CS 6378 Advanced Operating Systems Project 2

Software Design Document for Distributed Database Project

Shruthi Deshpande

The University of Texas at Dallas

Contents

[1. Scope 3](#_Toc386672976)

[2. Motivation 3](#_Toc386672977)

[3. System Assumptions & Requirements 4](#_Toc386672978)

[4. Design Considerations 4](#_Toc386672979)

[5. Design Description 4](#_Toc386672980)

[5.1 Techniques used in Dynamo 5](#_Toc386672981)

[5.1.1 Partition Algorithm 6](#_Toc386672982)

[5.1.2 Replication 6](#_Toc386672983)

[5.2 Architectural constraints and functional requirements 6](#_Toc386672984)

[5.2.1 Data model 7](#_Toc386672985)

[6. Functional Requirements of the distributed system. 7](#_Toc386672986)

[6.1 Use case diagrams 8](#_Toc386672987)

[6.1.1 Use case view of the system. 8](#_Toc386672988)

[6.1.2 Use case view of the client choosing any one of the 3 replicas to read object 9](#_Toc386672989)

[6.1.3 Use Case view of the client inserting/updating object into at least two replicas. 10](#_Toc386672990)

[6.1.4 Use Case view of the client that is unable to access at least 2 servers that are required to store an object. 11](#_Toc386672991)

[6.2 Sequence Diagrams. 12](#_Toc386672992)

[6.2.1 Sequence diagram for failure case of operation (Insert/Update) when 1 replica is available 12](#_Toc386672993)

[6.2.2 Sequence diagram for successful case of operation (Insert/Update) when 2 replicas are available 13](#_Toc386672994)

[6.2.3 Sequence diagram for successful case of operation (Insert/Update) when 3 replicas are available 14](#_Toc386672995)

[6.2.4 Sequence diagram for read operation on randomly selected replica 15](#_Toc386672996)

[6.2.5 Sequence diagram for one client failure case of update operation when there is a concurrency in updating the same object 16](#_Toc386672997)

[6.2.6 Sequence diagram for successful case of 2 clients Update operation when there is no concurrency on same object: 17](#_Toc386672998)

[6.3 Class Diagram 18](#_Toc386672999)

[6.3.1 Classes Description: 18](#_Toc386673000)

[6.4 CONCURRENCY ASPECTS 19](#_Toc386673001)

[7. Test Cases & Results 20](#_Toc386673002)

[TestCase-1: Ensuring each client performs the Insert at three servers numbered: H(Ok)modulo 7, H(Ok)+1 modulo 7, and H(Ok)+2 modulo 7, when it insert an Object or not. 20](#_Toc386673003)

[TestCase-2: Ensuring each client performs the Update at three servers numbered: H(Ok)modulo 7, H(Ok)+1 modulo 7, and H(Ok)+2 modulo 7, when it inserts an Object or not. 20](#_Toc386673004)

[TestCase-3: Ensuring client should be able to randomly choose any of the three replicas of an object when it wishes to read the value of the object. 21](#_Toc386673005)

[TestCase-4: Ensuring when a client wishes to insert an object into the data repository, it should be able to successfully perform the operation on at least two, and if possible all the three servers that are required to store the object. 21](#_Toc386673006)

[TestCase-5: Ensuring when a client wishes to update an object into the data repository, it should be able to successfully perform the operation on at least two, and if possible all the three servers that are required to store the object. 21](#_Toc386673007)

[TestCase-6: Ensuring when a client wishes to update an object into the data repository, it should fail the operation if less than 2 replicas available. 22](#_Toc386673008)

[TestCase-7: Ensuring if two clients try to concurrently write to the same object and at least two replicas are available, the two writes must be performed in the same order at all the replicas of the object. 22](#_Toc386673009)

[Works Cited 22](#_Toc386673010)

# Scope

This document will describe the motivation behind the design of a distributed database system followed by details about high level design and algorithm details of the project. The project is based on the principles of Amazon Dynamo’s highly available key value store (DeCandia, et al., 2007) and the Google File System (Ghemawat, Gobioff, & Leung, 2003). This document includes design motivations, software requirements, design description, use case view, sequential flow of communication between servers and clients, discussion on concurrency issues and results.

