**Department of Computer Science**

**BSCCS Final Year Project 2023-2024**

**Interim Report I**

**23CS086**

**Vector Distributed Database for**

**Extending LLM capabilities with private knowledge.**

**(Volume 1 of 1 )**

**For Official Use Only**

|  |  |  |
| --- | --- | --- |
| Student Name | **:** | **LIAO Hua Xin** |
| Student No. | **:** | **56264031** |
| Programme Code | **:** | **BSCEGU4** |
|  |  |  |
|  |  |  |
|  |  |  |
| Supervisor | **:** | **Prof LI, Zhenjiang** |
| Date | **:** | **November 13, 2023** |

[1 Introduction 3](#_Toc158117614)

[1.1 Distributed Database over Centralized Database 4](#_Toc158117615)

[1.2 Distributed Database Breakdown 4](#_Toc158117616)

[1.3 Indexing of Vector Spaces 5](#_Toc158117617)

[1.4 Development Approach 5](#_Toc158117618)

[2. Literature Review 6](#_Toc158117619)

[2.1 Consouses Algorithms 6](#_Toc158117620)

[2.1.1 Paxos 6](#_Toc158117621)

[2.1.2 Raft 7](#_Toc158117622)

[2.1.3 Practical Byzantine Fault Tolerance (PBFT) 8](#_Toc158117623)

[2.2 Query Engine 8](#_Toc158117624)

[2.2.1 Custom Query Engine 9](#_Toc158117625)

[2.2.2 LlamaIndex 9](#_Toc158117626)

[3. Proposed System Design 10](#_Toc158117627)

[3.1 High-level Overview 10](#_Toc158117628)

[3.2 System Diagram 10](#_Toc158117629)

[3.3 System Components 11](#_Toc158117630)

[3.3.1 Front End Component 12](#_Toc158117631)

[3.3.2 Backend configuration 12](#_Toc158117632)

[3.3.3 Vector Database 12](#_Toc158117633)

[4. Methodology & Implementation 13](#_Toc158117634)

[4.1 First Approach 13](#_Toc158117635)

[4.2 Backend Focus and Testing 14](#_Toc158117636)

[4.3 Simple In-Memory Vector Store 14](#_Toc158117637)

[4.4 Database Enhancement 15](#_Toc158117638)

[5. Monthly Logs 16](#_Toc158117639)

[6. Project Plan 17](#_Toc158117640)

[7. References 18](#_Toc158117641)

### 1 Introduction

The solution proposed in this paper aims to create a vector based distributed database to keep large all the information about the import and export of the goods in Hong Kong. It is unclear how exactly the Hong Kong Government keeps track of those documents. However, according to the Commerce and Economic Development Bureau, the city has been promoting adopting an electronic and digital approach. [1] The reason behind storing those information as a high dimensional vector comes from the opportunity in the future of connecting this database to a generative AI model to facilitate the enquiry regarding specific goods.

#### 1.1 Distributed Database over Centralized Database

Nowadays, the adoption of a centralized database is being substituted by using decentralized solutions. The main reason behind this is not the reduced cost but its benefits[2]. Most people may not realize this, but most of the solutions we use to deploy database solutions, such as MongoDB, Apache Cassandra, and Apache Hadoop, are implemented using a decentralized approach [3]. The main benefit of adopting a decentralized database is its scalability feature, allowing the whole system to handle a higher data input by adding more nodes and resources to the system [3]. Another significant benefit is the reliability of the service as, if one of the nodes becomes unavailable, there are still several participants in the system that can keep the service running, and depending on which approach is taken, either data fragmentation or data replication, it may lead only to partial unavailability or total availability of the data [3]. The techniques mentioned and other benefits of the solution will be discussed in future chapters.

