Probabilistic CAP implementation for Latency and Consistency SLAs

Project Workbook

By

Ambika Bohra (ambika.bohra@sjsu.edu)

Dhanashree Gaonkar(dhanashree.gaonkar@sjsu.edu)

Lavanya Kandukuri (lavanya.kandukuri@sjsu.edu)

Nikhila Galala (nikhila.galala@sjsu.edu)

07/27/2018

Advisor: Prof. Rakesh Ranjan

Table of Contents

Chapter 1. Literature Search, State of the Art

1.1 Literature Search (Ambika and Lavanya)

The CAP theorem was firstly introduced by Eric Brewer (2000) as a conjecture in a conference keynote. Brewer postulated that three properties (Consistency, Availability, and Partition-tolerance) are desirable from a real-world web service. A distributed system can pick at most two of these three desirable properties. Later, Seth Gilbert and Nancy Lynch (2002) demonstrated this theorem. Various distributed storage systems design is influenced by the CAP theorem. Abadi, D. (2010) stated in a blog that "CAP falls far short of giving a complete picture of the engineering tradeoffs behind building scalable, distributed systems". In reality, partition-tolerance is inevitable, so the system has to choose between CP and AP during a network partition.

In a recent research, Abadi, D. (2012) proposed a new formulation - PACELC, that helps to understand the trade-off while designing a Distributed Database System (DDBS). He stated that CAP theorem does not suffice for modern applications and NoSQL databases such as Cassandra and Riak. In a DDBS, when data is replicated to different servers, the actual trade-off between latency and consistency occurs. Cassandra and Riak give up on consistency for lower latency by default. Moreover, most KV stores offer only weak consistency (eventual consistency). Rao J., Shekita E., Tata S (2011) analyzed two options i.e. weak-read and quorum read for Cassandra's latency-consistency trade-off. These options increase consistency but at the cost of system's latency. The PACELC provides a user-adjustable setting to modify the ELC trade-off. So, increased consistency can be achieved by increasing R+W at the cost of latency (Abadi, D., 2012).

Rahman, Tseng, Nguyen, Gupta & Vaidya (2017) proposed a probabilistic CAP system where applications can either choose a latency SLA or a consistency SLA. It is a qualitative characterization of the CAP trade-off. The PCAP system can be connected to key-value stores

such as Cassandra and Riak. A probabilistic latency SLA can benefit the applications that expect freshness. On the other hand, a known use case of the probabilistic consistency SLA is a web-search application.

For the research purpose, we aim to implement the “Probabilistic CAP” system proposed by Rahman., et. al (2017).

1.2 State-of-the-Art Summary (Dhanashree)

The leading-edge tools and techniques used would be:

1.2.1 Virtual Nodes

EC2 instances:

A virtual server deployed in Amazon's Elastic Compute Cloud for providing computing capabilities on Amazon Web Services (AWS) infrastructure. These instances will act as additional nodes in the network.

Docker Containers:

Virtual servers can be set up using the latest container technology. Docker containers provide lightweight, stand-alone software packages that can be deployed independently from the underlying environment. This project will use Docker containers to push the code inside virtual and physical nodes.

1.2.2 Message Exchange

Protobuf:

A message exchange mechanism that is language-neutral and platform-neutral. Protobuf can be validated by the nodes that are responsible for generating and consuming these messages. Protobuf will be used in the project for defining all the messages that will be exchanged during the algorithm implementation.

1.2.3 API Gateway

KONG:

Kong is an API gateway. That means it is a form of middleware between computing clients and API-based applications. Kong easily and consistently extends the features of any APIs. Kong will be used to visualize, inspect and monitor APIs.

1.2.4 Workload Simulation

YCSB:

The Yahoo! Cloud Serving Benchmark (YCSB) is an open-source specification and program suite for evaluating retrieval and maintenance capabilities of computer programs. It is often used to compare the relative performance of NoSQL database management systems.

1.2.5 No-SQL Database

Cassandra:

It is a NoSQL database designed for distributed data storage. Cassandra has high availability and has a columnar structure for data storage. The architectural design of Cassandra is a mixture of Dynamo and HBase. It supports eventual consistency and thus in event of network partition favors availability over consistency.

Cassandra supports replication amongst the cluster. REST calls can be made to Cassandra interface for data access. It has a decentralized cluster management design like Dynamo where every node has a similar task allotment and there is no master-slave configuration. Cassandra is incrementally scalable like Dynamo where nodes can be added or deleted without the need of manual redistribution. Cassandra supports super column strategy wherein it can store multiple data structures in its values. Data can be queried based on the key as it is stored in the form of a key-value store.

Riak:

Riak is also a NoSQL database inspired from the Dynamo design. Its data reconciliation model is designed in such a way that the services consuming Riak have the freedom to a trade-off between consistency or availability depending on the service level agreement with the customers. Riak by default is eventually consistent but can be configured to be strongly consistent during its read/ write operations. Riak has a master-less peer to peer architecture where each node in a Riak cluster is the same. This uniform nature forms the basis of Riak's fault tolerance and scalability. Riak has a quorum approach by default in its querying model. Riak is used for distributed data storage systems and is originally developed in Erlang. For data object validation and versioning, vector clocks are used in the Riak design. Riak API is exposed in the form of REST web services and can be consumed by making HTTP calls to these endpoints.

1.2.6 RPC Framework

gRPC:

gRPC is a cutting-edge high-performance RPC framework which supports HTTP/2.0 which is a major revision on current HTTP protocol adopted by world wide web. gRPC can be used in any environment and has readily available support for SDKs in C++, Java, Python, Go, Node.js. It can efficiently connect services in and across data centers with pluggable support for load balancing, health checking, and authentication. This project will implement gRPC for message passing between the PCAP system and the database clusters.

REST:

Representational State Transfer has inspired both the W3C Technical Architecture Group's architecture document [Web Arch] and many who see it as a model for how to build Web services. REST interactions are "stateless" in the sense that the meaning of a message does not depend on the state of the conversation. Thus, applications using REST endpoints are easily scalable and fault tolerant.

1.2.7 Data visualization using Charts

Google Charts:

Google Charts is one of the most popular JavaScript charting library which can render interactive and animated charts when supplied with appropriate data. Data visualization dashboards usually use Google Charts for data rendering. It has all the basic charting APIs required like- bar chart, line chart, bubble chart, etc. We would be using Google charts on our dashboard to show the performance based on the specified SLA for a target system.

1.3 References (Ambika)

[1] Brewer, E. A. (2000). Towards robust distributed systems (abstract). *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing - PODC '00*. doi:10.1145/343477.343502.

[2] Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, *33*(2), 51. doi:10.1145/564585.564601.

[3] Fox, A., & Brewer, E. (1999). Harvest, yield, and scalable tolerant systems. *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*. p 174–178, doi:10.1109/hotos.1999.798396.

[4] Brewer, E. (2010). A certain freedom: thoughts on the CAP theorem. *Proceeding of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing - PODC '10*. doi:10.1145/1835698.1835701.

[5] Abadi, D. (2010). Problems with CAP, and Yahoo's little known NoSQL system [Blog Post]. Retrieved from http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html.

[6] Abadi, D. (2012). Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. *Computer*, *45*(2), 37-42. doi:10.1109/mc.2012.33.

[7] Brewer, E. (2012). CAP twelve years later: How the "rules" have changed. *Computer*, *45*(2), 23-29. doi:10.1109/mc.2012.37.

[8] Bailis P., & Ghodsi A. (2013). Eventual Consistency Today: Limitations, Extensions, and Beyond. *Queue*, *11*(3), 20. doi:10.1145/2460276.2462076.

[9] Lakshman A., and Malik P. (2009). Cassandra: Structured Storage System on a P2P Network, *Proc. 28th ACM Symp. Principles of Distributed Computing (PODC 09)*, ACM, article no. 5; doi:10.1145/1582716.1582722.

