Software Requirements Specification

for

Distributed Hash Table

Version 1.0 approved

Prepared by <author>

<organization>

<10/23/19>

Table of Contents

Table of Contents ii

Revision History ii

1. Introduction 1

1.1 Purpose 1

1.2 Product Scope 1

1.3 References 1

2. Overall Description 2

2.1 Product Perspective 2

2.2 Product Functions 2

2.3 User Classes and Characteristics 3

2.4 Design and Implementation Constraints 4

2.5 Assumptions and Dependencies 4

3. Specific Requirements 4

3.1 User Interfaces 4

3.2 Functional Requirements 5

3.3 System Feature 2 (and so on) 5

3.4 Performance Requirements 5

3.5 Logical Database Requirements 5

3.6 Software Quality Attributes 5

Appendix B: Analysis Models 6

Appendix C: Requirement Traceability Matrix (TBD) 6

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Andrew | 10/14/19 | Initial changes | 0.0 |
| Andrew | 10/20/19 | Added Product Perspective, Functions, Functional Requirements (CRUD), Traceability Matrix | 0.7 |
| Daniyal | 10/20/19 | Wrote Section 3.5(logical database requirements, added to section 3.4(Performance Requirements), added to section 2.3 |  |
| Palak | 10/20/19 | Wrote sections 2.4, 3.4, Added to sections 1.1, 3.2 |  |
| Rachana Mandal | 10/20/19 | Wrote sections 1.2, 2.5, 3.6 |  |
| Palak | 10/21/19 | Added to section 2.3 |  |
| Rachana | 10/22/19 | Added to 2.1 and references [4], [5] |  |
| Daniyal Adzhiyev | 10/22/19 | Wrote Section 3.1 (UI) |  |
| Palak | 10/22/19 | Wrote section 3.3 |  |

# Introduction

## Purpose

Cloud computing is the computing trend for the next decade. Many applications like online stores, discussion forums, and many others; require storage of state. This project provides cloud storage API for such applications.

This project also addresses the need for scaling without reconfiguring the entire data structure. Our purpose is to provide an in-memory distributed hash table which can accommodate new values without rearranging all nodes, which makes it more time efficient.

<Identify the product whose software requirements are specified in this document, including the revision or release number. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.>

## Product Scope

A Distributed Hashtable is used to store data in-memory across multiple nodes. It will be deployed in the cloud as an in-memory data service. The primary benefit of this system is that it allows data to scale along with the service. It is important due to its high performance and self-administrating properties such as internal redistribution of memory among its nodes.

It will provide high performance and scalable memory cache service for any application using this service such as online stores, discussion forums and many others. It is also reliable and fault tolerant even if any node goes down due to power failures or as nodes join or leave as the system scales up or down.

<Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here.>

## References

|  |  |
| --- | --- |
| [1] | Code Academy, "What is CRUD?," 2019. [Online]. Available: https://www.codecademy.com/articles/what-is-crud. [Accessed 19 10 2019]. |
| [2] | Amazon, "Amazon DynamoDB," Amazon, 2019. [Online]. Available: https://aws.amazon.com/dynamodb/?sc\_channel=PS&sc\_campaign=acquisition\_IN&sc\_publisher=bing&sc\_medium=ACQ-P%7CPS-BI%7CBrand%7CDesktop%7CSU%7CDatabase%7CDynamoDB%7CIN%7CEN%7CText&sc\_content=dynamodb\_e&sc\_detail=dynamodb%20aws&sc\_category=Database&sc\_segmen. [Accessed 19 10 2019]. |
| [3] | Microsoft, "Microsoft Azure Tables," Microsoft, 2019. [Online]. Available: https://azure.microsoft.com/en-us/services/storage/tables/?&OCID=AID2000128\_SEM\_Bb18uD39&MarinID=Bb18uD39\_79439783846512\_%2Bazure%20%2Btable\_bb\_c\_\_1271035491868426\_kwd-79439741380976:loc-190&lnkd=Bing\_Azure\_Brand&msclkid=12b252792f5214389af80fd2a251452f&e. [Accessed 27 7 2019]. |
| [4] | *“Performance at Scale with Amazon ElastiCache”*, https://d1.awsstatic.com/whitepapers/performance-at-scale-with-amazon-elasticache.6a4eb08a74d1ff0a63b3d527b1b578e8974fbb31.pdf |
| [5] | *“Azure Cache for Redis description”*, https://docs.microsoft.com/en-us/azure/azure-cache-for-redis/cache-overview |

Please add any throughout the document here.

