CSE-681 Software Modelling and Analysis

Instructor- Dr. Jim Fawcett

Project #1

Key/Value Database-OCD

Version- 1.0

PavanKumar Bodineni (SUID#433383347)

9-16-2015

Contents

Figures

1. Executive Summary

Traditional relational databases has its own limitation when it comes to performance and storing of huge data .When requirements demand extremely dynamic, high performance with real-time results and high availability data stores, we have to use NoSQL databases in this scenario. NoSQL databases do not have a common way to query the data (i.e. similar to SQL of relational databases) and each solution provides its own query system. The NoSQL taxonomy supports key-value stores, document store, Big Table, and graph databases. In this project we are storing data in key-value stores. The key value type basically, uses a hash table in which there exists a unique key and a pointer to a particular item of data.

The purpose of the project is to implement a generic key/value in-memory database by storing key, value and metadata associated with it. The project should provide capability to:

- Create items described by metadata and holding an instance of some generic type.
- Create and Manage a Key/Value database with capability to store and delete Key/Value pairs.
- Edit Values, text metadata.
- Persist database contents to an XML file.
- Augment database contents from an XML file with the same format as persisted, above.
- Support a variety of queries, both simple and compound.
- Support demonstration of all functional requirements through a series of discrete tests with display to the console.

The intended users of the project are Developer, Teaching Assistant and Instructor. The developers will use this document as guidance for design and implementation in later phases. The TA's and instructor shall use this to check whether all the requirements are met and all the critical issues related to project are addressed.

Below are some of the critical issues associated with the project, and solutions are discussed in detail in further sections

- Constructing new immutable database from query results.
- Maintaining eventual consistency when queried on database.
- Implementing compound queries on top of Key/Value database.
- Implementing sharding will be great challenge in the project. On what basis we shall shard? How to store the shard results?
- Restoring and augmenting database from an existing XML file. What if existing XML file is corrupted? How to query data from shards?
- Authentication and authorization issue should be handled when the clients connects remotely to server to access NoSQL database. This concerns will be addressed in project 2.
- Maintaining data in shards reliable and fault tolerant.

This document further provides discussions on interactions between modules of the application, flow of activities to achieve tasks with support of diagrams, use cases of this application, Solutions to the critical issues associated with the application.

2. Introduction

Application needs have been changing dramatically in large part because of growing numbers of users that applications must support and growth in the volume and variety of data that developers must work with. As a result, the use of NoSQL technology is increasing among internet companies and enterprises because it offers data management capabilities that meet the needs of modern applications. It's increasingly considered a viable alternative to relational databases, especially as more organizations recognize that operating at scale is more effectively achieved running on clusters of standard, commodity servers, and a schema-less data model is often a better approach for handling the variety and type of data most often captured and processed today. It also reduces the overhead of impedance mismatch between the relational data structures and the in-memory data structures of the application.Using NoSQL databases allows flexibility to develop without having to convert in-memory structures to relational structures.

NoSQL database can be implemented in four types: Key-value stores, Document databases, Graph stores, Wide-column stores. As we are implementing Key/Value data store in this project, we will be talking much about it. Every single item in the key/value database is stored as an attribute name (or "key"), together with its value. Examples of key-value stores are Riak and Voldemort.

2.1. Objective and Key idea

The objective of the project is to use the real benefits of NoSQL database to store very large data. NoSQL systems originated to provide high throughput, fault-tolerant horizontally scalable simple data storage and retrieval with a bare minimum of additional functionality. The key idea is to implement Key/Value data store which supports addition and deletion of key/value pairs. When data is very large, it should be partitioned into smaller, faster, more easily managed parts called data shards. NoSQL systems maximize throughput by limiting how the sharded data is managed and accessed.

2.2. Obligations

The primary obligations of Key/Value data store is to provide

- Provide mechanism to handle very large collections of data.
- High throughput with data from streams.
- Tree data model to store parent, child relationships. Each key shall store all the child keys associated with it.
- Support heterogeneous collections of data.
- Horizontal scalability which will support high availability of database.