# Motivation

Google has about 13 data centers [Figure 1] housing 900000 servers all over the world. These commodity and high performance computers are on multiple clusters that are distributed worldwide. There is a need to serve thousands of queries per second and a query can read 100’s of MBs of data. One such query can consume 10’s of billions of CPU cycles!! A company like Google stores dozens of copies of the entire Web!



Figure 1 Google data center

Amazon data center has around 450,000 servers which store 40 billion objects in cloud. Amazon Web Services (AWS) has 40000 of the servers dedicated to cloud customers. In order to guarantee service level agreements (SLA) there is a need a build a distributed storage system that:

1. Can scale
2. Is simple : key-value
3. Is highly available and fault tolerant.

# System Assumptions & Requirements

* Query model: Simple read and write operations to a data item that is uniquely identified by a key.
* ACID properties : Atomicity, Consistency, Isolation and Durability
* Efficiency: Latency requirements which are measured at the 99.9th percentile of the distribution.
* Other assumptions: Operation environment is assumed to be non-hostile and there are no security related requirements such as authentication and authorization.

# Design Considerations

* The system sacrifices strong consistency for availability.
* Conflict resolution is executed during read instead of write.
* Incremental scalability, symmetry, decentralization and heterogeneity.

# Design Description

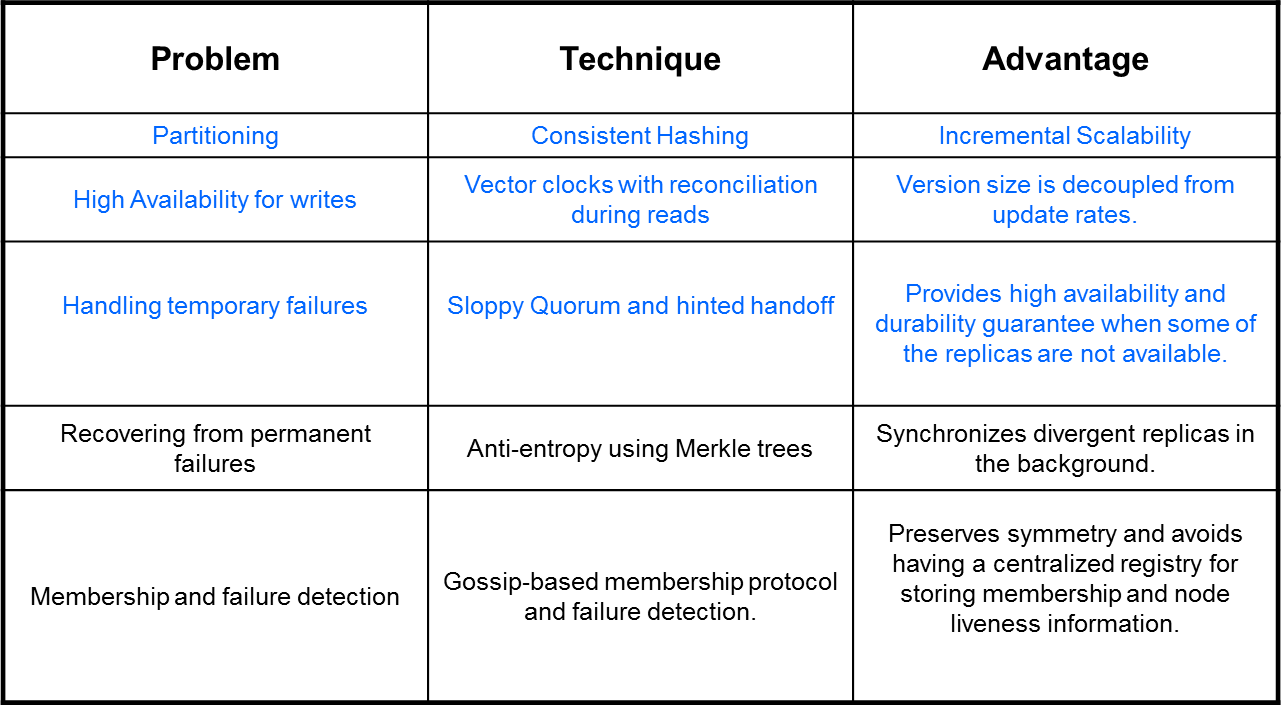
The software design document provides an architectural overview of the project. The document presents different types of abstraction for a clear understanding of the system.