#### 1.2 Distributed Database Breakdown

Implementations of a distributed database have been discussed for many years, and there are already multiple solutions with different techniques in the market. When developing a distributed database, there are several similar steps to take. First of all, the selection between fragmentation and duplication. Both techniques take into consideration the replication of the data aspects as well. Fragmentation is also divided into 3 categories [2][4]:

* Horizontal fragmentation consists of dividing the database table into subsets grouped in rows.
* Vertical fragmentation, which consists of dividing the database table into subsets grouped by columns
* Mixed fragmentation, which is a combination of horizontal and vertical fragmentation

Fragmentation is often followed by the replication technique, which stores one or multiple copies of data in different nodes; when the whole data set is replicated in several machines, we can refer to it as duplication [2]. Another critical decision to make when developing a distributed database is the choice of a consensus mechanism such as Paxos, Raft, and PBFT, which have been widely used and optimized. It is hard to estimate the “best” consensus mechanism as different algorithms have distinctive advantages[5]. Last, it decides between adopting the ACID ( Atomicity, Consistency, Isolation, Durability) or BASE ( Basically Available, Soft state, Eventually consistent) model. The choice of adoption of either of these two models will direct the development of a relational(ACID) or non-relational database(BASE) [6].

According to the CAP(consistency, availability, and partition tolerance) theorem, when developing a database, it is essential to satisfy at least two of those requirements, and depending on which one is taken with higher importance, it would lead to the adoption of either SQL or NO-SQL database[6].

#### 1.3 Indexing of Vector Spaces

This project aims to store data as vectors, leading to the necessity to adopt algorithms to index the vector spaces to facilitate the querying as there is the need to avoid looking through all the data at every information look-up. The problem faced is similar to KNNS (K-Nearest Neighbor Search) problems. Although many of them face the problem of the so-called curse of dimensionality, there are graph-based solutions such as HNSW (Hierarchical Navigable Small World) that are capable of offering great accuracy even with higher dimensional vectors.[7]

#### 1.4 Development Approach

The proposed solution in this paper aims to create a vector based distributed database to maximize the system's scalability, performance, and fault tolerance. When deciding which approach to take to develop the database, the focus will be on maximizing the features mentioned before. Some challenges and problems were not mentioned in the previous section, and the ones that this paper will primarily tackle are the following ones[8]:

* Replication Control: As the Replication of the data is used to copy the same information across different nodes in the system, there will be an additional effort to propagate the changes in all the nodes and ensure that they are uniform.
* Deadlock Handling: A common problem arises when numerous users are trying to access the same resource, and it may lead to the unavailability of specific data for a prolonged time.
* Resource Management: All the nodes in the system are sparse in different geographic regions, marking the importance of a protocol that eases the communication between the network and application layer.

The first step would be the analysis of the characteristics of every approach in developing a distributed database and developing a prototype, which will focus on finding solutions to the problem mentioned above. The second essential aspect of the development process of the solution proposed in this paper consists of multiple performance tests, which will take as a parameter multiple aspects such as the number of clients, network bandwidth, cache size, and many others[9]. Depending on the results of those tests, the selection of different algorithms may be considered, and further test cases will be performed.

### 2. Literature Review

The first step in designing the distributed database is choosing an adequate consensus algorithm, and the first stage of this project was researching the most popular ones available and deciding which approach to take.

#### 2.1 Consouses Algorithms

##### 2.1.1Paxos

The Paxos algorithm ensures that the nodes in the system will agree on a specific value, even with the possibility of failure or network delays. The algorithm works following a sequence of different phases involving all the nodes in the network, which will cover different roles: proposers, acceptors, and learners[10]. The main phases can be summarized in 3; the first phase consists of the proposer sending to acceptors a list of “proposals” to accept, which then will wait and collect all the responses. The second phase consists of reiterating the first phase, meaning that the system will keep sending new proposals with the highest vote until it will not receives any response; it will proceed to send the final proposal to the acceptors, which will respond if the majority of the nodes approved the value, the proposal will send a learn message to all the learners in the system so they can update their value[9].

The system is not fault-proof, as failure can always occur between the nodes, which are also called processors.