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

# Overall Description

## Product Perspective

In computer science, hash tables are one of the fundamental components of many algorithms. In the family of data structures Hashtables offer some of the fastest CRUD operations. [1] With the advent of cloud computing, arised a need for a new type of Hashtable, one that is distributed across a pool of machines, or even the planet.

There are a number of solutions available for a Distributed Hashtable, such as Amazon DynamoDB, and Microsoft Azure Tables. [2] [3] The purpose of this project is to create a custom implementation, since we want to use an in-house hashtable. The Distributed Hashtable will support CRUD operations just like a normal hashtable. Additionally, besides of relying on the resources of a single computer, it will be hosted on multiple nodes allowing for high scale and availability. This is similar to how Amazon ElastiCache [4] and Azure cache for Redis [5] provides fast in-memory cache service in the cloud to any other applications utilyzing the service which is also robust and highly scalable.

<Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.>

## Product Functions

The major functionality of the distributed hashtable will support CRUD operations. It will provide an API that can be used from the internet. The API would support posting new values, getting and updated existing values, and removing previously added values. The APIs provided by Distributed Hashtables will be atomic, consistent, and isolated. The distributed hashtable will behave much like a normal hashtable to the user.

The hashtable will allow any kind of data to be stored in the value. The values that will be stored will be strings/objects that are not modified. However, it will be up to the API caller to serialize the data, similarly to a typical hashtable. Some of the data stored in the hashtable could be:

* Images or photos
* HTML product information
* Transactions or reports
* Any type of indices

Largely, what the API callers store in the hashtable will be up to them. The API callers will need to consider, how often the data modified, what parts of the data will be modified; such that the data is partitioned efficiently in the distributed hashtable. Additionally, API callers will be able to create multiple distributed hashtables for different kinds of data or services.

Out of the box the hashtable will provide methods of dealing with concurrency. For example, for insertion of a duplicate key, the hashtable will indicate to the user if this was the first key, and allow the API user to change the value, or abort the operation in such cases. This functionality is critical for API users for handling duplicate requests, and retries.

The distributed hashtable will be easy to deploy and consume by a customer. After the hashtable is deployed, it will be possible to add or remove more nodes to the cluster. Adding more nodes to the cluster will automatically move data as needed without anything needed from the API users.

The distributed hashtable will be fault tolerant, allowing some machines in the cluster to be restarted, or turned off. However, if all the machines in the cluster are turned off, then the hashtable will not be able to service the API queries. In such cases, it is expected that the API caller plan for situations where all of the machines might be inaccessible.

<Summarize the major functions the product must perform or must let the user perform. Details will be provided in Section 3, so only a high level summary (such as a bullet list) is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or object class diagram, is often effective.>

## User Classes and Characteristics

**2.3.0 General User**

A user who will work with the UI for commercial purposes to query and retrieve data from the distributed hashtable as needed.

### API User

This will be a developer, who will want to learn how to program the APIs.

### Distributed Hashtable Administrator

This will be a person responsible for updating the distributed hashtable, as well as updating the machine and networking hardware, Operating System, etc.

### Architect

This will be someone who decides how to use the distributed hashtable.

### Database Programmer

This is a developer who may want to understand how to import, or export business data from/to the distributed hashtable. They may also be interested in knowing how to structure DAL to the distributed hashtable, as well as add background tasks to complete transactions. A database programmer is also interested in knowing how to backup/restore the business data.

### Business Manager

This will be someone responsible for purchasing hardware, networking equipment; or provisioning hardware required in the cloud. They will be interested in understanding the efficiency of the distributed hash table, and how much money it will cost to run out of pocket.

### Distributed Hashtable Expert

This will be a person who works closely with the architect to give inputs on the development and maintenance the distributed hashtable code.

**2.3.7 Distributed Hashtable Developer**

This will be a person who works closely with the Distributed Hashtable Administrator to develop and maintain the distributed hashtable code.

<Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the most important user classes for this product from those who are less important to satisfy.>

## Design and Implementation Constraints

One of the implementation constraints is the time required to setup a distributed computing deployment on platforms like AWS. Also, due to limitations on the AWS free account, we have a limit on numbers of node when we will be generating random key value pairs for our distributed hash table. There are hardware constraints based on how much storage is available for the distributed hash table. The absence of a strong user interface is also a design constraint.

<Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer’s organization will be responsible for maintaining the delivered software).>

## Assumptions and Dependencies

**Concurrency**

The assumption for the distributed hashtable is that the API callers will be calling the APIs concurrently in a distributed environment. This means that some extra care will need to be taken in cases where:

* Two keys are added concurrently
* A get/remove Key is performed on a key that doesn’t exist

It will be up to the API caller to handle cases where they might expect a key to exist, but it was already removed.

**Dependencies**

The dependencies for this project are:

* Java 8 as the development environment
* AWS as the cloud solution where the project will be hosted.
* Kubernetes for deployment and management of this project in the AWS cloud

<List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project, unless they are already documented elsewhere (for example, in the vision and scope document or the project plan).>

# Specific Requirements

## User Interfaces

The Distributed Hashtable will incorporate a UI on a simple standalone desktop application. The purpose of the application will be to help the development team with testing and assessing the performance of the DHT. The UI application will allow users to query, update, and add/remove new key, value pairs. In addition it will display which node the data came from and the run-time in milliseconds to perform each operation. The UI will show the number of active nodes and offline nodes in the system.

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

## Functional Requirements

<This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.>

### Support CRUD operation

The distributed hashtable should be able to support Create, Read, Update and, Delete operations. It will be able to scale out, and be fault tolerant.

<Don’t really say “System Feature 1.” State the feature name in just a few words.>

4.1.1 Description

The distributed hashtable will allow for an API caller to add values. The values would be accessible using a key defined by the API user. The API should be simple to use, and not require major modifications to the calling service. The Distributed Hashtable should be fast enough to be used in a real time application.

The distributed hashtable will store keys and values as UTF-8 strings. The string may not be acceptable for Chinese alphabets, it will be up to the API caller to re-encode the strings, or to convert these strings to in memory objects. There are many excellent libraries available for serializing and deserializing data.

The data size stored is expected to be <1 MB per key. The maximum size of the distributed hashtable will depend on the number of machines available and the sum of the sizes of the keys. The key can be a UTF-8 string up to 512 characters long. No localization is to be considered for comparing equality of two keys. Two keys are equivalent if the ordinal byte match.

<Provide a short description of the feature. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).>

4.1.3 Functional Requirements

<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use “TBD” as a placeholder to indicate when necessary information is not yet available.>

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>

**REQ-1: Hashtable to support Get operations**

The get operation will retrieve an existing key from the hashtable. The API should be like

String Get(string key)

The user passes a key to the API and the return is the value from the hash-table. If the KEY is null, an InvalidKeyExceptionIt is possible for a value to be NULL. Therefore if the key does not exist in the distributed hashtabke, a KeyNotFoundException should be raised.

Get will work immediately after AddorUpdate was called. Get will return the most recent value that was added or updated, without any delay.

**REQ-2: Hashtable Add and Update operations**

The API for AddorUpdate is:

string AddorUpdate(string key, string value, lambda updateOnConflict)

It will accept a string key and value. The AddOrUpdate function should handle concurrency. In these cases, the same key may be added by two different machines at the same time. When this happens, the API should invoke the updateOnConflict lambda to let the caller update the final value.

The function will return the new value was effectively added.

**REQ-3: Hashtable to support Remove function**

The API for remove is

Bool Remove(string key)

Remove will accept a key as a string. Because Distributed Hashtable is designed for concurrency, will not throw an exception if a key doesn’t exist. Instead, if a key doesn’t exist, Remove will return false if no change was made. If a key was removed, then this method will return true.

REQ-1:

REQ-2:

### Scaling

REQ-1: The Distributed hash-table will scale with the size of the cluster

When new information/nodes are added, the hash table will not rearrange all existing data. In We achieve time efficiency by rearranging only some information.

