# AquilaDB: A Decentralised Neural Information Retrieval System

**Authors** (ETH)
0x7cF68682478ca8E6299F2a6A00B047Ea3E727Ec6
0xd1B749c8DE5c6243e4f12C8d16d74Ae5Af8D2A50

**Abstract.** A decentralized search layer would promote distribution, collaboration, and consumption of data on an internet-scale. We propose an adaptive search layer on top of the existing data layer. This new layer federates the discovery of new contextual meanings from the same data layer and conveniently distributes it. This repetitive and collaborative transformation of raw unstructured data into contextualized structured representations would eventually increase the value of the data layer. A generic representation layer that is equally accessible to any agent (human or machine) might open up a decentralized free market for data through an emergent process. This market will benefit all the data owners, manipulators, and consumers. Data markets could unlock a large portion of data that stays frozen behind the walls of large organizations and make it potent.

## 1. Introduction

The internet is a global platform for peer to peer communication. The Web acted as an inter-connected data layer on top of that. This data layer grows exponentially as the Web evolves. A major share of data available on the Web is in unstructured form which is complicated for an agent to comprehend. Proper management and analysis of this ever-growing data could bring insights that might solve the greatest problems we face today. However, the interpretation of data is contextual. The same data can be used in two different contexts to generate insights that solve two different problems. Today, almost all sustainable solutions for data management on the Web are practically doing this contextual interpretation of public data and making a great impact. However, there is a big room for improvement and the unaddressed problem domain is huge. One obvious reason is the risk of the economic unsustainability of a solution. This is because of the scalability limitations present in strongly centralized economic models.

Web 3.0 is trying to solve these scalability problems from the ground up. Multiple building blocks are being introduced into this space from different angles. The goal of all these innovations is to reinvent a different economic model that scales beyond the limitations of Web 2.0. This includes decentralized data storage, immutable ledgers, and new models for governance and token economics.

Our proposed solution is expected to fill in between the underlying data layer and a market governed by a token economy. For this reason, the system by design is open-ended to enable adaptability between layers on both sides. The information captured is contextual to the defined schema and is pegged to the raw data available on the underlying data layer. The proposed system is designed to elastically scale from bare-metal to the cloud as the data and hardware demands grow.

## 2. Database

Databases are the first class citizens of this proposed system. It is a collection of JSON documents which follows a unique schema definition. It is the smallest divisible unit of information in a decentralized network. Any full-node which is interested in a specific database should replicate it from another full-node containing it. All the processing is then done by a full-node locally, over the data that's available till that moment.

### a. Schema

A Database is defined by its schema. Creation of a database involves defining a schema and then generating a unique content id for that schema.

#### i. Immutability

A schema definition, once defined and published to the system is immutable. Any alteration in its definition means the creation of a new database by publishing it to the system. Because there is no 'alter schema' functionality defined, any documents indexed under the old schema can not be transferred to the new one. This simple trade off is the backbone of this proposed system. However, it doesn't limit anyone from implementing a high level API for database alteration.

#### ii. Database id

A database id is generated automatically from the content hash of corresponding schema definition. Any two databases created independently with the same schema definition will have the same id across the network. This allows any independent actors to operate on their databases (offline or in closed networks) independently and synchronize later.

### iii. Schema definition

A schema is defined as a JSON object. This is because a JSON syntax is easy for an agent (both human and machine) to read and write. However, the schema definition is restricted to a predefined JSON format as described below.

1. _db_id_: content hash for the database schema definition
2. _indexes_: A nested JSON object specifies which keys of a document need to be indexed and their corresponding data type.
3. _model_: content id of the model which produces a generic representation for a document in the database.

### iv. Documents

As we have discussed, a database is a collection of JSON formatted documents. It is the smallest divisible information within a database. All documents are validated and all operations are performed locally within a full-node. A document is formatted as discussed below:

1. Every document should contain all the indexable keys and values as described in database schema definition. Otherwise, the document will be rejected on validation.
2. A document id is automatically generated from the hash of its contents.
3. A document shouldn't contain nested objects as values unless the key name is '_payload_'.
4. Do not use any of the reserved keys as they are used in schema definition. Below are the reserved keys:
    a. _id_: content hash generated for this document excluding _id_ and _deleted_ keys
    b. _vector_: This contains the semantic vector representation which is generated by the model specified in schema definition
    c. _payload_: Any valid JSON data. This key doesn't have any type limitations. Use this to keep any useful information that doesn't need to be processed. This data is neither indexed nor replicated. It is recommended to move everything that doesn't need to be indexed under this key to reduce computational and network resource utilization.

## b. Indexing

In the proposed system, indexes are redundant and are kept in two different formats for convenience at the expense of storage medium.