The relation between the processors N and the failure F is given by[10]:

Which defines the maximum number of nodes or processors in the system needed to achieve consensus, which, in this case, at least half of the nodes plus one that will not fail[9].

##### 2.1.2 Raft

Compared to Paxos, Raft is relatively easier to understand, and it divides all the nodes into three roles: leader, follower, and candidate [11]. The algorithm can be divided into three main phases. The first phase consists of the Leader election, and all the nodes are initialized as followers, and an election timeout is started. The group leader sends a heartbeat message to notify the other nodes about its existence. However, the follower will become a candidate because there are no leaders at the start when the timeout reaches 0. It will stay in this state until it wins the election or another candidate wins. To be selected as a leader, each candidate sends a Request Vote, and the node with more than half of the votes will be elected[11].

The second phase is called log replication, where clients will submit their request to the leader, which will append the command to its local log. This instruction will be broadcast to all the nodes in the system, and the leader will make sure all the participants in the network will also write it in their local log. Once the leader confirms that the log has been appended to the followers' log, it will commit the entry and inform the other nodes[11].

The last phase is about how the algorithm handles failures, for example, when a leader broadcasts its log to the network while a follower is out of service. These two nodes would have a different log history, and if, during the next election, the node with the different log wins and becomes the new leader, it would then overwrite the previous log. To tackle this problem, Raft implements a restriction that forces the previous leader always to submit the log before the next election, ensuring that the new leader in the next round will have the same local log[10].

##### 2.1.3 Practical Byzantine Fault Tolerance (PBFT)

The PBFT algorithm was designed to reach consensus even in the presence of nodes in the network that may send malicious or incorrect messages. The nodes in the system are called replicas, and they are divided into primary and backups, and there are mainly three stages to reach counseus[12].

The first stage consists of the master or primary node receiving the request message, and after verifying its correctness, it will generate a pre-prepare message and broadcast it to all the other nodes[12].

The second stage consists of the system's replicas to verify the pre-prepared message's authenticity. Afterward, they will generate a prepared message that will broadcast to all the nodes[12].

The last stage consists of sending a commit message after verifying the prepared message; it is necessary to reach a number of 2f+1 commit messages ( one node more than half of the replies in the network) to reach consensus and broadcast the changes[12].

Although it is vastly adopted because it does not use a large number of resources and it reaches counsels relatively fast, it has several disadvantages, such as high network usage caused by the high number of messages exchanged during the algorithm[12].

#### 2.2 Query Engine

When it comes developing a database, it is essential to take into considerations the interaction between the application or user and the data storage. In the last decades we have seen the popularity of the usage of SQL language, an English friendy language which was made possible through the SQL engine underneath.

The latter one translates SQL queries into executable operations by parsing the query to validate its syntax, optimizing it for efficiency, creating an execution plan with the required operations, and finally executing this plan to retrieve or manipulate data, thus allowing users to interact with the database using high-level commands.

##### 2.2.1 Custom Query Engine

Although many query engines exits in the market, an idea would be developing a custom query engine, focusing on defining a query language and parser, a robust execution engine, optimization techniques and lastly how to be connected to a data storage. One approach would be leveraging high level language like Scala, to benefit from abstraction while ensuring performance by translating the intructions into optimized C code[13]. The process of translating to a lower level langaugege is optimized through the usage of frameworks such as SC compiler, which instead of generating low-level code from high-level code in a single step, it will progressively lower the level of abstraction as shown in the Fig.1[13].

A diagram of a software development process

Description automatically generated with medium confidence

Fig.1. Abstraction Process Example

Source: Adapted from [13]

There are many other aspects to take into considerations but focusing on the development detail in this section would be out of scope.