2.3. Organizing principles

- The key/value data store uses C# data dictionary to store key/value relationships which internally uses hash map data structure.
- The main principle is to develop independent packages for specific functionality which will communicate only through Executive package.
- Shall use factory methods to construct a Value with supplied parameters and construct immutable database from query results.

3. Use Cases

3.1. Developer

Essentially each student developer is responsible for understanding and demonstrating each of the requirements, shall also provide solutions to all the critical issues related to the project. The developers use this application to extend NoSQL database functionalities to connect with clients remotely in project 4. Multiple clients connecting to database remotely can send data as input or can query results from database. Developers shall also use this application to build data management service in a large Software Development Collaboration Environment in project 5.

3.2. Teaching Assistant/Instructor

The TA will use the application to check whether all the requirements for project are successfully implemented or not. They will also verify whether solutions for all the critical issues are best addressed or not. TA's will also verify the feasibility of the application for future projects.

3.3. Other Applications

This application can be used by the systems which demands to process huge volumes of data and requires to provide results in faster time.

3.3.1. Extending to Project- #4

In project 4 we are going to implement Remote Key/Value database where multiple clients can connect to application concurrently and access the database. Using this application we can extend its capabilities by providing user interface by using WCF on top of Key/Value database.

3.3.2. Logging

Event log processing and analysis play a key role in applications ranging from security management, IT trouble shooting, to user behavior analysis. At the same time, as logs are found to be a valuable information source, log analysis tasks have become more sophisticated demanding both interactive exploratory query processing and batch computation. This application can be extended to support log data stores to support log processing and analysis, exploiting the scalability, reliability, and efficiency commonly found in Key-Value store systems.

3.3.3. Visualize threads

When an application is running in complex multithreading environment, it is difficult to identify behavior of the threads. In this scenario Key/Value database is much useful to visualize the threads behavior.

3.3.4. Store program data at runtime

Extending the application by providing an API to Key/Value database will allow applications which demands to store large amounts of data dynamically at runtime.

4. Application Activities

The key task of the application is to construct Key/value data store. When data is huge, relying on in-memory storage is not the solution, then we have to shard the data into smaller and efficient chunks and store in external files. Application should persist database contents by writing data to files. Database should also support ability to edit/delete or add new values to shards.

The overview of flow of activities for this project is expressed in below activity diagram. Sharding and Querying is expressed in detail in other activity diagrams.