[10] Rao, J., Shekita, E. J., & Tata, S. (2011). Using Paxos to build a scalable, consistent, and highly available datastore. *Proceedings of the VLDB Endowment*, *4*(4), 243-254. doi:10.14778/1938545.1938549.

[11] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, & I. Stoica. Probabilistically bounded staleness for practical partial quorums. *Proc. VLDB Endowment*, 5(8):776–787, 2012.

[12] Rahman, M. R., Tseng, L., Nguyen, S., Gupta, I., & Vaidya, N. (2017). Characterizing and Adapting the Consistency-Latency Tradeoff in Distributed Key-Value Stores. *ACM Transactions on Autonomous and Adaptive Systems*, *11*(4), 1-36. doi:10.1145/2997654.

[13] Terry, D. B., Prabhakaran, V., Kotla, R., Balakrishnan, M., Aguilera, M. K., & Abu-Libdeh, H. (2013). Consistency-based service level agreements for cloud storage. *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles - SOSP '13*. p. 309–324, doi:10.1145/2517349.2522731.

[14] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010). Benchmarking cloud serving systems with YCSB. *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*. doi:10.1145/1807128.1807152.

[15] Apache Cassandra. (n.d.). Retrieved from http://cassandra.apache.org/.

Chapter 2. Project Justification (Lavanya)

2.1 Background

CAP theorem states that a system can choose at most two of the three properties: Consistency (C), Availability/Latency (A) and Partition tolerance (P). When a network partition occurs, one needs to choose between consistency and availability or consistency and latency. Applications like Amazon and Google require low latency without which there could be heavy revenue loss even for a millisecond delay. Users expect fast responses and their satisfaction plays an important role. Customers, in this case, must compromise on consistency as these systems provide eventual consistency. At the same time, clients of applications like Netflix desire freshness, i.e., they expect latest data. For instance, Netflix users desire accuracy in video tracking, i.e., freshness in data returned. So, even if it is eventual consistency, clients expect that to be time-bound.

2.2 Objective

Our objective is to implement probabilistic models for Consistency, Availability, Partition tolerance to address problems with key-value stores pertaining to soft partitions in data centers. These models help in optimizing consistency when low latency is chosen and optimizing latency when consistency is chosen. Abadi, D. (2012) proposed a new formulation - PACELC, that helps to understand the trade-off while designing a Distributed Database System (DDBS). Rao J., Shekita E., Tata S (2011) analyzed two options i.e. weak-read and quorum read for Cassandra's latency-consistency trade-off. These options increase consistency, but at the cost of system's latency. In this project, we are implementing an algorithm proposed by Rahman, Tseng, Nguyen, Gupta & Vaidya (2017) which focuses on probabilistic CAP system to provide latency SLA and consistency SLA, which is integrated to Cassandra and Riak. If this algorithm is implemented successfully, one could achieve optimized consistency when low latency is chosen and optimized latency when consistency is chosen which leads to higher user satisfaction and ultimately higher revenue.

Chapter 3. Project Requirements (Nikhila)

3.1 Functional Requirements

1. gRPC is used for communication between nodes in the cluster. To achieve high throughput, low latency messaging gRPC is chosen over other messaging systems.
2. PCAP system is implemented on NoSQL key value store. Apache Cassandra database is chosen for the project which follows peer to peer communication. Yahoo cloud Serving Benchmarking connector for Cassandra is also available for performance testing.
3. Amazon Web Services Cloud is used to set up the cluster. This provides features like Load Balancing and Auto Scaling which is necessary for any distributed system.
4. Kong API gateway is used to configure different endpoints of the project. The features of Kong such as authentication, security, traffic control can be leveraged for the project.
5. Docker is used to build image for application being developed which can be executed in a containerized platform like kubernetes.
6. Yahoo Cloud Serving Benchmark (YCSB) is used to evaluate the performance of the system. Performance for different cluster size and traffic loads are measured using YCSB and reports are generated.

3.2 Non-Functional Requirements

3.2.1 Fault Tolerance

The Probabilistic CAP system being developed should be fault tolerant. It should handle the scenarios of node failure without affecting availability and consistency. The testing also includes node failure scenarios to ensure the system is fault tolerant.

3.2.2 Testability

We will use Yahoo Cloud Serving Benchmark to test the performance of the system. We can show analytics on how PCAP system improves the consistency and availability with incoming traffic and cluster size. We can compare the performance of the system as we increase cluster size using these results.

3.2.3 Scalability

Scalability is very important for any distributed system. Initially, we want to build a build a PCAP system small cluster and later scale it to the larger cluster to ensure handle high traffic. We also want to configure use nodes in AWS at different geographical locations and analyze the system performance.

3.2.4 Maintainability

The PCAP system being developed has a layered architecture. So the code is divided according to these layers and made modular. There should be no cascading changes. The cluster size and the database used are configurable.

3.2.5 Documentation

We are using Cassandra as NoSQL data store for the project. Apache Cassandra documentation is followed for implementation. For setting up the infrastructure, we are following AWS documentation. Yahoo Cloud Serving Benchmarking is used for performance testing.

Chapter 4. Dependencies and Deliverables (Ambika)

4.1 Dependencies

4.1.1 Computing resources for dense clusters

We have planned to deploy a medium density cluster using AWS EC2 and Docker container. We will create a development environment for our local machines for small clusters and later use EC2 for the final prototype.

4.1.2 Inter-node communication in cluster architecture

It is essential that two nodes in a cluster can communicate with each other. We have planned to accomplish this by using gRPC. With this, any two nodes can talk directly. Next step is to make sure that nodes in different clusters must go through respective master nodes while nodes in the same cluster can interact directly.

4.1.3 Read/Write Workload Simulation

For workload simulation, we have planned to use YCSB (Yahoo! Cloud Serving Benchmark) framework since it consists of a workload generating client and a set of standard workloads like read-heavy workloads and write-heavy workloads. The main advantage of using YCSB is that it can be extended to define new workload types.

4.1.4 Evaluation of performance

We need to deploy our system on AWS Cloud Server. Cloud serving systems are insufficient for apples-to-apples performance observation, which makes it challenging to learn the tradeoff between systems and the workloads. So, we have planned to use YCSB to facilitate the performance evaluation of our PCAP system.

4.1.5 Centralized Authentication using Kong API gateway

For access control and authorization and management of our API, we have planned to use Kong API gateway, which supports the Cassandra database.

4.2 Deliverables

By the end of the semester, we will have a working prototype of the Probabilistic CAP system as a proof of concept. It includes:

1. Network topology and database cluster configuration
2. Probabilistic CAP algorithm implementation
3. Containerization of applications
4. Workload simulation using benchmarking technologies
5. System performance evaluation and data visualization
6. Code review and Documentation

Chapter 5. Project Architecture (Dhanashree and Nikhila)