##### 2.2.2 LlamaIndex

The end goal of the project is being able to extend the querying capabilities of existing large language model using a custom vector database, and for that purpose, LlamaIndex is considered one of the most innotive existing solution to be used as integration between LLM and any datastorage[14]. It can serve the purpose as query engine, relieving the duty from the developers to write their own systems to parse the queries and allows an easy integration with most of the LLM in the circulation[15]. It allows the developers to upload their own documents, the information then will be divided into smaller, manageable chunks (indices) for analysis and stored in a chosen storage. In order to identify the indices most aligned with a query, an embedding is created for the query itself. Then, the system calculates the cosine similarity between this query embedding and the embeddings of various indices. This process helps in pinpointing the indices that closely match the query[14][15].

### 

### 3. Proposed System Design

#### 3.1 High-level Overview

The project will include three primary sections: the front-end interface, designed for ease of use, allowing users to upload documents and querying through a chat function. The second component is a backend service that facilitates communication between the front-end and the query engine through APIs. This backend connects to LlamaIndex, responsible for indexing documents and computing similarity scores between the documents and queries. Additionally, it offer the possibilities to leverage the capabilities of OpenAI's ChatGPT, enhancing document processing and query resolution capabilities with advanced AI functionalities. For testing purposes, two options of DB will be offered to able to create a comparison table between the usage of document based database such as MongoDB and a Vector Database.

#### 3.2 System Diagram

The system design of the project can be summarized in the Fig.2, the front end and backend layer are very straightforward as they only needed to perform testing and check if the results from the query are correct. In the previous chapter it was mentioned the possibility of using a custom developed query engine or LlamaIndex. The choice to use the latter one come from the complexity and time consuming task of developing from scratch an query engine. The tasks would consist constructing the logic for the parsing and translation of the input, to its optimazion and finally evaluation. It will also then lead the choice for choice of a suitable algorithm to extensive testing to ensure high scalability and performance as the volume increases[16]. This would heavily impact the development of the other sections of the project, meanwhile the framework already available offer extreme flexibility allowing even custom query engine and allowing the usage of different type of data storage, leading to the possibility of conducting tests on different scenarios[15].

A diagram of a software company

Description automatically generated with medium confidence

Fig.2. Vector Database Usage

#### 3.3 System Components

The project can be mainly divided in three components, the front-end section which can be a simple web application, which then is connected to LlamaIndex through a backend server. The third section are the database and for the purpose of testing the solution using a non vector based database, MongoDB will be adopted, as it is very simple to set up. On the other hand, most of the development time will be focused on developing a Vector DB from scratch.

##### 3.3.1 Front End Component

The front-end part will be made with React and will have a simple design with two parts: one for uploading personal documents like financial reports, and another for a chat where users can ask questions and get answers about their documents. After the wrapping up the application, it will be deployed on Amplify to simplify the integration with AWS services like EC2[17].

##### 3.3.2 Backend configuration

The backend part is a web server written in Python that provides APIs for the frontend to upload documents and send querie and connecting to LlamaIndex for processing. The system uses the Vector Store Index technique to break down documents into nodes and create vector embeddings, specifically using OpenAI's text-embedding-ada-002 model for its embeddings. It also involves choosing a storage solution to save indexed data to prevent re-indexing of the data, and for this project, two different storage will be adopted, MongoDB and a vector database. The last component of LlamaIndex is the choice of the query engine. The last stage can be divided into three phases, the retrieval of the most relevant documents, the postprocessing of the data and lastly the response synthesis, where the retrieved data is combined with the initial prompt and sent to the LLM to get a response. Extensive testing and evualuation has to be performed to find the most suitable configuration and find a balance between precision and performance

##### 3.3.3 Vector Database

The creation of a Vector Database splits into two key areas: one for data storage and persistence, and another for operational logic, including basic CRUD actions, similarity, and indexing methods. For storage, the options are developing a distributed database, covering everything from security to performance, as discussed earlier, or utilizing existing cloud storage solutions by creating an interface to manage them. As shown in the Fig 3, it would allow the system to automatically add different type of clouds service to store the indexed data. Using cloud services, there is also the advatange of delegating the complications of security measurements for the data.