For each document inserted, a pruned version of it is kept inside the full-node for processing. The original document is pushed down to a Distributed Content Addressed Storage Network (DCASN) like IPFS or SWARM for later use. This will reduce the computational complexity and memory footprint while it is being processed. This will also help in reducing network utilization during replication.

Once the pruned version of a document is created, specified key-value pairs are indexed in a local storage for fast search and retrieval. These key-value pairs are sharded (horizontal partition of data) as we will see in upcoming sections.

An index document (pruned document) is defined as follows:

i. _id_: content id of the document
ii. Indexable key-value pairs taken from the original document with respect to schema definition
iii. _timestamp_: the latest timestamp of a document creation or deletion event. This is to make sure that conflicts are resolved during index replication.
iv. _deleted_: Proposed system keeps everything from the creation of the database. When a delete request is executed, it doesn't remove a document from the disk. A delete request sets the '_deleted_' key of a document to 'True' as an indication.

## c. Data storage

In the previous sections, we have given a high level overview of document storage. The original documents inserted into a database are immediately pushed down to a DCASN (Document store). Only a pruned, minimal version of it is retained locally inside a full-node for further processing and replication (Index store).

### i. Index store

Index stores are sharded key-value stores. Data kept here is redundant to support fast filtering and retrieval. By default, two indexes will be created each for _doc_id_ - index document and _vector_ - _doc_id_ pairs. In addition, every user defined indexable key will also have separate key-value storage sorted by the key itself. For this reason, a user should be careful to keep the schema definition as small as possible. This will make the database storage economically feasible for most of the full-nodes in the decentralized network.

### ii. Document store

A document store is an external DCASN layer. The proposed system is designed in such a way that this layer is highly customizable. The full-node can interact through an API provided by any internal or external DCASN services. The DCSAN ingests the document and returns a content id for that document. This document is redeemable at any time from anywhere through the same APIs in exchange for the content id.

## 3. Replication

Replication of a database from one full-node to another involves replication of the index documents itself. The basic steps for replication involves a full-node connecting to another full-node specifying the database id to be replicated and then starting the replication process. The full-nodes involved in replication use the Adapted Couch replication protocol to arrive at a common consensus on the state of the documents. Replication can be unidirectional or bidirectional.

### a. Modified Couch replication protocol

The proposed system introduces small changes to the original Couch Replication Protocol at the API level. Our system suggests multi transport replication over full-nodes with the help of libp2p specifications. Because our system doesn't allow modification of documents, the very specifics of Couch Replication protocol can be ignored while generating 'changes feed'. For more information on couch replication protocol, visit CouchDB documentation.

### b. CRDT

Conflicts during replication are resolved with automated JSON CRDT as implemented by Automerge. Any same key conflicts are resolved based on the latest _timestamp_ among the documents in comparison.

With this implementation, the proposed system is vulnerable to document manipulations by an attacker full-node. A single malicious full-node can affect the integrity of the data by influencing eventual consistency through manipulated _timestamp_ value. So it is up to the users to connect to a trusted full-node for replication. However, the architecture of the proposed system is designed in such a way that a network wide consensus mechanism can be integrated in future versions to maintain a global scale immutable logging of events.

## 4. Sharding

To enjoy the benefits of both scalability and high availability, the proposed system is designed to be globally decentralized while being locally centralized. As an analogy, this can be compared to Bitcoin full-nodes and mining pools. The full nodes itself are globally decentralized and each of them contains a full copy of the blockchain. However, to distribute the computational complexity in mining a new block, a full-node distributes the computation to a mining pool of centralized sub-nodes. As a result, the full nodes contribute to the diversification and decentralization of the network. At the same time, a sub-node distributes it's computational requirements to multiple computers or to a cloud infrastructure.

### a. Raft Consensus

We propose the Raft consensus algorithm to manage sub-nodes co-operation. A sub-node cluster is independent in its existence. Any request from the full-node is routed automatically to the master sub-node. A master sub-node gets elected according to Raft consensus algorithm and will aggregate and route data for consistent retrieval. A master sub-node maintains a heartbeat to keep slave sub-nodes in synchronization and it helps in updating DHT (Distributed Hash Table) to the latest synchronized state.

### b. Modified Kademlia DHT

Every sub-node will contain a DHT with all other sub-node id as keys and their local address as values. Based on shard mapping, Node_id for each document is deterministically obtained. This helps in identifying the exact node that can store a document or respond to a query. As a result of this, any sub-node can be targeted with any outside request. An internal sub-node to sub-node routing with the help of DHT will serve this request efficiently.

## 5. Model serving

Model serving is done by an independent component. This component takes input data in a well known format to generate the corresponding latent vectors. As the first step, a model is published into a DCASN service before the creation of the database itself. This returns a content id for that model and is then added to a database schema definition. Because of the independent nature, model serving can be done at full-node as well as sub-node level.