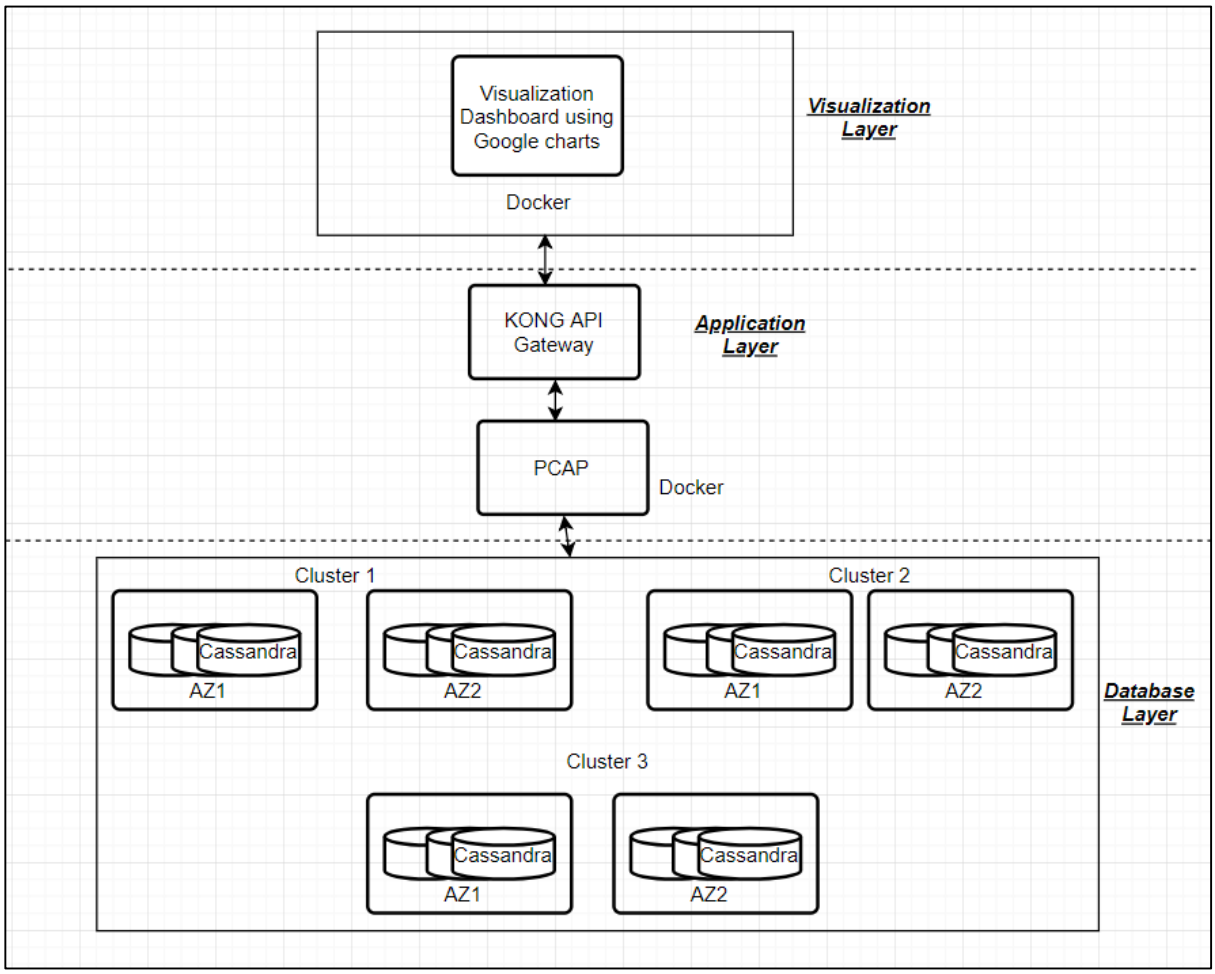


Fig.1 Layered architecture diagram

PCAP is a system designed for distributed data stores that adapts automatically in real-time under dynamic network to meet the service level agreement (SLA) while optimizing Consistency/ Availability parameter. The implementation of the algorithm proposed by Rahman., et al (2017) involves designing an ecosystem that will contain clusters of NoSQL databases. Each cluster will have nodes existing in multiple availability zones. The database cluster will be configured to allow data replication across the nodes. Implementation of PCAP will be present in the application layer. All the applications will be containerized using Docker for easy deployment and execution.

Every key-value store provides a read/write API over an asynchronous distributed message-passing network. It has a client-server architecture where data replication is done by the server; it is responsible to keep the data store in a consistent state. Clients, on the other hand, are responsible for the operations (read/ write). PCAP will form an application layer on top of this data storage system. The purpose of a PCAP system is to attain performance close to the unachievable envelope / theoretical consistency-latency tradeoff envelope as mentioned by Rahman., et. al (2017).

YCSB will be used on top of our application layer to evaluate the performance (workload simulation) of the NoSQL database used in the experiment. YCSB adapter for Cassandra is readily available, but if we choose to use Riak for our experiments we will have to develop our own YCSB adapter for the same.

There are few assumptions that are made about the underlying key-value store which are as follows:

1. Every key is being replicated across the cluster on multiple nodes.
2. There exists a master/coordinator node that manages client queries.
3. The existence of a read repair mechanism for reconciling of divergent replicas (eventual consistency).
4. Clocks on all the node servers are synchronized so that global timestamps can be used to detect stale data.

We will be dealing with two types of SLAs to implement PCAP system. A latency SLA meets the desired time deadline for read operations while maximizing the chance of getting fresh results (minimize staleness). An example application using this SLA would be the Amazon shopping cart, which doesn't want to lose customers due to high latency.

A consistency SLA meets the desired freshness probability while maximizing the chance of receiving the read result within a stipulated time. An example application using this SLA would be Google/Twitter search, where users want to receive ‘latest’ data as search results.

Data visualization for the performance of the PCAP system would be done through the Google Charts dashboard. Kong API gateway will be used to access the underlying PCAP system (application layer) APIs and visualize the data.