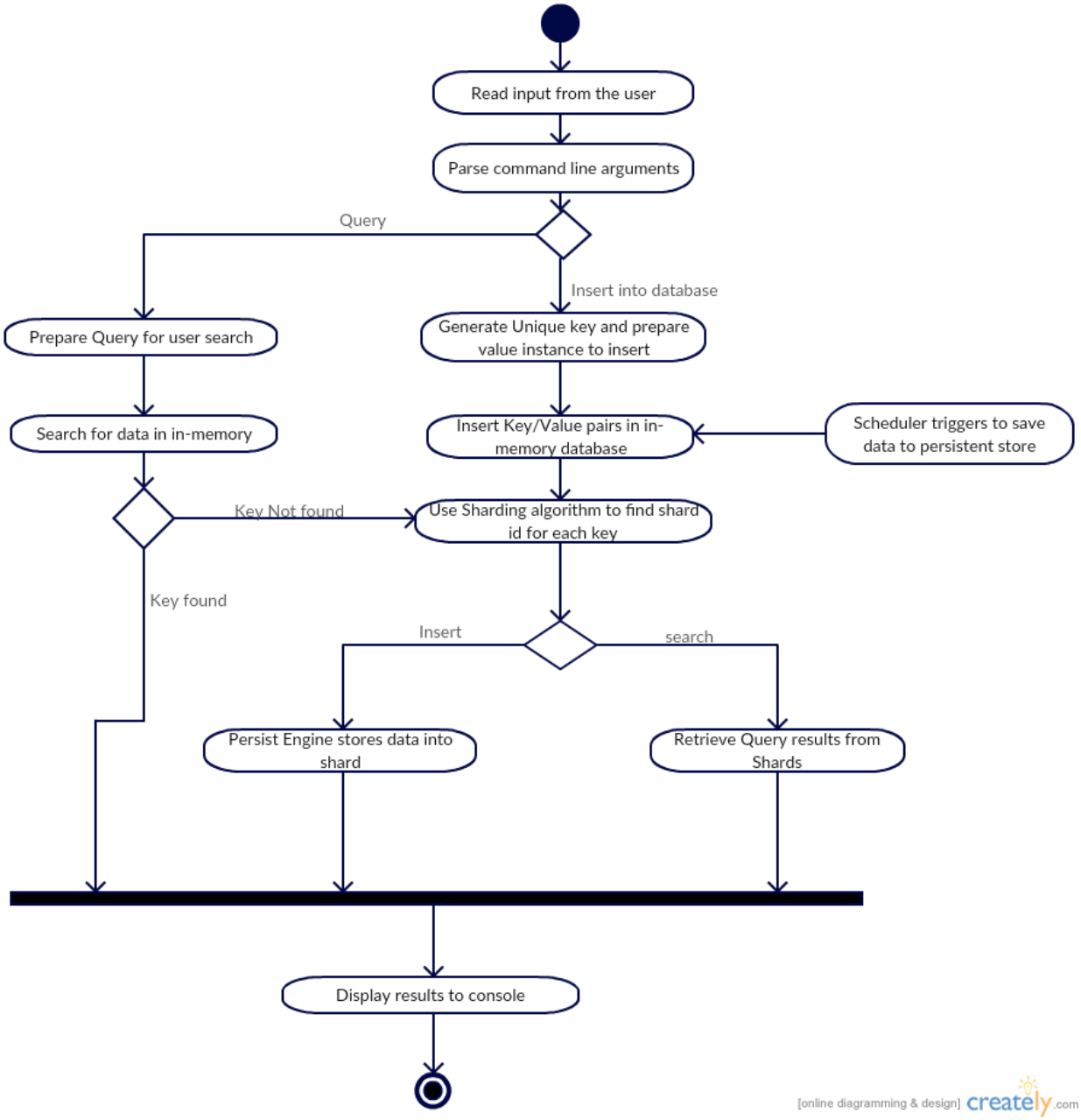

Figure 1: Activity diagram for Key/Value database

The main activities involved in the application are described below in a sequence

4.1. Read input from user and construct database

For constructing database initially, the user feeds XML file as input to the application which consists of value and metadata associated with it. Application creates instances of Key/Value pairs using a simple factory that may generate a unique key and construct a Value with supplied parameters. A hash value will be generated for each key, when keys are fed into hash function. Key/Value database is constructed using C# data dictionary where key/value pair will be stored in hash map using hash value associated with each key. Each key will be linked to an address which points to Value object. Each database Value has structured meta-data and an Instance of the generic type. We will choose to create a C# class to represent Values that might look something like this:

```
public class Value<Key, Data>
{
  private string name;
  private string description;
  private List<Key> children;
  private Data payload;
}
```

4.2. Edit/add/delete values

User should be provided ability to edit values including the addition and/or deletion of relationships, editing text metadata, and replacing an existing value's instance with a new instance. For adding new value, new key is generated and key/value pair will be stored in hash map in in-memory or shard. First the value object is searched in in-memory store, if not present, will search on shards and loads particular key/value pair to in-memory and then edit is performed.

4.3. Persist database contents to an XML file

Scheduler persist database contents to an XML file. It accepts a positive time interval or number of writes after which the database contents are to be persisted. If scheduler has to persist data after number of writes then it will maintain a check on database to find the number of writes and then calls persist engine to store data from in-memory to XML files. Loading into XML files is a continuous process till the inut condition becomes false.

Sharding distributes different data across multiple servers, so each server acts as source for subset of data. A NoSQL model may use a hash table to store key/value pairs incurring essentially constant time lookup and retrieval of its data, e.g., time independent of the size of the data. However, when the size of the managed data requires sharding, the constant time lookup and retrieval may be compromised by processing necessary to locate shards that contain the data we need to retrieve. To overcome this, we will manage multiple shards in memory using a Least Recently Used mapping strategy, much like a virtual memory system to store the most recently used shards in main memory. There are numerous methods for deciding how to shard your data, and it’s important to understand your transaction rates, table volumes, key distribution, and other characteristics of your application.