In order to fully document all the aspects of the architecture of the system, the document has the following subsections

:

1. Architectural constraints.
2. Distributed file systems and databases.
3. Use case realization which includes use case view, class and sequence diagrams.
4. Design's concurrency aspects.
5. Results

## 5.1 Techniques used in Dynamo



We implement solutions for the problems highlighted in blue in this project.

### 5.1.1 Partition Algorithm

Figure 2 Seven server ring for consistent hashing

The partition algorithm is based on the principle of consistent hashing and virtual nodes.

*Consistent hashing*: The output range of hash function is treated as a fixed circular space or ring.

*Virtual Nodes:* Each node can be responsible for more than one virtual node. Some of the advantages of using virtual nodes is that if a node becomes unavailable, the load handled by this node is evenly dispersed across the remaining nodes. When a node becomes available again, the new node accepts a roughly equivalent amount of load from each of the other available nodes.

### 5.1.2 Replication

Each data item is replicated at N hosts. The system targets applications that require only key- value access with primary focus on high availability where updates are not rejected even in the wake of network partitions or server failures. In order to achieve high availability and durability, the system replicates its data on multiple data stores. Each data item is replicated at 3 data servers. In the fig, the server B replicates the key k at nodes C and D in addition to storing it locally.

## 5.2 Architectural constraints and functional requirements

The setup we are working with has seven data servers S0, S1 … S6 and five clients C0, C1 ...C4. All the servers communicate with each other on First In, First Out (FIFO) communication channels. The FIFO channels are reliable during operation of the application. The application may experience disruption because of which no message can be communicated across that channel. This requires preserving reliability at massive scale when components in a system fail. We implement Amazon's solution to problems of partitioning and a demand for high availability.

To achieve this level of availability the system sacrifices some degree of consistency under certain failure scenarios. The data store in our system is a simple employee database which has employee id as the primary key. Here most services require primary key access to a data store. The system uses a synthesis of well-known techniques to achieve scalability and availability where data is partitioned and replicated using consistent hashing. We use a hash function H, such that for each object, Ok, H(Ok) yields a value in the range of 0-6.

* A client, Ci can insert / update an Object, Ok by computing the hash function H(Ok), H(Ok) +1 mod 7 , and H(Ok) + 2 mod 7.
* A client Ci can read an object, Ok from on the three servers H(Ok), H(Ok) +1 mod 7, H(Ok) +2 mod 7.

### 5.2.1 Data model

Database name: **Employee Record** (Maintained in HashMap)

**Employee Record** has the following fields:

Employee Id: Data type: integer, primary key.