Chapter 6. Project Design

6.1 Sequence Diagrams:

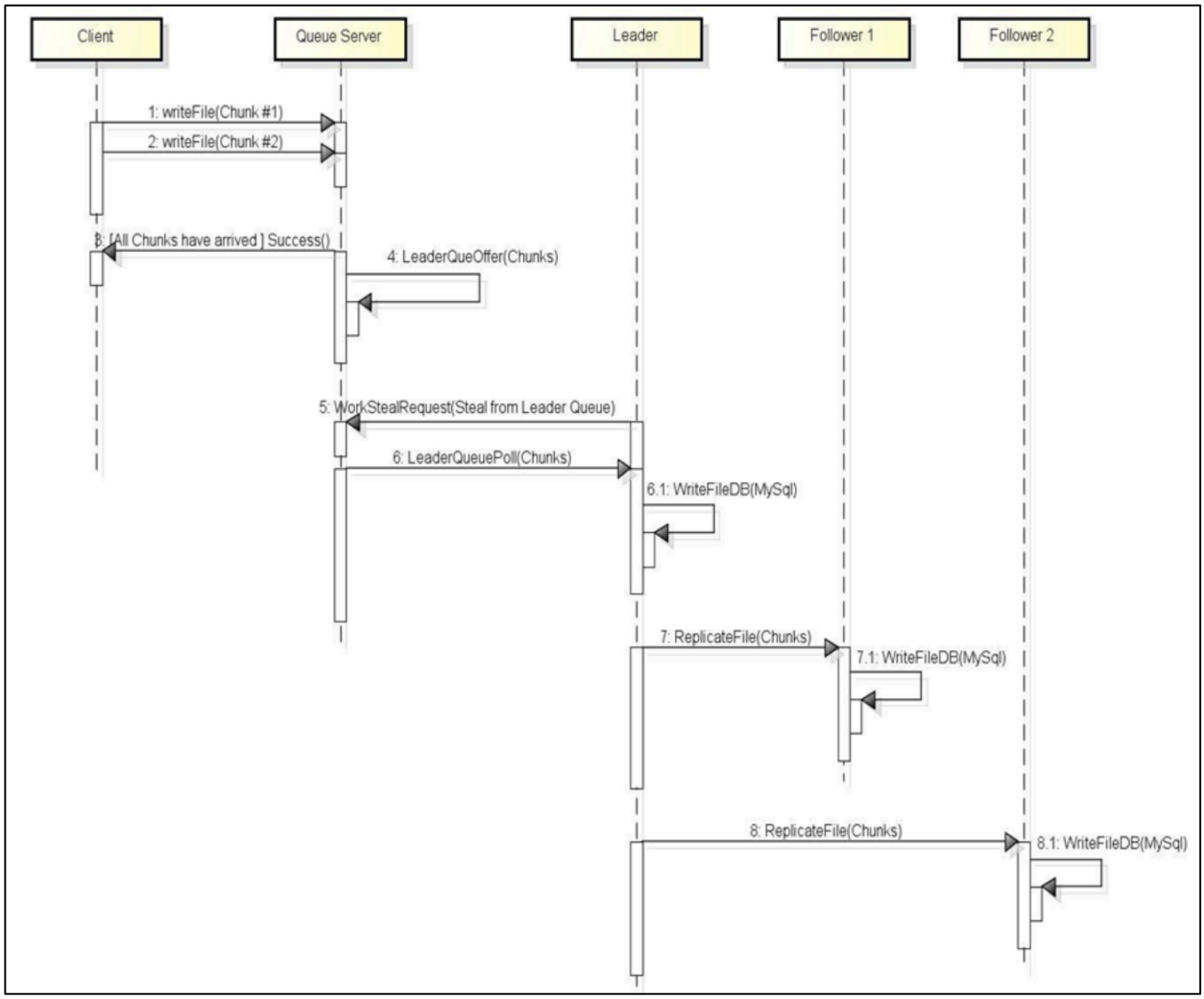


Fig.2 Sequence Diagram for Write

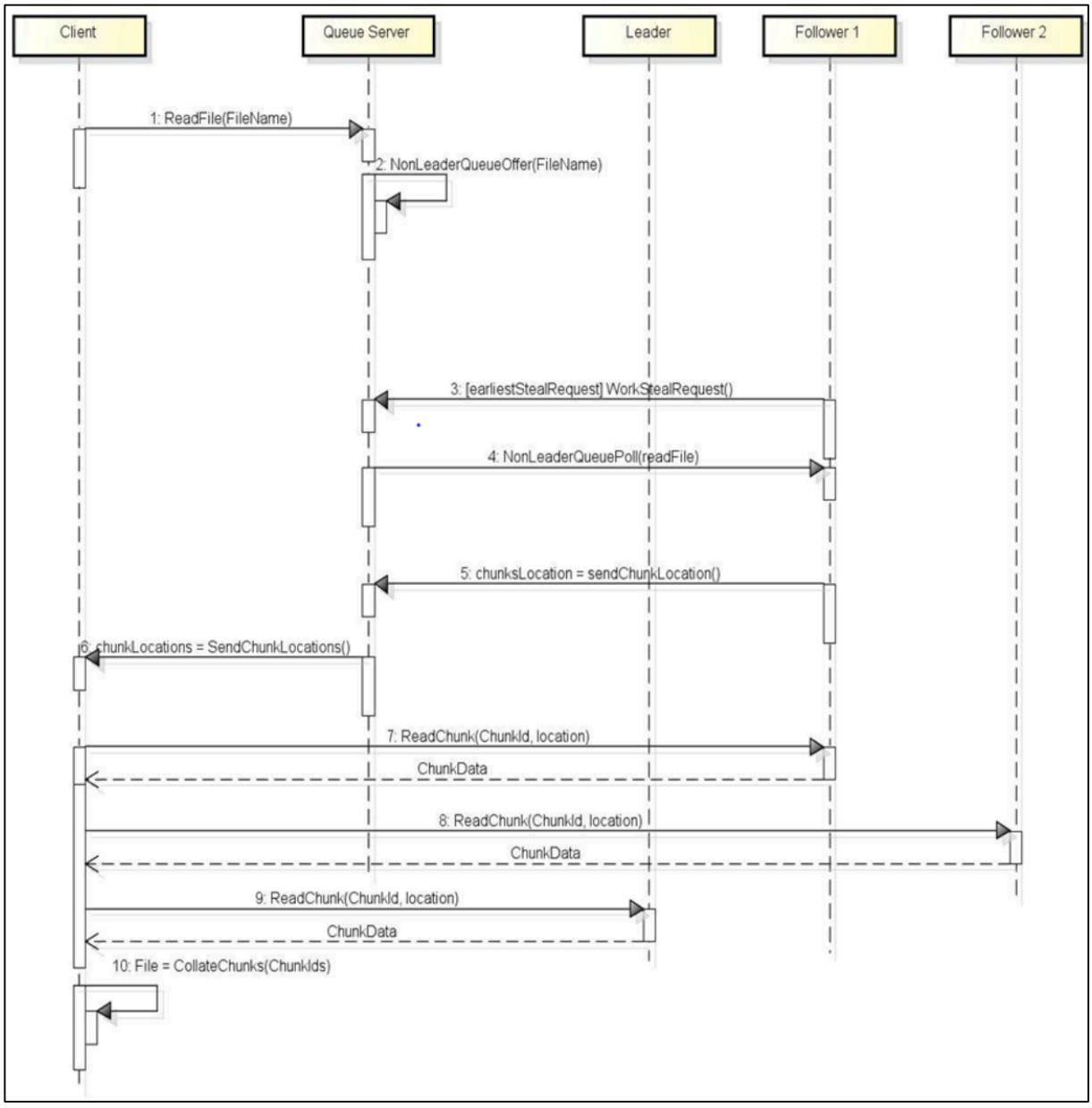


Fig. 3 Sequence Diagram for Read

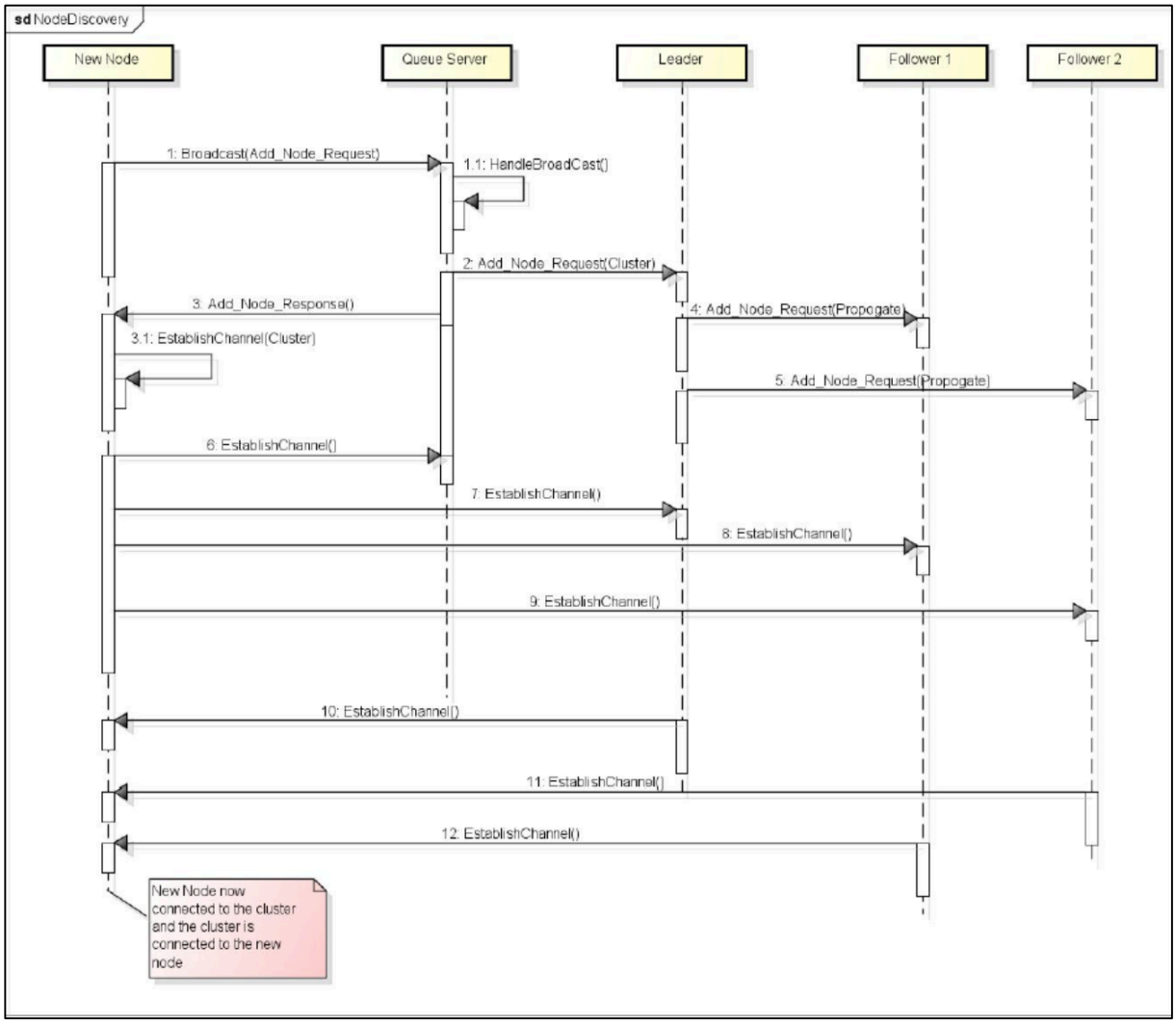


Fig. 4 Sequence Diagram for Node

6.2 Design diagram and Prototype Implementation:

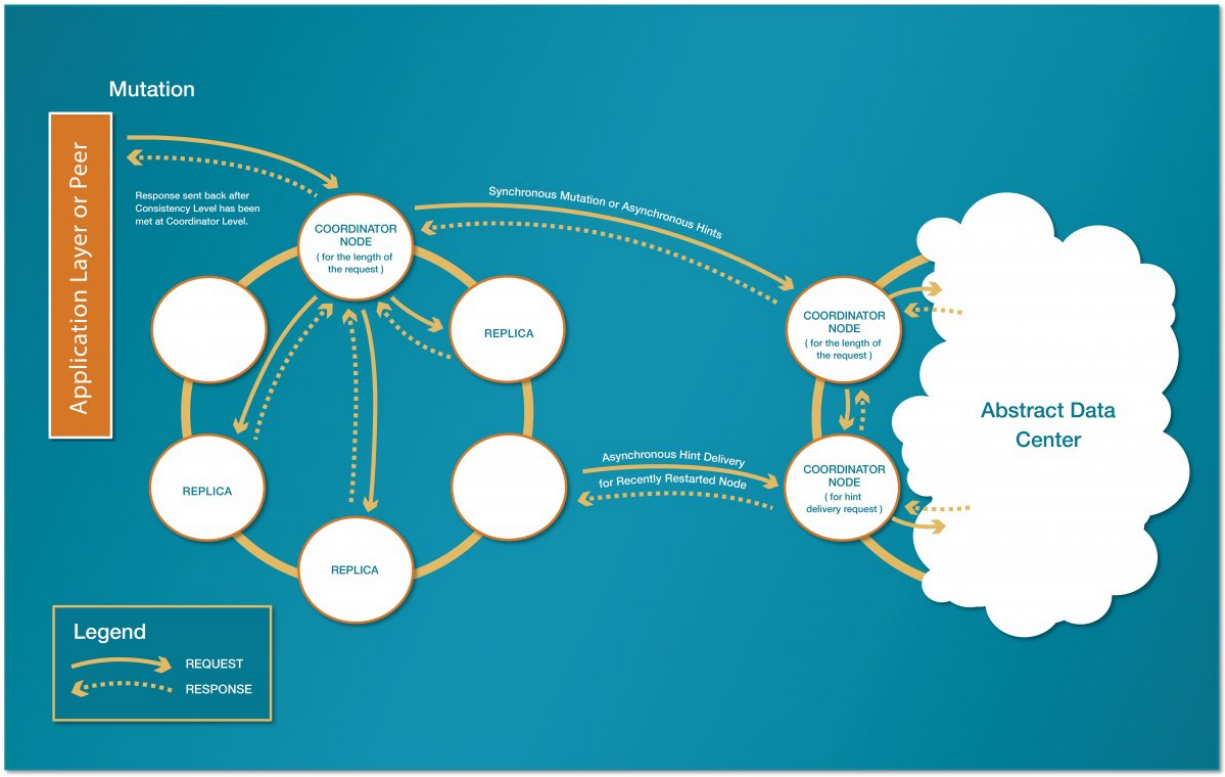


Fig.5 Project Design diagram

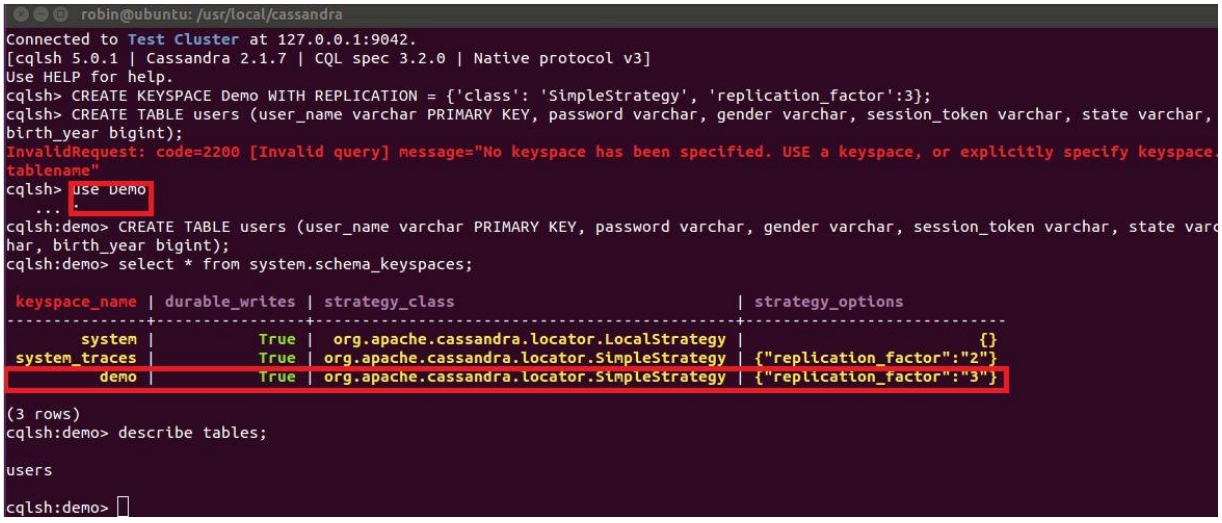