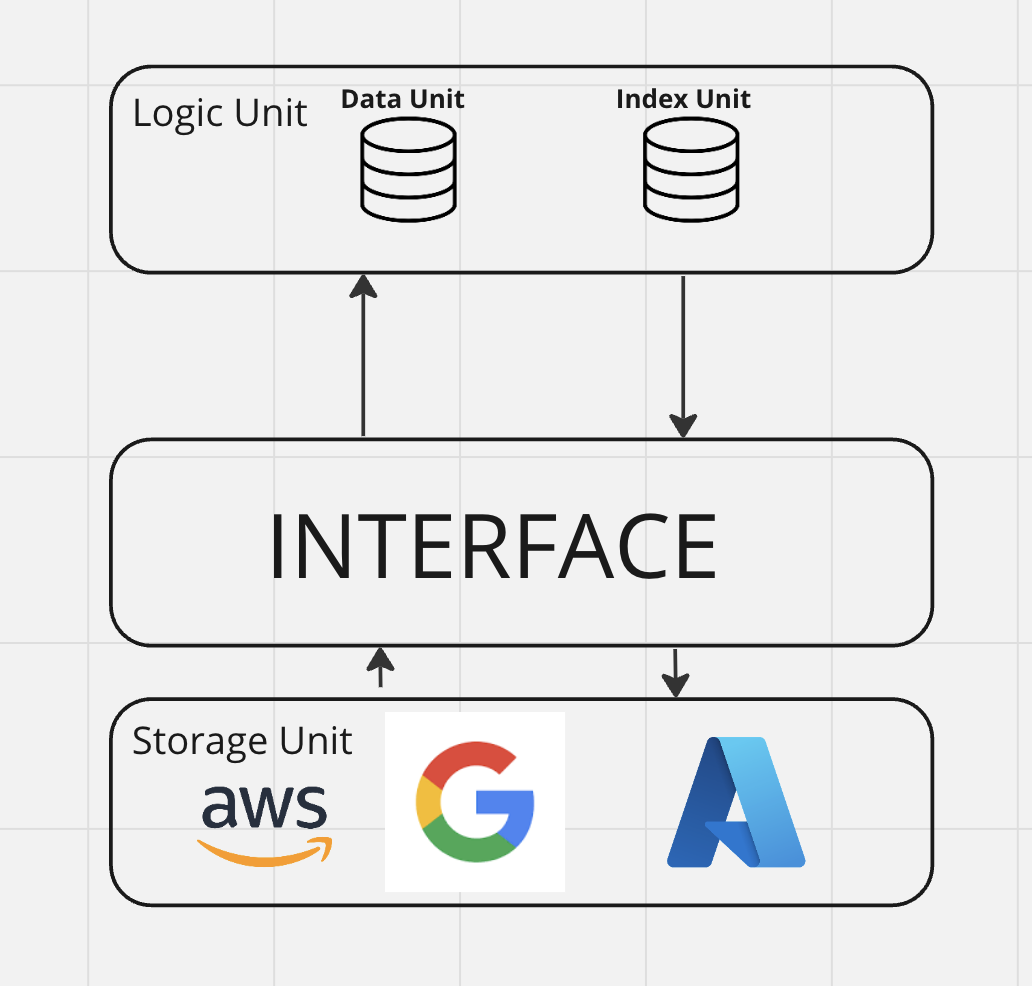


Fig.3. Cloud Solution Database

Both of the approaches, though, need a logic unit and the indexing plays a crucial role in processing queries efficiently. However, choosing the right indexes is difficult due to the variety of available indexes for vector similarity searches, each with its own set of pros and cons in terms of performance, accuracy, and storage requirements. For the initial setup, the one suggested by LlamaIndex will be adopted but different ones will be tested to see if better results can be achieved. For the data management, the insertion and deletion efficiency can be enhanced using LSM-tree[18]. New entries temporarily stay in an in-memory MemTable, which, upon reaching a certain size or time interval, moves to disk as a segment. The scalability and availability of the database can be handled by running multiple instances in different nodes and specific roles can be specified such as having more reading instances than writing instances, as it is expected to have many more input prompt and uploading new documents.