Employee Name: Data type: String

Employee Designation: Data type: String

Employee Salary: Data type: int

**Example Employee Record:**

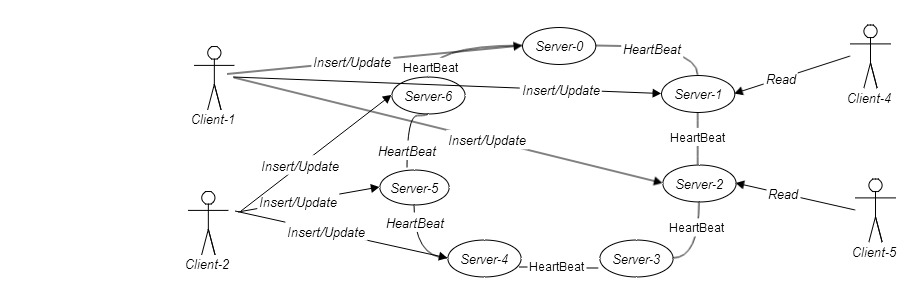
|  |  |  |  |
| --- | --- | --- | --- |
| EmployeeId | EmployeeName | EmployeeDesignation | EmployeeSalary |
| 101 | Shruthi | Software Engineer | 90000 |
| 102 | Shruthi | Technical Lead | 120000 |
| 103 | Shruthi | Project Lead | 120000 |

# Functional Requirements of the distributed system.

1. Client Ci shall insert Object Ok at three servers numbered: H(Ok)modulo 7, H(Ok)+1 modulo 7, and H(Ok)+2 modulo 7, when it inserts an Object.
2. Client Ci shall perform the update of Object Ok at three servers numbered: H(Ok)modulo 7, H(Ok)+1 modulo 7, and H(Ok)+2 modulo 7, when it updates an Object.
3. Client Ci shall read the Object Ok from any one of the three servers : H(Ok)modulo 7, H(Ok)+1 modulo 7, and H(Ok)+2 modulo 7, when it reads an Object.
4. Client should be able to randomly choose any of the three replicas of an object when it wishes to read the value of the object.
5. When a client wishes to update/insert an object into the data repository, it should be able to successfully perform the operation on at least two, and if possible all the three servers that are required to store the object.
6. If a client is unable to access at least two out of the three servers that are required to store an object, then the client does not perform updates to any replica of that object.
7. If two clients try to concurrently write to the same object and at least two replicas are available, the two writes must be performed in the same order at all the replicas of the object.

