, search for specific pattern or string from the metadata and get keys for matching pattern and find all the data between specific timestamps.



○ Enter key to get its Value:

○ Enter the key to get its Children:

○ Enter specified pattern to get all the matching keys:

○ Enter specific string to searched from metadata and get keys:

○ Enter time interval to get Keys: To

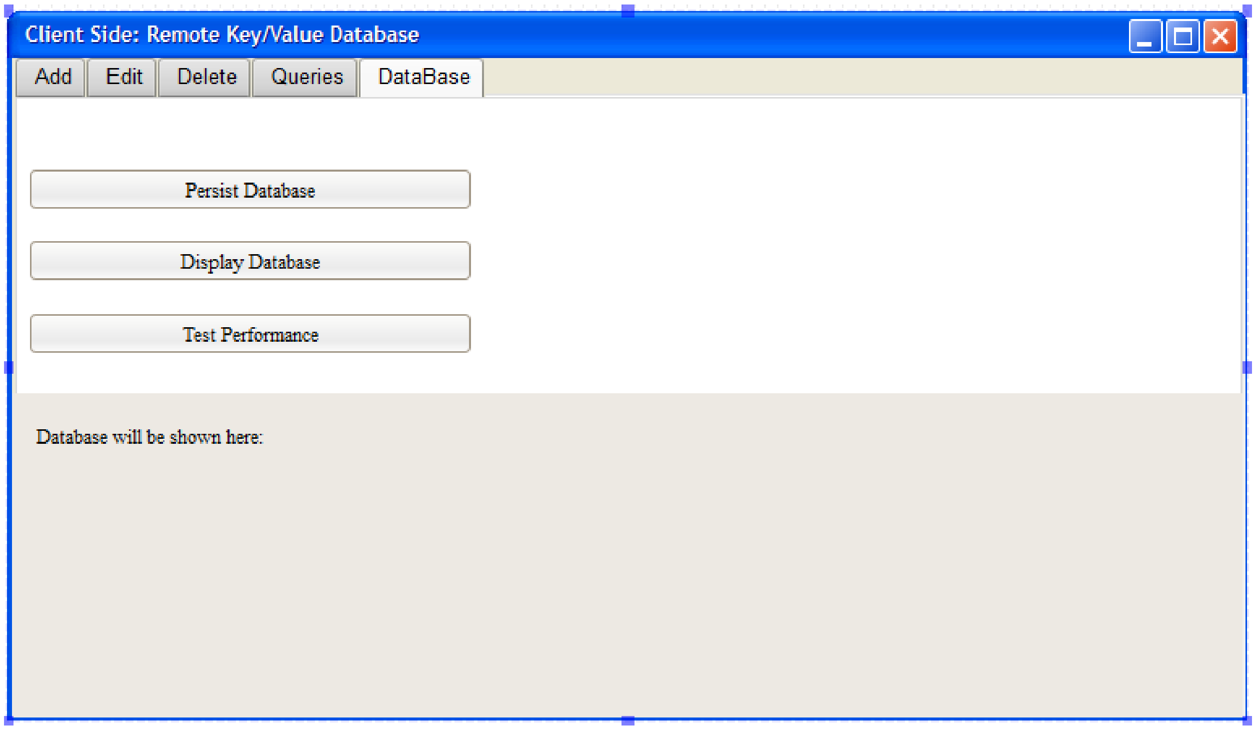
Query results would be shown here:

Fig 11: Perform queries on NoSQL database

7.5 **Database Operations:**

User will be able to perform some database operations which are essential to test the requirements of the problem. This view shows the database tab by using which the user will be able to persist the database and also retrieve the database and display it on the UI.

The test performance button will show the performance of the server for predefined test cases of different number of input data in the blocking queue of the server. This will display performance measured for all the test templates.



Database will be shown here:

Fig 12: Database Operations

8. Critical Issues

8.1 Demonstrating Requirements:

Issue:

For demonstrating the requirements in the given requirement document, an API needs to be developed and should be executed on the system to check if all the requirements are met correctly. This is a critical issue because if all the requirements are implemented correctly, but if some of them are not executed in an API then the designed system will lose points on being complete.

Solution:

Deliberately look at the requirements and decide the test cases for designing the API.

Impact on design:

By creating a perfect API and an application with GUI we display the requirement number and change on the database after each user interaction.

8.2 Eventual Consistency:

Issue:

In the non-sequel database the same data is stored across multiple servers because of the replication property of this database. This could lead to problems of updating database simultaneously across all the database at the same time, for example if one database is updated on the one server it might take some time before all the replicas of this server are updated. And if immediately after this a query is run on one of the replicated server the updated data is not guaranteed. This phenomenon is known as “Eventual Consistency”.

Solution:

We could build a system which has option to choose from different levels of consistency and application developer could select the level of consistency needed by his own needs and provide flexibility. Writing programming solution for mentioned solution needs to be developed in future expansion of the system.