### 4. Methodology & Implementation

#### 4.1 First Approach

Given the project's scope, the development priorities are clearly outlined, with the database development and backend integration with LlamaIndex as the first target to finished developing in order to see early results. Front-end development is set to be the lowest priority. For data uploading to LlamaIndex, which should be considered private knowledge, previous assignment papers will be uploaded to check to accurancy of the system.

#### 4.2 Backend Focus and Testing

At the current stage, the backend development would be primarly focused on configurating LlamaIndex and the reason why python is chosen for this step is because the latter one offers an extensive python documentation compared to other programming languages. For the first stage of development, the basic configuration are kept such as leaving the default configuration for indexing, choosing initially MongoDB as DataStorage and a simple In-Memory Vector Store. As for the query engine, the basic configuration is kept, simply transforming query into an embedding and performing top-k retrieval after performing semantic search on every chunks of the uploaded document.

The connection with OpenAI LLM is obtained through API keys offerend from the CS department and while the development is still going, all the tests and operations are conducted locally. Few noticeable results are worth mentioning in the first few round ot testing as the system performance is very poor when it comes a large volue of data, especially with the simple-in-memory vector store. The topic will be further discussed in the next chapter.

#### 4.3 Simple In-Memory Vector Store

During the first stage of development and test the functionalities of LlamaIndex,a simple in-memory vectore store was implemented, including 4 simple operations consisting in add, get delete and query. For the storage it is used a simple key-value data structure allowing very straightforward CRUD operations but the limitation of this choice were quite immediate, especially when dealing with a high volume of data. For the query methods, a semantic search is choose to define the similarity between an user prompt and the stored documents. Subsequently, the results will be returned to the query engine which would attach to the original user prompt, the chunk of text with the highest similarity.

#### 4.4 Database Enhancement

From the previous approach, one of the biggest enhancement that must be done is the vector database. The consideration of moving the actual storage from local machines to cloud is taken in consideration as mentioned previously, as it would enhance not only the scalability issues, but also the security concerns as the most of the data security would be handled by the cloud providers[]. Adding a layer of abstraction in our system for the storage layer, the only remaining layer to be enhance will be the computing one, defining functions such as adding, updating, removing and also enhacing the semantic search algorithm.

### 

### 5. Monthly Logs

**October 2023 :** I mainly focused on reading papers regarding the consensus algorithm, as it would be one of the most critical aspects of developing a prototype. Read journals and papers regarding adopting Paxos, Raft, and PBFT. Brainstormed their pros and cons and evaluated the option to go deep into different algorithms. Except for the consensus algorithm, I have also investigated the advantages of taking a partitioning approach instead of a replication approach when distributing the data in the different nodes. Each method has it is own pros and cons, and I have come to the conclusion to adopt the partitioning technique as I value scalability and performance.

**November 2023 :** Further research is in progress as the project's requirements include adopting a vector database. Mainly focused on choosing the indexing algorithms and how to change the possible architecture of the distributed database and started doubting if partitioning was the correct choice. Additional time would also be allocated to designing an architecture that allows the system to be plugged in with other generative AI model vector databases.

**December 2023 :** Further research is in progress as the project's requirements include adopting a vector database. Mainly focused on choosing the indexing algorithms and how to change the possible architecture of the distributed database and started doubting if partitioning was the correct choice. Additional time would also be allocated to designing an architecture that allows the system to be plugged in with other generative AI model vector databases.

**January 2024:** Further research is in progress as the project's requirements include adopting a vector database. Mainly focused on choosing the indexing algorithms and how to change the possible architecture of the distributed database and started doubting if partitioning was the correct choice. Additional time would also be allocated to designing an architecture that allows the system to be plugged in with other generative AI model vector databases.