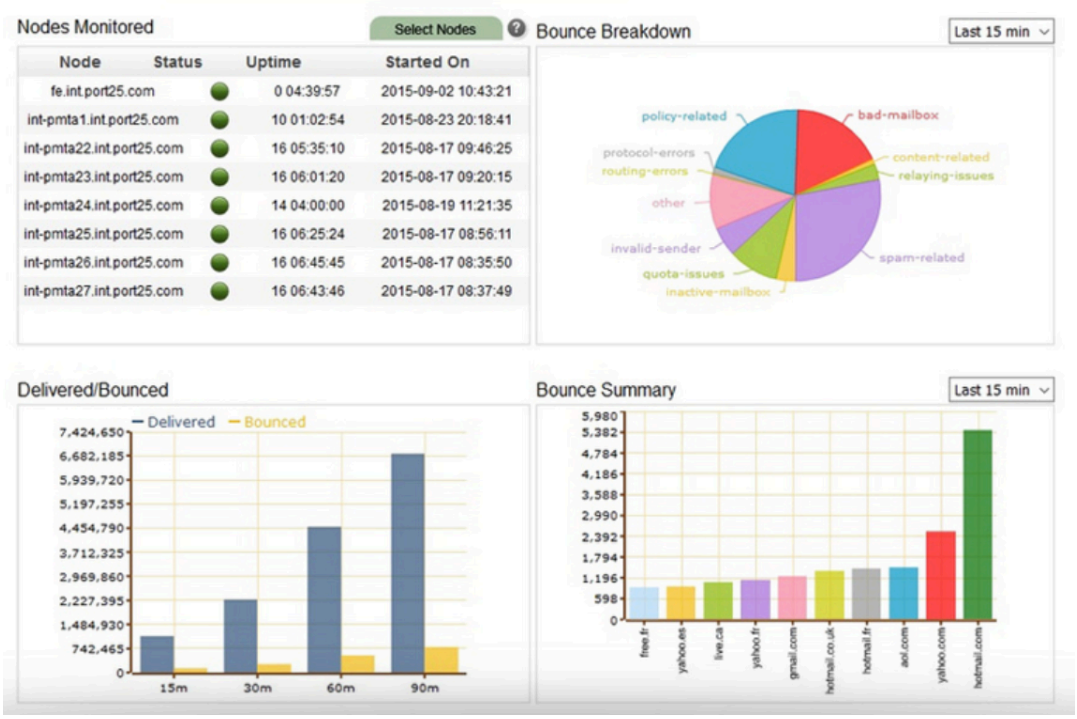


Fig. 6 Prototype implementation for Cassandra

6.3 UI Mock ups:



Chapter 7. QA, Performance, Deployment Plan

7.1 Quality Assurance

The testing procedure is followed at each stage of the project. Unit testing, Integration testing and System testing are followed in the project. The project is divided into modules and each module is handled each Team member.