### a. Model Hub

Model hub is not an inherent part of our proposed system. It is done at the application layer by anyone who builds upon this system. One recommended implementation of a model hub can be a static html page, that can be distributed over the internet, version controlled and improved through collaboration. The design decision to move the models into DCASN allows free distribution of the model (only the content id) at the application layer. This removes the heavy lifting of hosting and distribution from the model hub.

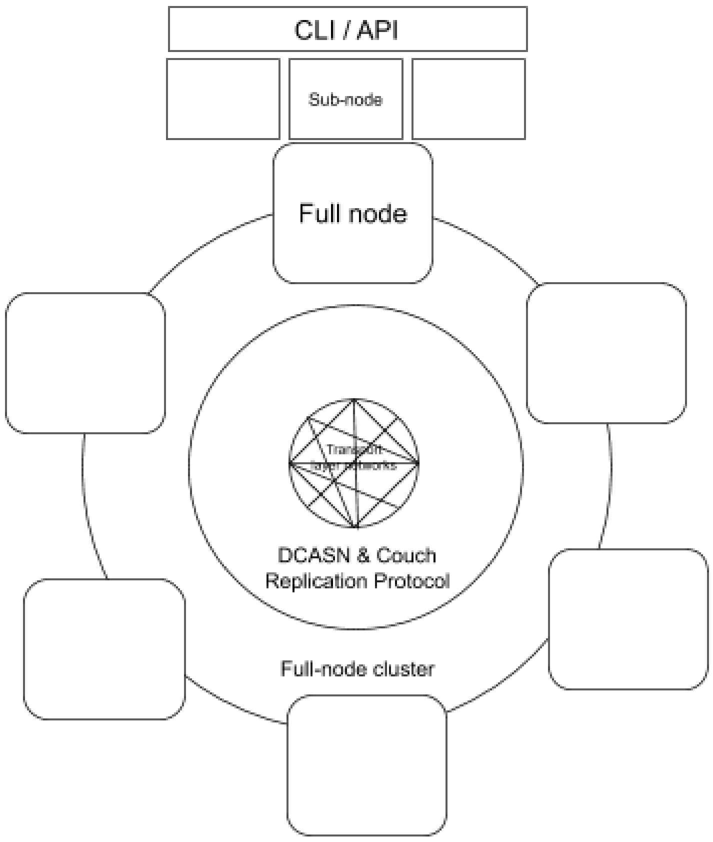
### b. Model fetching

When a database is replicated, a (full or sub) node could parse the schema definition to extract content id for the corresponding model. Once this content id is known, the node loads the model from DCASNetwork. The model serving module then serves this model to address new document insertions and queries.