### 

### 6. Project Plan

|  |  |
| --- | --- |
| November 2023 | Conclude research about vector database and how it can be adopted as a distributed database |
| December 2023 | Define the Security measurements to be taken and design the distributed database architecture |
| January 2024 | Start the core development of the database and trying out different algorithms |
| February 2024 | Keep working on prototype development and work on the second interim report |
| March 2024 | Completition of the first prototype of the database and list the different test to be done. |
| April 2024 | Do performance testing and unit testing, adjust and fix eventual bugs and complete final report. |

### 7. References

[1] “Government Electronic Trading Services,” *Commerce and Economic Development Bureau*. <https://www.cedb.gov.hk/en/policies/government-electronic-trading->services.html

[2] A. Vikiru, M. Muiruri, and I. L. Ateya, “An overview on cloud distributed databases for business environments,” *arXiv (Cornell University)*, Jan. 2023, doi: 10.48550/arxiv.2301.10673.

[3] P. V. Mole and E. B. Noche, “EmpiricalResearch on the Challenges of Distributed Databases: A Literature Review,” *Southeast Asian Journal of Science and Technology*, vol. 6, no. 2, Art. no. 2672–2992, 2021.

[4] Gupta, S., Saroha K. (2011), “Fundamental Research of Distributed Database”, IJSMS, 11

[5] Zhou, Sisi & Li, Kuanching & Xiao, Lijun & Cai, Jiahong & Liang, Wei & Castiglione, Arcangelo. (2023). A Systematic Review of Consensus Mechanisms in Blockchain. Mathematics. 11. 2248. 10.3390/math11102248.

[6] Birgen, Cansu. (2014). SQL vs. NoSQL. 10.13140/RG.2.2.22364.13447.

[7] P. Vadiraja, “Leveraging Reinforcement Learning for evaluating Robustness of KNN Search Algorithms,” *arXiv.org*, Feb. 10, 2021. https://arxiv.org/abs/2102.06525

[8] S. Rana, M. K. Sohel, and S. Arman, “Distributed Database Problems, Approaches and Solutions — A Study,” *International Journal of Machine Learning and Computing*, vol. 8, no. 5, Oct. 2018, doi: 10.18178/ijmlc.2018.8.5.731.

[9] G. Denaro, A. Polini, and W. Emmerich, “Performance testing of distributed component architectures,” in *Springer eBooks*, 2005, pp. 293–314. doi: 10.1007/3-540-27071-x\_14.

[10] R. Bhardwaj and D. Datta, “Consensus algorithm,” in *Studies in big data*, 2020, pp. 91–107. doi: 10.1007/978-3-030-38677-1\_5.

[11] J. Hu and K. Liu, “Raft consensus mechanism and the applications,” *Journal of Physics*, vol. 1544, no. 1, p. 012079, May 2020, doi: 10.1088/1742-6596/1544/1/012079.

[12] X. Zheng and W. Feng, “Research on practical byzantine fault tolerant consensus algorithm based on blockchain,” *Journal of Physics*, vol. 1802, no. 3, p. 032022, Mar. 2021, doi: 10.1088/1742-6596/1802/3/032022.

[13] Amir Shaikhha, Yannis Klonatos, and C. Koch, “Building Efficient Query Engines in a High-Level Language,” ACM Transactions on Database Systems, vol. 43, no. 1, pp. 1–45, Apr. 2018, doi: <https://doi.org/10.1145/3183653>.

[14] Zirnstein, Bruno. (2023). Extended context for InstructGPT with LlamaIndex.

[15] “LlamaIndex 0.9.24,” docs.llamaindex.ai. <https://docs.llamaindex.ai/en/stable/index.html>

[16] M. Rupley, “Introduction to Query Processing and Optimization Introduction to Query Processing and Optimization.” Available: https://clas.iusb.edu/computer-science-informatics/research/reports/TR-20080105-1.pdf

[17] V. Nguyen, “Building a Serverless Full-Stack Blog Application Using AWS Amplify,” 2022. Accessed: Feb. 05, 2024. [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/780986/Vy\_NguyenThanh.pdf?sequence=2&isAllowed=y

‌