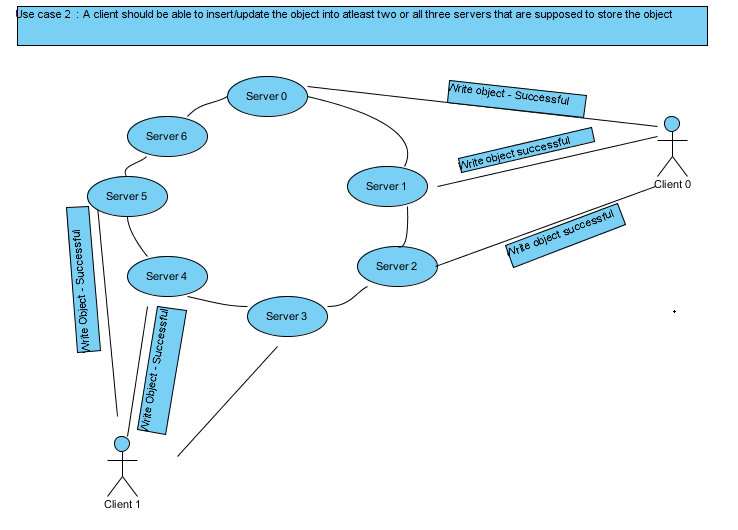
## Use case diagrams

### 6.1.1 Use case view of the system.



### 6.1.2 Use case view of the client choosing any one of the 3 replicas to read object

### 6.1.3 Use Case view of the client inserting/updating object into at least two replicas.

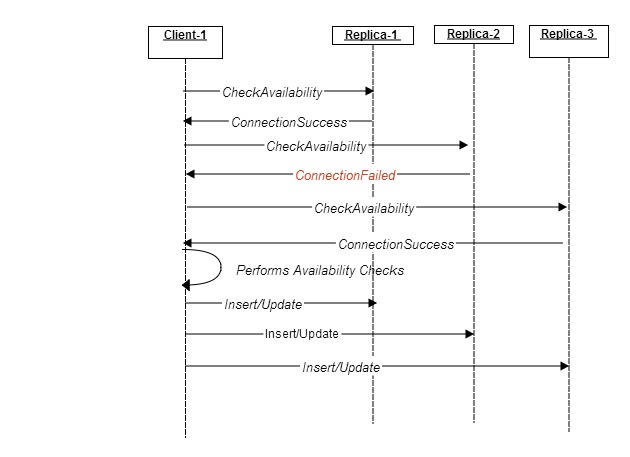


### 6.1.4 Use Case view of the client that is unable to access at least 2 servers that are required to store an object.

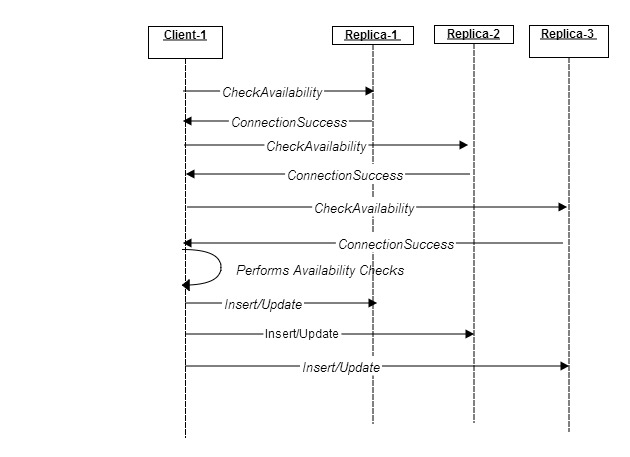
## Sequence Diagrams.

### 6.2.1 Sequence diagram for failure case of operation (Insert/Update) when 1 replica is available

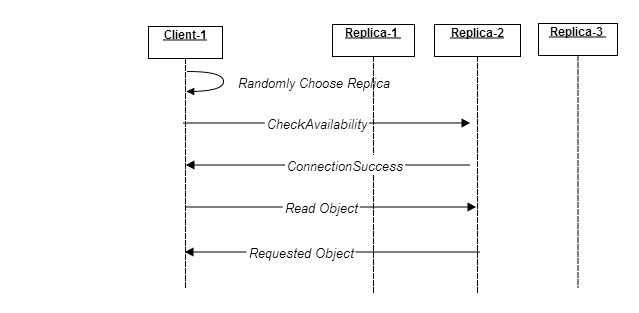
### 6.2.2 Sequence diagram for successful case of operation (Insert/Update) when 2 replicas are available



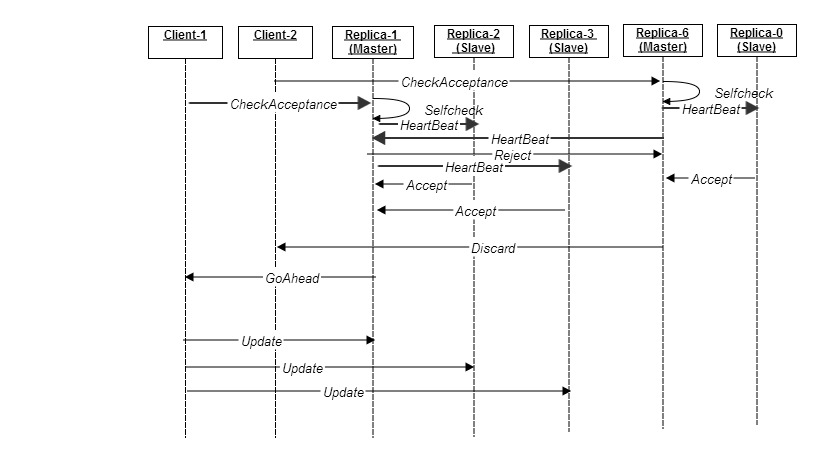
### 6.2.3 Sequence diagram for successful case of operation (Insert/Update) when 3 replicas are available