## System Feature 2 (and so on)

1. Client/Server system to do the CURD operations and access information stored in the nodes
2. Distributed hashtable as a data structure to organize and maintain our project
3. Cloud setup to deploy the project on services like AWS

## Performance Requirements

1. Scalability: The time efficiency achieved with adding new information to the hashtable is a major performance requirement as the goal is to only rearrange some information to accommodate new data.
2. Reliability: The Distributed Hashtable must be reliable and useable at all times. If specific nodes go down, the Key, Value pairs should be redistributed among active nodes and cause minimal impact to the user. It should be useable 99.99% of the time.
3. Accuracy: The Distributed Hashtable must be accurate. Keys should give the correct Value in the table at all times. The Key, Value pairs need to be accurate and updated in real-time. There should be no out-of-date or incorrect values for any of the data.
4. Speed: Querying the Distributed Hashtable must be near instantaneous. Add/Update or Remove Operations should take less than 100ms while get operations should take less than 50ms.
5. Security: The Distributed Hashtable must be secure. The data in the DHT must only be accessed by authorized users. It should be secure from any external source or malicious attackers attempting to gain access to the data.
6. Cloud Deployment: Deploying the in-memory hashtable on an online computing platform is another performance requirement which will be achieved using technologies like Kubernetes and AWS.

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

## Logical Database Requirements

While the DHT will act as a Database of its own the project will require a back-up logical database of all Keys and their respective Values. As the security, stability, and accuracy of the data is of upmost importance the back-up logical database must meet be flawless in all of these categories.

* 1. Security – The database must secure the data of the project. Data should only be accessed or altered by persons that are authorized. It should be safe from malicious attacks that attempt to gain access.
  2. Stability – The database must be stable at all times. Authorized users should be able to access and update data at any time. The data cannot be lost and there must always be a secure backup of all the data.
  3. Accuracy – The data must be accurate. The data needs to be up to date in real time. If any changes are made to the data it needs to be immediately updated in the database as well. The database should not be out of date or hold any incorrect values.

## Software Quality Attributes

Some of the important software quality attributes of this project are:

1. Performance

It will provide a real time distributed in-memory memory cache allowing applications to retrieve the data from memory without the need to fetch the data from the physical storage every time.

For desired performance characteristics of the Distributed Hashtable are:

|  |  |
| --- | --- |
| **Operation** | **Latency** |
| AddOrUpdate | <100 ms |
| Get | <50 ms |
| Remove | <100 ms |

1. Scalability

The system will be scalable. It will be deployed in the cloud and with an initial set of nodes for the distributed memory system. As the application using this system grows and demands increased cache, this system can auto scale by adding newer nodes to meet the demand.

|  |  |
| --- | --- |
| **Operation** | **Throughput** |
| AddOrUpdate | 500 req/s |
| Get | 1000 req/s |
| Remove | 500 req/s |

1. Reliability

The distributed hashtable will be reliable and fault tolerant. Even if a few nodes go down due to power failures or any other reasons, the system continues to function correctly. It will also self-administer by redistributing its storage amongst the nodes as the nodes in the system gets added or removed.

As mentioned previously the distributed hashtable will exhibit ACID (i.e. Atomicity, Consistency, Isolation, Durability) properties.

The distributed hashtable will be designed for an availability of 99.99%. The reliability of the distributed hashtable will also be near 100%. All the values are to be returned to the API callers byte accurately as they were added.

The distributed hashtable will be open-source and extensible by third-parties. It will be runnable from Kubernetes on AWS, or other platforms. It will also be runnable from Dev machine for testing, and emulation purposes.

The API will be interoperable and usable from different Operating systems, Languages, and Platforms.

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

Appendix C: Requirement Traceability Matrix (TBD)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Item | Functional Requirement | Input | Output | Description | State Variable | Possible States |
| AddOrUpdate | CRUD operations | Key, value | Value added | Adds or updates a value in the dictionary | Key | Added, Removed |
| Get | CRUD Operations | Key | Value, or KeyNotFound | Retrieves a value from the distributed hashtable | AddOrUpdate(key) | Value, or KeyNotFound |
| Remove | CRUD Operations | Key | True if changed, false otherwise | Removes an item from the hashtable | AddOrUpdate(key) | None, List of contacts |
| Hashtable browser | UI | Key, Value | Updated UI (depending on key) | Shows a list of keys, values | UI | Shows current state of hashtable, nodes |

<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>