## 6. Network

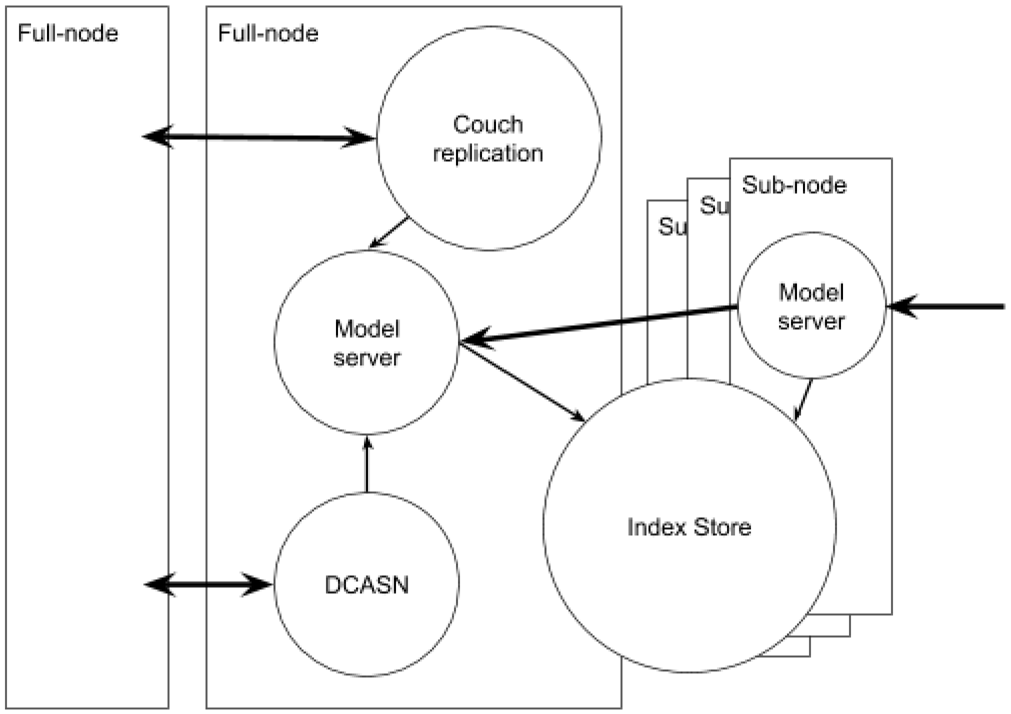
### a. Architecture

Full-node cluster is the backbone for the proposed system. A full-node cluster is powered by two independent, decentralized networks. DCASN is one among them. It stores and distributes content such as database schema, original documents and pretrained model. The second network is the Couch replication network. This moves indexes between full-nodes through eventual consistent replication and deterministically arrives at a consensus for database state. This architecture is designed to allow easy integration of more networks in future versions. As an example, we can integrate a global immutable event ledger for better consensus and token economics.



Despite the decentralized nature of the full-node cluster, a full-node to sub-node relationship is centralized. This is important for local scalability because a full-node can contain multiple databases and different databases can have different storage and compute resource requirements. A centralized local cluster allows a full node to easily scale between edge devices and cloud infrastructure.

## b. Nodes



### i. Full-node

A user can configure a full node to contain multiple databases if they want. Joining a full-node cluster and subscribing to a database involves connecting to another trusted full-node and enabling replication for a database. This replication can be one directional (subscribe) or two directional (synchronize). Once the replication process is started for at least one database, the node is entitled to be a member of the full-node cluster.

Before making a service request, a client should identify a full-node that is capable of serving the database of interest. This knowledge should be acquired from other sources such as the internet or a marketplace. In this way, the proposed system doesn't limit the application layer customizations and integration with existing systems or tools in any manner.

For instance, multiple peers could provide the same service with different trade-offs such as latency, quality of data, high availability, affordability etc. There can also be applications on top of this service to dynamically choose between service providers and databases. So, in essence, the proposed system doesn't need to do anything regarding routing of requests.

#### ii. Sub-node

Sub-nodes expose APIs to serve all requests from the outside. These include but not limited to insert documents, delete documents, search documents, on-board or off-board itself. Every sub-node keeps a part of the entire index under a full-node. Sub-nodes can join and leave dynamically to support horizontal scalability.

Requests are routed through sub-node communication by maintaining a distributed hash table. Communications between a full-node and sub-nodes are co-ordinated by a master sub-node. The master sub-node election process and synchronization is achieved through Raft consensus algorithm.

### c. Validation

Every document that flows into a full-node or sub-node is validated. Validation process is straight forward. A node checks whether the input document is a valid JSON. If it is, the next check is to make sure that the document follows the schema definition of the corresponding database. Validation step also includes a check for encrypted data, which we discuss in the sections below. Once the document is validated, it will be forwarded to the next module. Otherwise, it is ignored.

## 7. Privacy

The Internet is a complicated cluster of networks with varying levels of permission. Data privacy (or data ownership) is one among them. Our proposed system in its first version allows very basic yet very useful data protection techniques of public key cryptography.

### a. Data encryption

With the proposed system, the advantage is the seamless replication of any database from one full-node to another. However, there might be some databases that a full-node doesn’t want anyone to replicate. In a decentralized system, it is not possible for a full-node to determine whether a subscriber full-node is trustworthy or not and act accordingly. By simply encrypting data before replication shifts this responsibility to the subscriber full-node. As long as the subscriber full-node could use a valid private key to make sense of the data, the replication happens. Otherwise, it will be terminated.

## 8. Conclusion

We have proposed a system that brings a flexible layer for discovery and management of structured information on the web 3.0. This search layer is designed to well adapt to the ever growing decentralized data stores (content addressed). Information is stored as a generic representation that any kind of agent can manipulate. This in turn brings cooperation between different agents and might encourage an emergent free data market.

## References

[1] Bhaskar Mitra et al., “An Introduction to Neural Information Retrieval”, 2018.
[2] Damien Katz et al., “Apache Software Foundation CouchDB", 2005.
[3] Juan Benet, “IPFS - Content Addressed, Versioned, P2P File System”, 2014.
[4] Diego Ongaro et al., “In Search of an Understandable Consensus Algorithm”, 2014.
[5] PetarMaymounkov et al., “Kademlia:A Peer-to-peer Information System”, 2002.
[6] Johnson et al., “Billion-scale similarity search”, 2017.
[7] Martin Kleppmann et al., “A Conflict-Free Replicated JSON Datatype”, 2017
[8] Satoshi Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System”, 2008

**This paper (aims to educate the community) is a consolidation of experiments happening at https://github.com/a-mma/AquilaDB and https://github.com/a-mma/QnA-baker from 2018. Navigate for source code exploration and participation in future developments.*