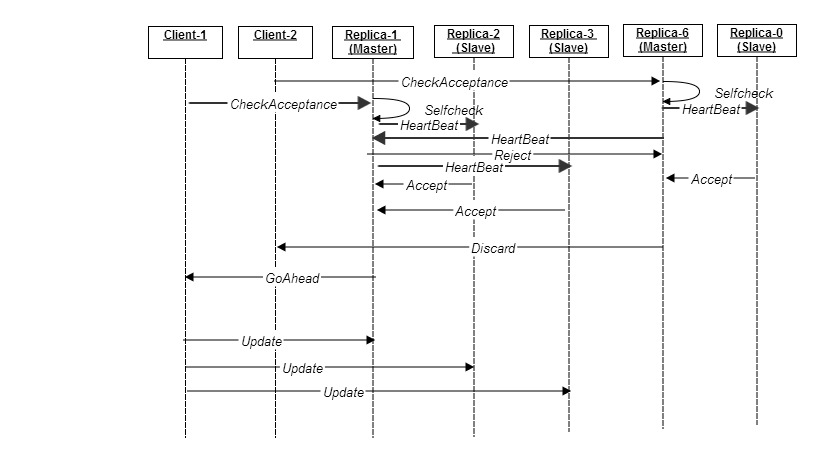
### 6.2.4 Sequence diagram for read operation on randomly selected replica



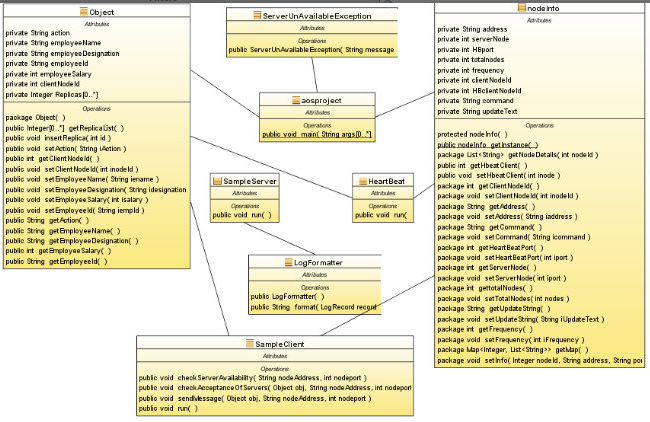
### 6.2.5 Sequence diagram for one client failure case of update operation when there is a concurrency in updating the same object



### 6.2.6 Sequence diagram for successful case of 2 clients Update operation when there is no concurrency on same object:



## Class Diagram



### 6.3.1 Classes Description:

***Class SampleServer:***

This class is responsible to accept insert/update/read requests from Clients and maintain the employee record.

***Class HeartBeat:***

This class is responsible for controlling concurrent access to same object by multiple clients.

***Class ServerUnAvailableException:***

This class is an exception handler class for Server unavailable case.

***Class Object:***

This class contains the common object for insert/update/read employee record.

***Class SampleClient:***

This class is responsible to perform insert/update/read operations on servers.

***Class nodeInfo:***

This is a singleton class, which maintains shareable information between threads.

***Class aosproject:***

This class reads the configurations files and instantiates the server/client threads.

***Class LogFormatter:***

This class is responsible to format the logs of Server.

## CONCURRENCY ASPECTS

It is noted that there is a concurrency issue when 2 or more clients try to update the same object in any of the 3 replicas.

**Observations:**

It is noted that when we tried to update designation field of Employee ID 103 Designation to Technical Lead by client 1 on Replica 0, the operation was successful.

[UPDATE]::103 Shruthi TechnicalLead 200000.

When the object with Employee ID 103 was checked for designation field in Replica 6 and Replica 1, the values were different.