Design Impact:

Implementation of above mentioned solution will make the system more developer friendly and would attract large user base for the system.

8.3 Authentication:

Issue:

When the remote servers are included in the main system a critical issue of authenticity rises. If no extra care is taken about the access points of data then we are making the database vulnerable to security threats. Hence the system needs some kind of well-structured authentication system.

Solution:

In this system we are using XML to create messages and then sending the message in WCF format in the communication channel. We can keep a tag named “<id>…</id>” in the message which could be easily extracted by the authentication package at server side. The authentication package will have access to database having all the subscribed client ids.

When a request comes to the client, the authentication package will reads its id and will compare it with the all the ids in the database. In case it cannot extract the id tag from the message it will discard the message by declaring that request as malicious.

When an id doesn't match with other ids in the database it will just discard the data request, as requests from unknown sources could harm the database, by flooding, modifying or deleting essential data from the database.

The authentication will only let the known client access and manipulate the database and keeps it safe from all harms.

To make this prototype full proof we could use combination of password with the id and force some of the policies described below:

- Passwords should be refrained to use dictionary words, name of the user or birthdates.
- Passwords should expire after certain amount of time to be changed to new password.
- Reuse of password should be forbidden.
- After a certain number of login failures, the account should be locked until further inspection is done on account activity.
- User access from unknown devices and geolocation should not be allowed.

Design Impact:

Carrying out the authentication effectuation in the system brings a sense of confidence among the users and makes the database secure from unsafe accesses which keeps data integrity intact.

8.4 Data Privacy:

Issue:

By creating this system we have created a very powerful system which can store different data types from different users at a single place. Many subscribed clients could use this system but there is no data privacy implemented in the system. That is if a client doesn't own a particular set of data, still he will be able to access and modify it easily by just querying for their keys. More dangerously, the client has very powerful functionality to get the whole database, by using display database, which essentially kills data privacy 100%.

Solution:

We can implement some type of data portioning in the database. Which can be done by keeping different client's data in different XML files and giving authentication to only selected client on that stored file. But this can harm the performance in a way that the more the user the more files created, and while replying the requests the database will have to internally find the assigned file from huge number of files. This system could be implemented efficiently using layered structure in which the xml files will have an address directory from which they could be found quickly and processed upon.

Design Impact:

If we can implement data privacy in our system it will be favored by most of the users as privacy is indeed a primary requirement in any data storage system, and this will make the whole prototype system more mature in terms of real world standards.

8.5 Performance Testing:

Issue:

Performance testing is a critical aspect of any system as it will decide the real world performance of the system under huge loads. We have created a system where there is one server and many clients connected to same server. It is possible that all the clients can request for some data at the same time, at this time the server could be exhausted and could break down.

Solution:

To solve this critical issue we have implemented a blocking queues in both server and client side. Blocking queue won't accept any new enqueues until there is no space in the queue this will keep the old requests safe and will keep the new requests in the waiting. The client will service each request one after another and new requests would be added to the empty places in the queues. We have dedicated whole section 6.3 on performance testing which talks in detail about performance tests done on the system.

Design Impact:

By implementing blocking queues in place of simple queues we are making the system prone to breaking down due to overload.

8.6 Locking:

Issue:

When more than one clients are trying to manipulate the same data and get the original data then they might get wrong data as a result. Hence we should implement some system to lock the access of a data while it is being used by one client.

Solution:

We can use some bit like flag with all the data bytes, when a data byte is being accessed for any reason the flag bit would be set, and no other request would be able to access that byte.

Design Impact:

If this flag bit could be implemented then it will remove the issue of locking toughly from the system. But it will use more memory than before as each data byte would use a flag bit to keep a track of the access.

8.7 Communication Errors:

Issue:

We rely on some kind of communication channel to transmit messages between client and server. There could be some inherent issues which could make the message erroneous.

Solution:

We can use checksum with the messages to check it at the receiver side. If the message has errors we can have a system which can request the data again from the client or server.

Design Impact:

By implementing checksum we are making sure that our system has no errors as far as communication channel is concerned.

9. Conclusion

To summarize the whole operational concept document we have documented a Remote Key/Value NoSQL system which supports efficient query handling for both complex and simple type of query requests from more than one remote clients. The performance testing is also possible by the user for different pre-defined test cases. Also the uses and users of the system are explained.

We have also mentioned some critical issues like authentication, performance testing, data privacy and locking for the system and mentioned the solution for the same. If this solutions could be implemented thoroughly while implementing the system, it would create a close to perfect system, if not these issues could create some serious trouble while enforcing final system.

10. References

1) tutorials.jenkov.com/java-concurrency/**blocking-queues**.html

2) https://piazza.com/class/idvr3t7iowm3zh

3) http://ecs.syr.edu/faculty/fawcett/handouts/webpages/BlogNoSql.htm

4) OCD of Key/Value NoSQL Database by Achal Velani (self-citation)