How I implemented sharding concept in this project is expressed in below activity diagram. The critical issues associated with Sharding and solutions for them are explained in section 6.6.

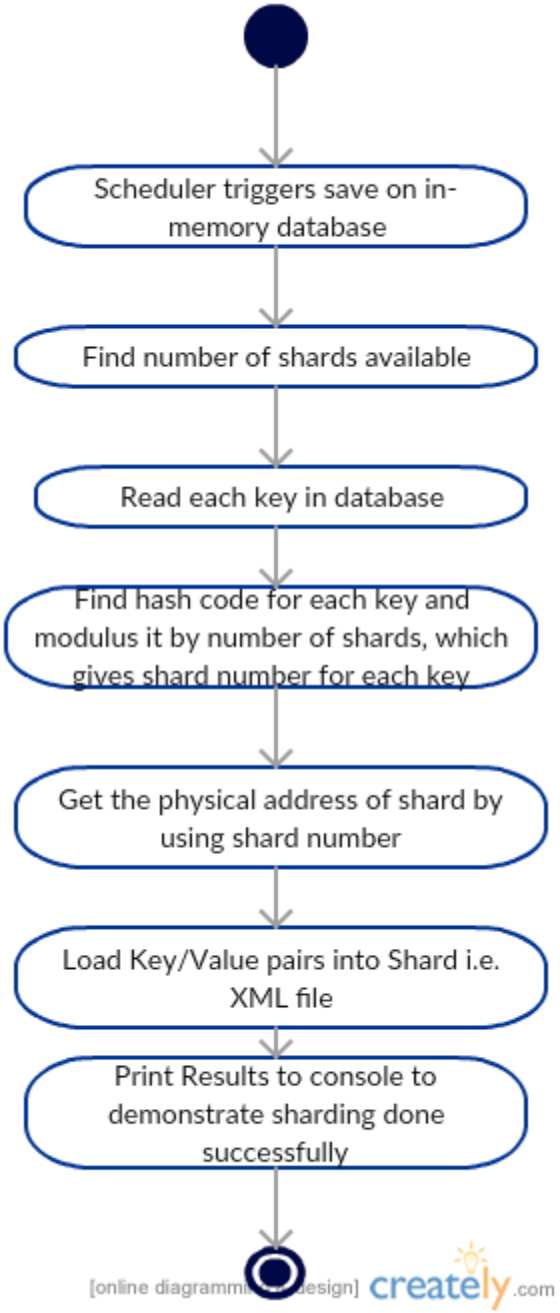

Figure 2: Activity diagram for Sharding

4.4. Restore/Augment database contents from an XML file

Database can be restored from an existing XML file as same format as persisted. When the database is restored, all the data in the database will be over written by contents in XML file. We can also augment data to database from an XML file which will append contents to existing database. Later augmentation, contents of database can be again written to XML file.

4.5. Support a variety of queries, both simple and compound

Data in the database is stored in the form of Key/Value pairs where each key is associated with an address, which will point to the value object. A new virtual immutable database is constructed from the result of any query that returns a collection of keys. Each key in the immutable database is associated with same address in the original database. Compound queries can be performed on top of new immutable database, which was constructed from query result.

Queries can be performed on database to find the value, children of a specified key, set of all keys matching a specified pattern which defaults to all keys, all keys that contain a specified string in their metadata section, and keys that contain values written within a specified time-date interval.