Unit Testing: Each module is developed and tested for functionality. Shell scripts are written to check read, write and update operation on the data stored in database. Following unit testing would minimize errors at further stages.

Integration Testing: Modules are integrated after the implementation of each module. This is done stage by stage. During the integration, the interaction between the modules and functionality is tested. This is done for seamless interaction between the modules as per design.

System Testing: After the completion of implementation of PCAP on Cassandra cluster, the entire system is tested for the success scenarios of database transactions like read, write, update and delete. Scripts are written to check the many transactions on the database cluster.

Performance Testing: The performance of the PCAP implementation needs to be compared with the existing CAP approach. Yahoo Cloud Serving Data (YCSB) is a framework that is used to benchmark he NOSQL stores. The latency and throughput parameters are tested using this framework for different set of operations.

YCSB core framework has predefined workloads. Each workload has set of transactions for read, write, update operations. There are workloads that are specific for heavy read, heavy write, heavy read and write operation. Each workload has two phases: Load phase and Transaction phase.

In the load phase, the data is loaded to perform operations in the next stage. In the transaction phase, the database operations are performed on the data as per the workload. The parameter

file is loaded in both the stages. It contains the configuration needed for benchmarking. For example, the number of the client threads used for workload operations, the number of operations that have to be executed per second are configured in the parameter file.

At the end of operation, the throughput, average latency is shown in the results. These results are used for evaluation of PCAP algorithm and are compared with existing CAP approach.

Load Testing: The YCSB framework also supports clients to define their own user defined workloads. For Load testing the number of operations count can be increased in the parameters file of existing core workloads or new workloads could be defined. com.yahoo.ycsb.DB package is used to create new workloads. We are planning to implement workloads for read, write and update operations to test the PCAP implementation exhaustively. Configurations needed for new workload must be defined in parameter file. These parameters are loaded before the execution of the workload. The data that must be inserted is defined in load method and the operations that must be performed are defined in the transactions method. The count that these operations have to be repeated can defined as million in parameter file to perform the load test.

7.2 Test Plan and Schedule:

Test Type	Test Scope	Target Date
Unit Testing	Each team member is responsible for unit testing their code as and when development is done	Throughout the project lifecycle
Integration Testing	Integrating all the components one by one at the end. Testing modules for any dependencies.	Throughout the project lifecycle

Functional Testing	Creating network topology and test node connectivity	08/30/2018
	Generate workload using YCSB	09/01/2018
	PCAP System implementation	09/25/2018
	Connect Cassandra to YCSB	10/08/2018
	End to end flow with UI	10/25/2018
Performance Testing	Test if the entire application is running as expected and if consistency and latency SLAs are fine-tuned depending on the requirements	10/30/2018

7.3 Performance Evaluation:

PCAP implementation is done to achieve latency and consistency trade-off during dynamic network conditions. We need to evaluate the performance of the system implemented and compare with existing CAP approach. The metrics like throughput, latency and execution time help in performance evaluation. YCSB framework is used to evaluate the performance of the system. As mention above, YCSB has predefined workloads and supports user define workloads. These workloads help if performing database transactions and evaluate the latency and throughput. These statistics are compared with CAP results and plotted on Google charts for analyzing the performance of PCAP algorithm.

7.4 Deployment Plan:

Task	Target Date
Implementation of individual modules	10/05/2018
AWS deployment	10/25/2018
testing of individual modules on AWS	11/02/2018
Integration of all the modules	11/16/2018
Testing of the integrated application on AWS	11/23/2018
Regression testing	12/05/2018

Chapter 8. Implementation Plan and Progress (Dhanashree and Nikhila)

8.1 Environment Setup

1. The Project needs a cluster of nodes communicating with each other. AWS cloud services are used for the infrastructure.
2. Python3 and JAVA are chosen as programming languages for development which are presently inbuilt in AWS EC2 instances.
3. Clusters of Cassandra database nodes are configured. Each Cluster has a coordinator which acts as a proxy between the application and the nodes or the replicas.
4. The coordinator determines which nodes in the cluster should get the read and write requests based on the configurations.
5. gRPC is installed in the nodes which is used for messaging between the nodes.

8.2 Tools and Frameworks Used for Development

1. gRPC/REST will be used for asynchronous messaging between the nodes to handle high traffic with low latency.
2. Docker is used to containerize the application development and can be executed in a containerized platform like Kubernetes.
3. Kong API gateway is used to configure the endpoints of the REST calls of the application.
4. Cassandra NoSQL database is chosen as the key-value store for the project. Cluster of Cassandra nodes are configured with a master node to manage the client queries.
5. Google Charts is used for visualization of performance statistics of the application developed.

8.3 PCAP Implementation

1. PCAP Application is built above the data store. The application is built as per algorithm proposed by Rahman., et al (2017). The purpose of the implementation is to improve the consistency and latency trade-off.

2. Time-stamps/vector clocks will be used to synchronize all the nodes.

3. Given an application with a consistency or availability SLA, we will implement an adaptive controller to convert the given system into a PCAP system that can meet the specified SLAs.

4. We mainly use a read delay knob, which adds a tunable time delay before any read operation. Controlling this delay allows fine grained trade-off between C and A.

5. Our adaptive controller uses an active approach, where we inject a small number of test operations to estimate current metric values, so that we can tune the control knobs based on how far we are from the SLA. The active approach allows better control on the SLA convergence.

8.4 Performance Evaluation

1. Yahoo Cloud Serving Benchmark(YCSB) is used to evaluate the performance of the PCAP system.

2. The YCSB connector for Cassandra database is available. The performance of Cassandra database with PCAP Implementation is tested with high traffic load.

3. The performance results are visualized using Google charts.

8.5 Progress of the Project

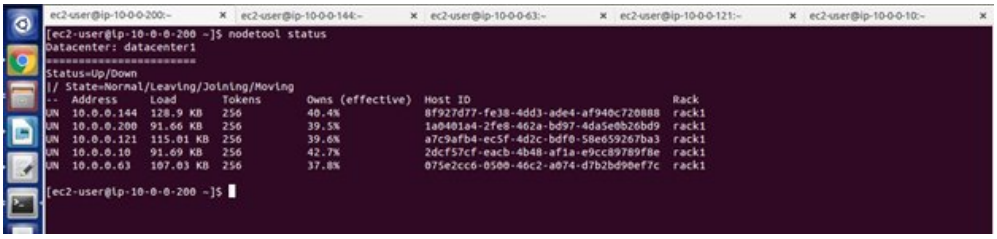
1. We have explored and read research papers (Rahman., et al (2017), Eric Brewer (2000)) and online articles and understood how to implement the PCAP system.

2. We have explored different performance evaluation methods of a distributed system and finalized the Yahoo Cloud Serving Benchmark Technique to evaluate the performance of the PCAP system being developed.

3. We have made the set up for gRPC and implemented messaging between two different application. This can be later integrated to PCAP application.

4. We have explored different NoSQL stores and finalized Cassandra as Key-Value store for the application since the YSCB connector for Cassandra is readily available for performance evaluation.