[UPDATE]::103 Shruthi ProjectLead 200000.

This problem is approached by implementing the concepts in Google File System Mechanism to control the access to replicas using Master- Slave mechanism.

Check Access To Replicas

Replica-0 (Master)

GOAHEAD

Client-1

Update

Replica-2(Slave)

Update

Update

Replica-1(Slave)

Check Access To Replicas

Client-2

DISCARD

Check Access To Replicas

# Test Cases & Results

### TestCase-1: Ensuring each client performs the Insert at three servers numbered: H(Ok)modulo 7, H(Ok)+1 modulo 7, and H(Ok)+2 modulo 7, when it insert an Object or not.

$ java aosproject 1 insert

AvailableReplicas =3

[SENT] insert 101 Shruthi SoftwareEngineer 120000 in Replica 4

[SENT] insert 101 Shruthi SoftwareEngineer 120000 in Replica 5

[SENT] insert 101 Shruthi SoftwareEngineer 120000 in Replica 6

AvailableReplicas =3

[SENT] insert 102 Shruthi SoftwareEngineer 120000 in Replica 5

[SENT] insert 102 Shruthi SoftwareEngineer 120000 in Replica 6

[SENT] insert 102 Shruthi SoftwareEngineer 120000 in Replica 0

AvailableReplicas =3

[SENT] insert 103 Shruthi SoftwareEngineer 120000 in Replica 6

[SENT] insert 103 Shruthi SoftwareEngineer 120000 in Replica 0

[SENT] insert 103 Shruthi SoftwareEngineer 120000 in Replica 1

AvailableReplicas =3

[SENT] insert 104 Shruthi SoftwareEngineer 120000 in Replica 0

[SENT] insert 104 Shruthi SoftwareEngineer 120000 in Replica 1

[SENT] insert 104 Shruthi SoftwareEngineer 120000 in Replica 2

### TestCase-2: Ensuring each client performs the Update at three servers numbered: H(Ok)modulo 7, H(Ok)+1 modulo 7, and H(Ok)+2 modulo 7, when it inserts an Object or not.

$ java aosproject 1 update ProjectHead 8 7

ACKNOWLEDGEMENT RECEIVED GOAHEAD

[SENT] update 101 Shruthi ProjectHead 200000 in Replica 4

[SENT] update 101 Shruthi ProjectHead 200000 in Replica 5

[SENT] update 101 Shruthi ProjectHead 200000 in Replica 6

ACKNOWLEDGEMENT RECEIVED GOAHEAD

[SENT] update 102 Shruthi ProjectHead 200000 in Replica 5

[SENT] update 102 Shruthi ProjectHead 200000 in Replica 6

[SENT] update 102 Shruthi ProjectHead 200000 in Replica 0

ACKNOWLEDGEMENT RECEIVED GOAHEAD

[SENT] update 103 Shruthi ProjectHead 200000 in Replica 6

[SENT] update 103 Shruthi ProjectHead 200000 in Replica 0

[SENT] update 103 Shruthi ProjectHead 200000 in Replica 1

### TestCase-3: Ensuring client should be able to randomly choose any of the three replicas of an object when it wishes to read the value of the object.

$ java aosproject 1 read 7

[READING]101 record from Replica 4

[RECORD] 101 Shruthi ProjectHead 200000

[READING]102 record from Replica 0

[RECORD] 102 Shruthi ProjectHead 200000

[READING]103 record from Replica 1

[RECORD] 103 Shruthi ProjectHead 200000

[READING]104 record from Replica 2

[RECORD] 104 Shruthi ProjectHead 200000

### TestCase-4: Ensuring when a client wishes to insert an object into the data repository, it should be able to successfully perform the operation on at least two, and if possible all the three servers that are required to store the object.