The activity diagram for query processing is mentioned below

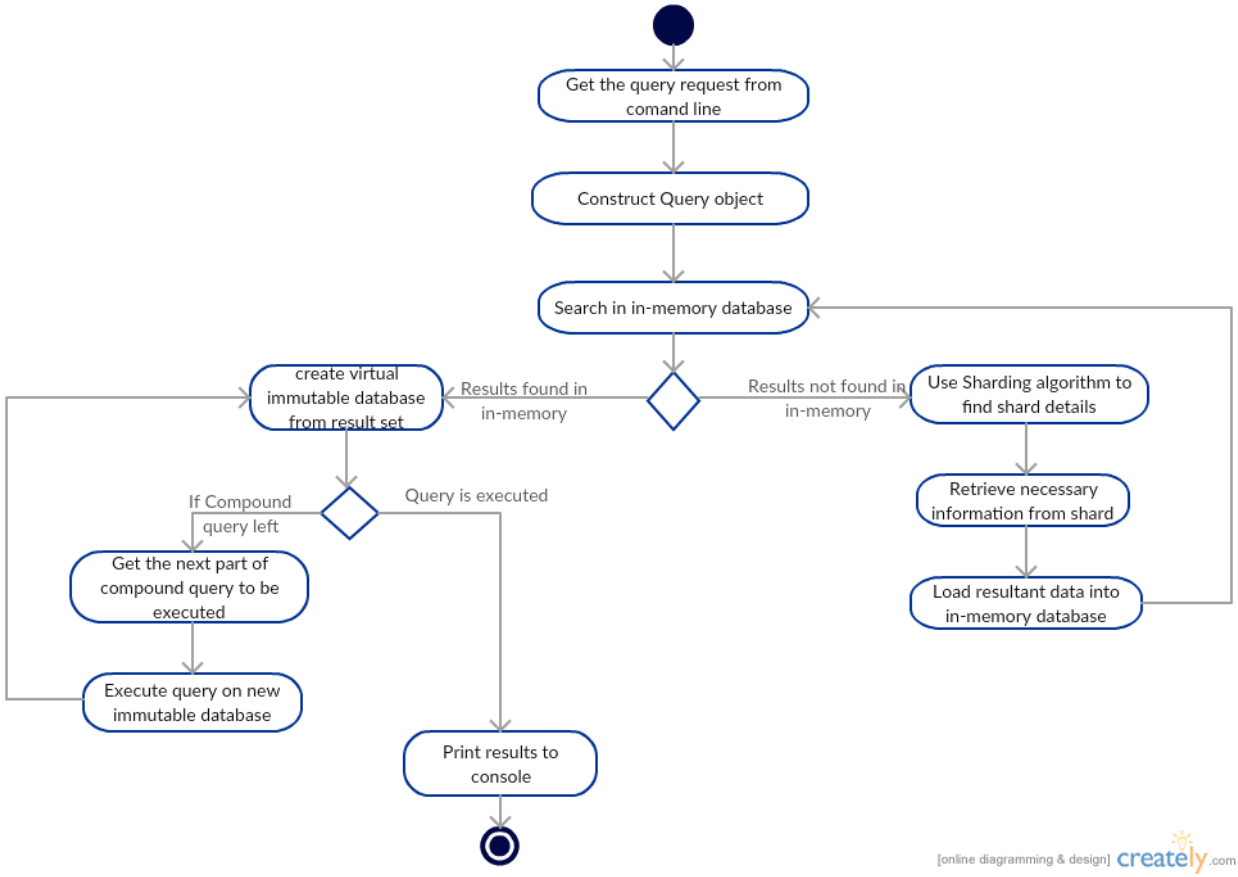

Figure 3: Activity diagram for Query Processing

4.6. Display results

All the requirements shall be demonstrated successfully through a series of discrete tests with display to the console. Query results of key/value database shall be displayed to the user through console.

5. Partitions

This section describes each package in the application which includes a statement of their responsibilities and their interactions with other packages.