5. Cassandra cluster is setup to implement PCAP system on top of this NoSQL database. AWS account is setup along with initial EC2 configuration for data node procurement. Cassandra is installed and configured to communicate to all the nodes in the topology. The cluster setup is successful which can be checked with nodetool status command as follows:



Using the default logging library of Log4J present in Cassandra, it has been debugged and various logging levels have been explored. Unit testing is done using CircleCI.

6. YCSB framework is installed and evaluated on Cassandra Cluster. The YCSB core workload operations are executed and the results are collected. Latency and through of the operations of each workload are calculated. The same workloads would be executed after PCAP implementation and the resulted are compared with CAP approach.

```
[OVERALL], RunTime(ms), 5015
[OVERALL], Throughput(ops/sec), 199.40179461615153
[TOTAL_GCS_Copy], Count, 9
[TOTAL_GC_TIME_Copy], Time(ms), 28
[TOTAL_GC_TIME_%_Copy], Time(%), 0.5583250249252244
[TOTAL_GCS_MarkSweepCompact], Count, 0
[TOTAL_GC_TIME_MarkSweepCompact], Time(ms), 0
[TOTAL_GC_TIME_%_MarkSweepCompact], Time(%), 0.0
[TOTAL_GCs], Count, 9
[TOTAL_GC_TIME], Time(ms), 28
[TOTAL_GC_TIME_%], Time(%), 0.5583250249252244
[READ], Operations, 483
[READ], AverageLatency(us), 1450.8592132505175
[READ], MinLatency(us), 523
[READ], MaxLatency(us), 33791
[READ], 95thPercentileLatency(us), 2485
[READ], 99thPercentileLatency(us), 4911
[READ], Return=OK, 483
[CLEANUP], Operations, 1
[CLEANUP], AverageLatency(us), 2233344.0
[CLEANUP], MinLatency(us), 2232320
[CLEANUP], MaxLatency(us), 2234367
[CLEANUP], 95thPercentileLatency(us), 2234367
[CLEANUP], 99thPercentileLatency(us), 2234367
[UPDATE], Operations, 517
[UPDATE], AverageLatency(us), 1359.4390715667312
[UPDATE], MinLatency(us), 522
[UPDATE], MaxLatency(us), 14423
[UPDATE], 95thPercentileLatency(us), 2503
[UPDATE], 99thPercentileLatency(us), 5067
[UPDATE], Return=OK, 517
ubuntu@ip-172-31-22-139:~/ycsb-0.14.0$
```

7. Riak has been downloaded and installed for the cluster setup. Riak as a NoSQL key value store has been explored and after configuring all the basic requirements, cluster setup has been taken up. After adding nodes to a cluster and connecting them, the results on the terminal looked like:

```
================================= Membership ==================================
Status     Ring    Pending    Node
-------------------------------------------------------------------------------
valid     100.0%     20.3%    'riak@192.168.1.10'
valid       0.0%     20.3%    'riak@192.168.1.11'
valid       0.0%     20.3%    'riak@192.168.1.12'
valid       0.0%     20.3%    'riak@192.168.1.13'
valid       0.0%     18.8%    'riak@192.168.1.14'
-------------------------------------------------------------------------------
Valid:5 / Leaving:0 / Exiting:0 / Joining:0 / Down:0
```

Apart from this, a cluster has been set up on AWS by creating EC2 instances and connecting them using SSH. The running instances can be shown as follows:

☐	i-b86aad44	t1.micro	us-east-1a	running	Initializing	None	ec2-54-152-201-100.co...
☐	i-bb6aad47	t1.micro	us-east-1a	running	Initializing	None	ec2-52-1-145-42.compu...
☐	i-bf6aad43	t1.micro	us-east-1a	running	Initializing	None	ec2-54-174-16-143.com...
☐	i-c96aad35	t1.micro	us-east-1a	running	Initializing	None	ec2-54-173-232-139.co...
☑	i-ca6aad36	t1.micro	us-east-1a	running	Initializing	None	ec2-54-173-253-77.com...

8. Google charts API provides a rich variety of charts from line charts to complex hierarchical tree maps. The interactive charts are based on SVG/HTML technologies to give a cross-browser compatibility and cross-platform portability to other devices. Certain libraries are required to be loaded for using google charts. Firstly, a javascript file (loader.js) is loaded in the HTML file. Then core chart package API is loaded using google.charts.load method. By configuring the Google Visualization API, dashboard and charts can be drawn. It provides various accessing methods to populate any type of data. Data is wrapped in a JavaScript class called google.visualization.DataTable, which is defined in the Google Visualization library. Data table contains the data for the chart. Columns of data table represent the legends and rows represent the corresponding data. A JSON object is passed as data to initialize the data table. We would get the data from the backend as a JSON object and pass it to data table. Then, we would setup configuration parameters to create the dashboard and make charts for statistics visualization.

8.6 High Level Implementation plan and Progress:

S.no	Task Item	Sub Task Item	Percentage Complete	Resources
1	Project plan		100%	
		Project scope and Abstract	100%	All
		Project scheduling	100%	All
2	Project Requirements		100%	

		Understanding technical requirements	100%	All
		Understanding functional requirements	100%	All
3	Project Design		100%	
		Consolidated requirements	100%	All
		Overall architecture design	100%	All
		Detailed architecture design	100%	All
4	Implementation Part A (Environment Setup and testing)		100%	
		Cassandra Setup and Testing	100%	Dhanashree
		Riak Setup and Testing	100%	Lavanya
		YCSB Setup and Testing	100%	Nikhila
		Google Chart Setup	100%	Ambika
5	Implementation Part B (Algorithm implementation, data visualization and testing)		0%	
		Derive probabilistic models for tradeoff between consistency and Latency	0%	All

		Algorithm implementation on Cassandra and Riak	0%	All
		Visualize the tradeoff parameters and analyze the results	0%	All
6	Performance Evaluation		30%	
		YCSB framework study (Part A)	100%	All
		YCSB benchmarking on Cassandra cluster (Part A)	100%	Nikhila
		YCSB implementation of New Workloads (Part B)	0%	All
		YCSB performance evaluation on PCAP implementation (Part B)	0%	All
7	Testing		25%	
		Prepare Test and Deployment Plan	100%	All
		Acceptance Test plan	0%	All
		End to End Testing	0%	All
8	Report and Presentation		40%	
		Workbook1	100%	All

		Workbook2	100%	All
		Final Report and Presentation	0%	All

Chapter 9. Project Schedule (Ambika and Lavanya)

Phase	Task Name	Start Date	End Date	Participant
Literature Search	Find research and study materials related to CAP	6/8/18	6/15/18	Ambika, Lavanya
	Identifying the existing solutions and checking feasibility of proposed solutions.	6/15/18	6/21/18	Ambika, Nikhila
	Research the tools and technologies to be used	6/21/18	7/5/18	Ambika, Dhanashree, Lavanya, Nikhila
	Read research paper and base implementation of Rahman, M.R. (2017)	6/25/18	7/5/18	Ambika, Dhanashree, Lavanya, Nikhila
Environment Setup	Find relevant development tools and simulation environments	7/6/18	7/11/18	Dhanashree, Nikhila
	Install required development tools	7/12/18	7/20/18	Ambika, Dhanashree, Lavanya, Nikhila

	Cluster setup	7/18/18	7/23/18	Ambika, Lavanya,
	YCSB setup	7/20/18	7/25/18	Dhanashree, Lavanya
Prototype Implementation	Create Network topology and cluster configuration	7/25/18	7/28/18	Ambika and Nikhila
	Test cluster nodes connectivity	7/28/18	7/30/18	Dhanashree, Lavanya
	Basic implementation of the PCAP system	7/31/18	8/5/18	Ambika, Dhanashree, Lavanya, Nikhila
	Identifying test cases	8/5/18	8/9/18	Dhanashree, Nikhila