$ java aosproject 1 insert

AvailableReplicas =3

[SENT] insert 101 Shruthi SoftwareEngineer 120000 in Replica 4

[SENT] insert 101 Shruthi SoftwareEngineer 120000 in Replica 5

[SENT] insert 101 Shruthi SoftwareEngineer 120000 in Replica 6

AvailableReplicas =3

[SENT] insert 102 Shruthi SoftwareEngineer 120000 in Replica 5

[SENT] insert 102 Shruthi SoftwareEngineer 120000 in Replica 6

[SENT] insert 102 Shruthi SoftwareEngineer 120000 in Replica 0

AvailableReplicas =3

[SENT] insert 103 Shruthi SoftwareEngineer 120000 in Replica 6

[SENT] insert 103 Shruthi SoftwareEngineer 120000 in Replica 0

[SENT] insert 103 Shruthi SoftwareEngineer 120000 in Replica 1

**AvailableReplicas =2**

**[SENT] insert 104 Shruthi SoftwareEngineer 120000 in Replica 1**

**[SENT] insert 104 Shruthi SoftwareEngineer 120000 in Replica 2**

### TestCase-5: Ensuring when a client wishes to update an object into the data repository, it should be able to successfully perform the operation on at least two, and if possible all the three servers that are required to store the object.

$ java aosproject 1 update ProjectHead 10 7

AvailableReplicas =3

ACKNOWLEDGEMENT RECEIVED GOAHEAD

[SENT] update 101 Shruthi ProjectHead 200000 in Replica 4

[SENT] update 101 Shruthi ProjectHead 200000 in Replica 5

[SENT] update 101 Shruthi ProjectHead 200000 in Replica 6

**AvailableReplicas =2**

**ACKNOWLEDGEMENT RECEIVED GOAHEAD**

**[SENT] update 102 Shruthi ProjectHead 200000 in Replica 5**

**[SENT] update 102 Shruthi ProjectHead 200000 in Replica 6**

**AvailableReplicas =2**

**ACKNOWLEDGEMENT RECEIVED GOAHEAD**

**[SENT] update 103 Shruthi ProjectHead 200000 in Replica 6**

**[SENT] update 103 Shruthi ProjectHead 200000 in Replica 1**

### TestCase-6: Ensuring when a client wishes to update an object into the data repository, it should fail the operation if less than 2 replicas available.

$ java aosproject 1 update ProjectHead 10 7

AvailableReplicas =3

ACKNOWLEDGEMENT RECEIVED GOAHEAD

[SENT] update 101 Shruthi ProjectHead 200000 in Replica 4

[SENT] update 101 Shruthi ProjectHead 200000 in Replica 5

[SENT] update 101 Shruthi ProjectHead 200000 in Replica 6

AvailableReplicas =2

ACKNOWLEDGEMENT RECEIVED GOAHEAD

[SENT] update 102 Shruthi ProjectHead 200000 in Replica 5

[SENT] update 102 Shruthi ProjectHead 200000 in Replica 6

ACKNOWLEDGEMENT RECEIVED GOAHEAD

[SENT] update 114 Shruthi ProjectHead 200000 in Replica 4

[SENT] update 114 Shruthi ProjectHead 200000 in Replica 5

**AvailableReplicas =1**

**Cannot Perform Update because lessthan 2 replicas available**

### TestCase-7: Ensuring if two clients try to concurrently write to the same object and at least two replicas are available, the two writes must be performed in the same order at all the replicas of the object.

Performed the Update on Server using 3 clients, Noticed that Master Replicas at each server controlling the clients to update an object on slave replicas.

# Works Cited

*Data Centers - Google*. (n.d.). Retrieved from http://www.google.co.in/about/datacenters/efficiency/internal/#servers

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., . . . Vogels, W. (2007). Dynamo: Amazon's Highly Avaialable Key-value Store. ACM Symposium on Operating Systems Principles (SOSP).

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google File System. ACM symposium on Operating Systems Principle (SOSP), 2003.