Below is the package diagram for Key/Value database

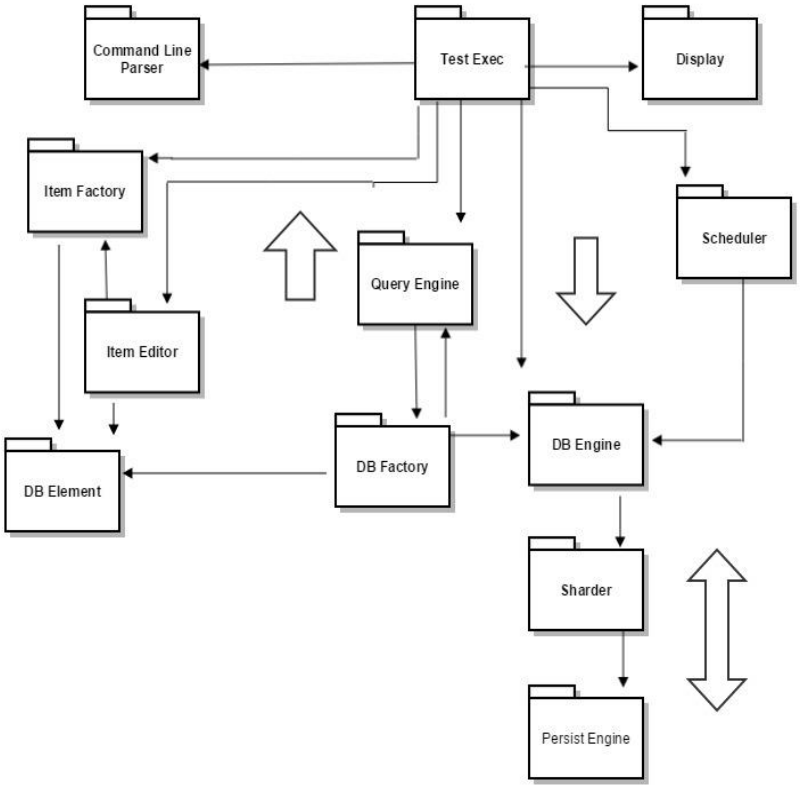

Figure 4: Package diagram for Key/Value database

5.1. Executive

The executive is responsible the main control flow of the application. Executive is the entry point to the application. It takes the input from command line and directs the control to various packages to perform particular task. The executive initiates below sequence of actions

- Read command line input
- Parse XML file
- Construct value object and metadata
- Load data into database
- Querying to database
- Sharding data into persistent storage
- Scheduling
- Call display package to display results

5.2. Command line parser

This package is responsible to parse the command line input from user. It scans the input provided and identifies the operation requested by the user and returns results to executive. Valid operations may include adding/editing/deleting value to/from database, scheduling, querying results, augment database.

5.3. Item module

This module responsibility is to provide wrapper to edit or insert values to database. There are two packages in this module one for Item creation and other for Editing Item.

5.3.1. Item Factory

Item factory is a wrapper module for DB element, provided to construct value and its metadata from input parameters, given by executive module. Item factory generates unique key for each value and its responsibility is to prepare key/value pairs which can be stored in database. Value object consists of

- Metadata:
  - A name string
  - A text description of the item
  - A time-date string recording the date and time the value was written to the database.
  - a finite number (possibly zero) of child relationships with other values.
- An instance of the generic type. This might be a string, a container of a set of values of the same type, or some other collection of data captured in some, perhaps custom, data structure

5.3.2. Item Editor

When a user requests to change any value in database, Executive uses this package to update any value in database. This package handles responsibility of Editing of values including the addition and/or deletion of relationships, editing text metadata, and replacing an existing value's instance with a new instance. We can't edit keys in this project.

5.4. Query processing Module

This module is responsible to create, store key/value database, process user queries. The responsibilities of this module can be divided into below three tasks

5.4.1. DB Engine

The DB engine is repository for in-memory storage for data dictionary object using hashing technique. The responsibilities of DB Engine are as follows

- It interacts with Executive package to get Key/Value pairs that are constructed by Item factory to store those in data dictionary.
- Holds the relationships between keys in the form on parent/child relationships.
- Interacts with sharder component, which then partitions data into smaller chunks based on date criteria.
- Data is organized in form of least recently used shards stored in in-memory.

DB Engine uses below class to store Key/Value pairs in database. It receives Key and value as input and feeds them to data dictionary.

```
public class noSQLdb<Key,Value>
{
 private Dictionary<Key,Value>
}
```

5.4.2. Query Engine

The responsibility of the query engine is to process the user queries. It stores all the user queries in a queue and process one after other on top of Key/Value database to get the results.

- Interacts with DB Engine to run user queries on database.
- Communicates with DB Factory to execute compound queries on top of other query results.
- Shall provide query results back to Executive, which will be displayed to user using Display module.

5.4.3. DB Factory

The responsibility of DB Factory is to provide interface to perform query operations.

- DB Factory provides an interface to create new virtual database from the set of keys resultant from query execution. These database is virtual which will look like original database where keys will point to same value object as the original keys point to. These new database can act as a source for compound queries.
- It also provides an interface to perform queries on database i.e. provides interface to add, edit, delete or update operations.