9.1 Gantt Chart

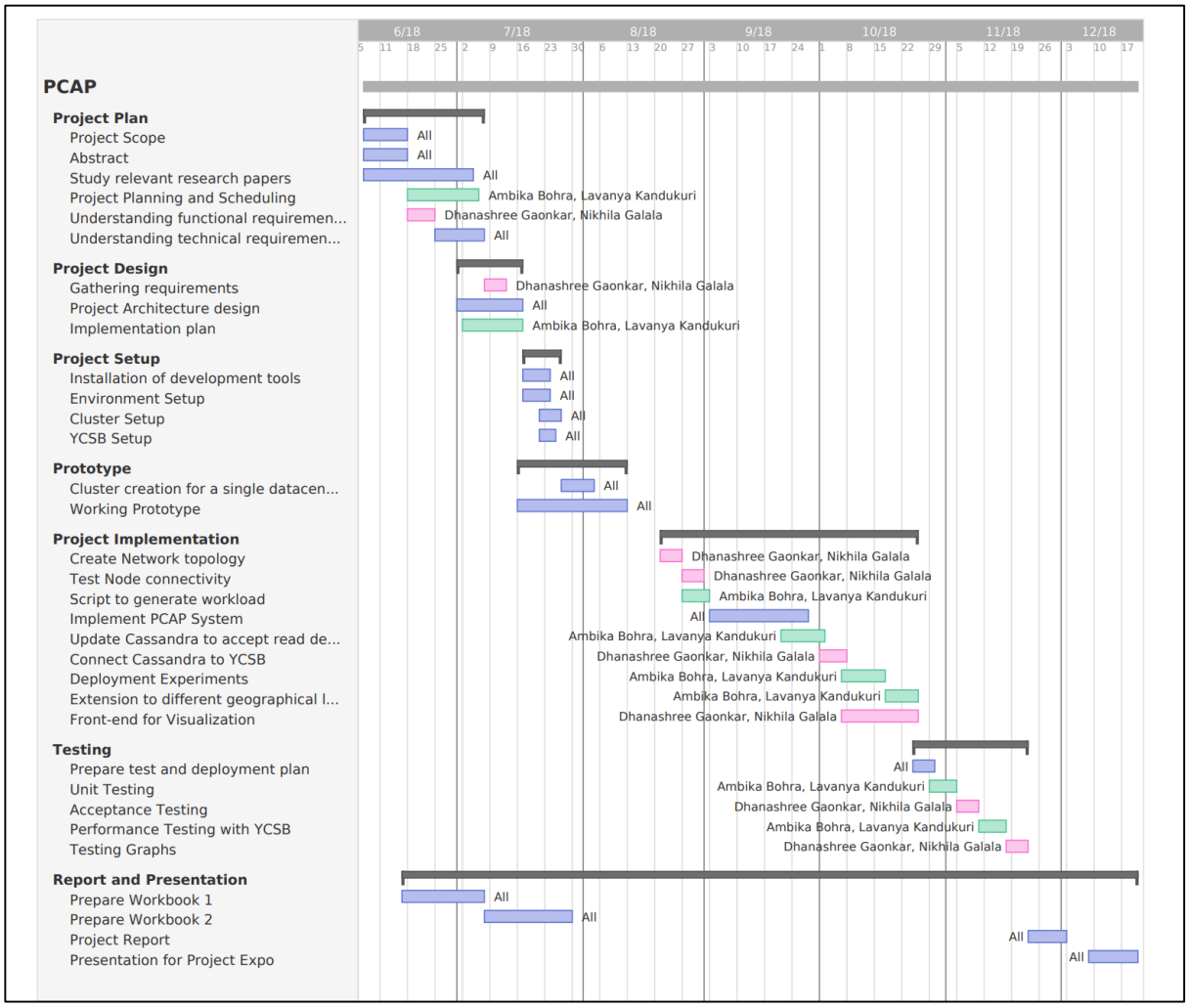


Fig 7. Gantt Chart

9.2 PERT Chart

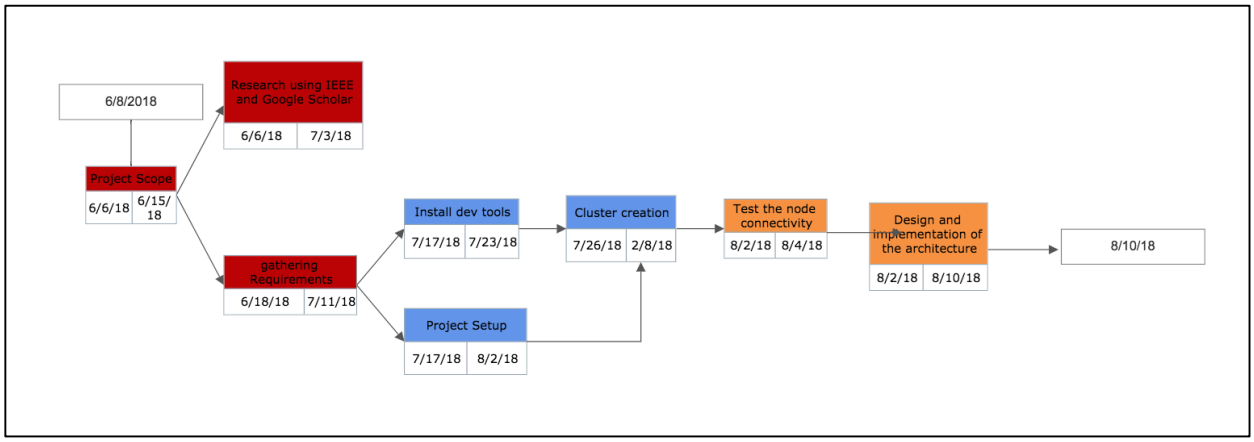


Fig 8. PERT Chart

Task Number	Task Name	Start Date	End Date	Predecessor
A	Project Scope	6/6/2018	6/15/2018	None
B	Abstract	6/6/2018	6/15/2018	A
C	Study relevant research papers	6/6/2018	7/3/2018	A
D	Project planning and scheduling	6/18/2018	7/4/2018	C
E	Understanding functional requirements	6/18/2018	6/22/2018	D

F	Understanding technical requirements	6/25/2018	7/5/2018	D
G	Gathering requirements	7/6/2018	7/11/2018	E
H	Project Architecture design	6/29/2018	7/16/2018	F
I	Implementation plan	7/2/2018	7/16/2018	H
J	Installing development tools	7/17/2018	7/23/2018	I
K	Environment Setup	7/17/2018	7/23/2018	I
L	Cluster Setup	7/20/2018	7/25/2018	K
M	YCSB Setup	7/20/2018	7/24/2018	K
N	Cluster creation	7/26/2018	8/2/2018	L
O	Working prototype	7/16/2018	8/10/2018	J
P	Create network topology	8/21/2018	8/24/2018	O
Q	Test node connectivity	8/27/2018	8/30/2017	P
R	Script to generate workload	8/27/2018	8/31/2017	P
S	Implement PCAP system	9/3/2018	9/26/2018	R

T	Update Cassandra to accept read delay	9/20/2018	10/1/2018	S
U	Connect Cassandra to YCSB	10/1/2018	10/5/2018	S
V	Deployment experiments	10/5/2018	10/16/2018	U
W	Extension to different geographical regions	10/17/2018	10/24/2018	V
X	Front-end for visualization	10/5/2018	10/24/2018	U
Y	Prepare test and deployment plan	10/24/2018	10/29/2018	X
Z	Unit Testing	10/29/2018	11/2/2018	Y
AA	Acceptance Testing	11/5/2018	11/8/2018	Z
AB	Performance testing with YCSB	11/9/2018	11/15/2018	AA
AC	Testing Graphs	11/16/2018	11/21/2018	AB
AD	Prepare Workbook1	6/15/2018	7/5/2018	D
AE	Prepare Workbook2	7/6/2018	7/27/2018	I
AF	Project Report	11/22/2018	11/30/2018	AC

AG	Presentation for Project Expo	12/7/2018	12/19/2018	AF