5.4.4. DB Element

DB element is the package which contains the actual Key and Value object structure.

- DB Element is used by Item factory to construct value object.
- DB Factory uses DB element to read the value object that is constructed

5.5. Sharder Module

This package provides distributed query support, database can retain full query expressive power even when data is distributed across hundreds of servers. It reduces ability to perform complex data queries.

This package is responsible for following activities

- Shall divide data into smaller chunks based on particular criteria (depends on nature of data) and store them in multiple files. Sharder contains a routine which will identify to which shard each row of data will go.
- Contains a lookup table to maintain shard information (location of shards).
- Based on user input, Scheduler interacts with sharder to shard data.
- Sharder interacts with persist engine to store shards into multiple persistent files.

5.6. Persist Engine module

The responsibility of the package is to store database into a persistent store. The responsibilities of the package include

- Shall interact with Sharder to receive data shards and store them to a persist store.
- Shall store shard information into multiple XML files.

- Shall also support Restoration/Augmentation of database from already existing files.

5.7. Scheduler module

The responsibility of this package is to schedule the database contents to persist store.

- Executive interacts with this package, which provides a positive time interval or number of writes after which the database contents are persisted. The save process shall continue till the input condition is failed.
- This package will have a thread which interacts with DB Engine to periodically check numbers of writes on database. If condition is satisfied then stores data from in-memory to persist storage.
- If the input is time interval, Scheduler will save data to persist store periodically for those time intervals.

5.8. Display

The display component is responsible to display results of application onto console. Executive interacts with display module to demonstrate all the requirements. It displays query results from query engine and also success/error messages for other operations. It also displays error messages if user input is not proper.

6. Critical Issues

6.1. Implementing Data Sharding

Determining the optimum method for sharding the data is highly variable, change from application to application. There are numerous methods for deciding how to shard data and it is closely related to nature of data we are sharding. It's important to understand the transaction rates, table volumes, key distribution, and other characteristics of the application before sharding.

Solution: The key can be integer, string or any value type. First I will feed the key to hash function which will generate a hash code for each key. As a designer I am completely aware of how many shards I am going to split data into. Now I will apply modulus on hash value generated for key to number of shares, which will result in shard number that particular key goes. While querying, if key doesn't exist in in-memory then we have to search in shards and same above solution can be applied to find on which shard key resides.

We can increase number of shards when data becomes more in shards, same as how vector works internally. When data in shards is full then I will double the number of shards and restructure the entire data in all shards. We can make the data in shards evenly distributed by generating keys in particular sequence order. Here sharding is completely based on key, so if we generate keys in sequence order then we can ensure that data in shards are evenly distributed.

6.2. Support querying

Key/Value data store does not provide any query language to support queries on top of database. User will never know how to get results from database now. One more issue related to querying is how can we get results faster and efficient when data is distributed in many shards?

Solution: To support Queries on database we will provide a wrapper to user, where user will just provide command and input values to perform any operation. Application will first identify the command and then calls specific operation based on the command. Commands shall be given as below

/e for editing values

/d for deleting key/value pair

/a for adding new key/value pair

/q for querying database

The second issue might arise in situations where we are searching for all the keys that have particular value. If search is performed on top of value then we have to search for results on all the shards. Map-Reduce is one solution to get better results. We will write a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Mapper will search on all shards in parallel to generate intermediate values. We can make this search faster by creating indexes on most frequently used values.

6.3. Implementing compound queries

To implement compound queries it doesn't make sense to search entire database for each internal query it contains. We have to find some mechanism to implement these compound queries effectively.

Solution: The compound query shall be split into individual queries first. Then we will start querying from inner query by creating new virtual database for each query result, which will act as source for its outer query. This process will run in a loop till complete query is executed and finally results are displayed to the user.

6.4. Constructing new immutable database

We will construct a new virtual database from resultant set of keys in which keys will have reference to value object in original database, looks like new database. In this case some keys will have child relationship with other keys which are part of new immutable database. If we remove those references it will create dangling references to invalid child relationships.

Solution: The solution to this problem is to keep track of all the keys that exists in new immutable database. While displaying results I will check whether keys in child relationships exist in database or not, if not I will ignore those keys, display only other keys.

6.5. Maintaining data in shards reliable and fault-tolerant.

Application must be reliable and fault-tolerant, and cannot be subject to frequent outages. In fact, due to the distributed nature of multiple shard databases, the criticality of a well-designed approach is even greater.

Solution: To handle this scenario we will ensure that at least 2 copies of shard data are available at the time of outage or failure. This can be handled by application by using parity bit called 'Active'. Application will set this bit automatically to '0' if main shard is not available, then automatically application will perform operations on replicated shards.. But this approach comes with the cost of reliable replication mechanism.

6.6. Maintaining Eventual Consistency

Eventual consistency by definition says that the storage system guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value. Good database must provide Eventual Consistency.

Solution: Write operation happens only active shards and read happens on cold shards, i.e., replicated shards. For every particular time interval active shard updates data into replicated shards with new updated data. Reading data has to be locked during the period of update or replication process to ensure that no other processes are updating the same data. The time required for a query to return the latest value is usually very short. However, in rare situations when the replication latency increases, the time can be much longer.

6.7. Authentication and authorization issue.

Safe Authentication and authorization mechanism shall be provided to when clients connect to database from remotely. Each client can connect to database by using correct password. The password will be provided from application side to each client. This issue will be better addressed in project 3 and 4.

6.8. Inconsistent input data

Input data in the XML file might contain inconsistent values in many cases like not providing required fields or value type mismatch etc. If this issues are not well handled application doesn't know what to do in such scenarios.

Solution: We will design an application specific schema to provide input in particular format. In that case application will easily find which all inputs are inconsistent and write those back to user.

7. References

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

2) http://codefutures.com/database-sharding/

3) Project starter code given by Prof. Jim Fawcett

8. Appendix

8.1. Data dictionary structure

The objects of this DBElement class are the values in our key/value database. The payload field contains the actual value, the child relationships are stored in the List data structure.

```
public class DBElement<Key, Data>
  {
    public string name { get; set; }
    public string descr { get; set; }
    public DateTime timeStamp { get; set; }
    public List<Key> children { get; set; }
    public Data payload { get; set; }
 }
```

DB Engine class contains the .net dictionary object, functions are exposed which performs the specific operations over the data dictionary. Few of the notable functions are implemented in the below code snippet.

```
  public class DBEngine<Key, Value>
  {
    // - Data Dictionary which holds the key value pairs
    private Dictionary<Key, Value> dbStore;
    public DBEngine()
    {
      dbStore = new Dictionary<Key, Value>();
    }
```

```
// - DB Engine class which contains the .net dictionary object
 public bool insert(Key key, Value val)
 {
   if (dbStore.Keys.Contains(key))
     return false;
   dbStore[key] = val;
   return true;
 }
// - DB Engine class which gets the value from the .net dictionary object
 public bool getValue(Key key, out Value val)
 {
   if(dbStore.Keys.Contains(key))
   {
     val = dbStore[key];
     return true;
   }
   val = default(Value);
   return false;
 }
    // - DB Engine class which contains the .net dictionary object
    public IEnumerable<Key> Keys()
    {
          return dbStore.Keys;
    }
    // - DB Engine class which deletes the key from the .net dictionary object
    public bool deleteKey(Key key)
    {
           if (dbStore.Keys.Contains(key))
              {
                 dbStore.Remove(key);
              }
```

```
            return false;
        }
    }
    // - DB Engine class which updates the value from the .net dictionary object
    public bool updateValue(Key key, Value val)
    {
            if (dbStore.Keys.Contains(key))
            {
                dbStore[key] = val;
                return true;
            }
            return false;
    }
```

8.2. Sample shard information in XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<kvdatabase>
<data-pair>
<key>KEY_433383347</key>
<value>
<metadata> <name> C# in a NutShell </name>
<description> Programming Book </description>
<timestamp> 16-SEPTEMBER-2016 20:54:16</timestamp>
<children> <child> KEY_120463267 </child>
<child> KEY_3738836732 </child>
<child> KEY_63238336823 </child>
<child> KEY_3626836736 </child></children>
</metadata>
<payload>payload@1234</payload>
</value>
</data-pair>
</kvdatabase>
```