

# Modified TorqueDB

Jeet Ahuja, Akshat Kumar, Yogesh Simmhan  
*Department of Computational and Data Sciences*  
*Indian Institute of Science*  
Bangalore, India

**Abstract**—Immense growth in the field of IoT and Edge has also brought with it and exponential increment in the amount of data that comes naturally because of the 1000s of sensors which export time-series data in real time. As such, its often essential to query for such data for multiple purposes ranging from working with a subset of data to monitoring the health of the infrastructure. With the improvement of edge devices, we can now retain and store data locally on the edge devices itself rather than going through the hassle of moving every single bit of it to the central cloud. However, what such systems do lack is the availability of flexible querying over all such data which may be spread across 100s of such devices. This is where TorqueDB, developed by Dhruv Garg, et al. comes in handy. By default, TorqueDB leverages on the use of ElfStore, a distributed edge-local file store for file system and InfluxDB, a time-series database. For temporal querying on the data across the edge and fog devices. We intend to make the TorqueDB system more efficient by replacing ElfStore with IPFS (Inter-Planetary file system) and use Elasticsearch from ELK stack for querying and indexing over the data.

## I. INTRODUCTION

### A. Context

IoT domains span physical infrastructure such as Smart Cities, Smart Transportation and Industrial IoT, to consumer devices such as Smart watches and smart appliances. In IoT applications the data collected from an array of sensors is analysed and decisions are made within few seconds to control the system.

As of late, edge devices are being widely deployed, because they help gather and transmit observations from the sensors, and to enact control decisions. This has resulted in a lot of data, especially time-series data.

### B. Motivation

IoT applications are often deployed on an array of several sensors that collect huge amounts of time-series data that help monitor, analyse, and take appropriate action over the concerned events.

However, a boom in this field has seen an explosive growth in the amount of time-series data, generated especially by high-samplingfrequency sensors. As such, the actions to be taken as an outcome of such data is generally a result of the analysis performed over a recent subset of the collected data. This is more flexible than Complex Event Processing (CEP) and publish-subscribe systems that operate on data streaming

data and limit possible queries.

Although, TSDBs like InfluxDB are popular choices for hosting such data from sensors and perform temporal queries on the data an these TSDBs reside on the cloud. However, since the sensors producing this data and the application consuming this data tend to reside close to the edge devices, sending every query to the central cloud can incur significant latency which is an important factor to consider in cases where the Edge application requires sub-second query latency to respond to dynamic solutions. Also, there are the factors of unreliability, security and cost when moving data from edge to cloud and serving queries in the opposite direction.

Also, we lose a lot of compute and storage potential in terms of the resources available on the edge and fog devices themselves. This gives us the motivation to develop an efficient storage and querying system that resides right on the edge and fog devices.

### C. Contributions

This paper contributes in modifying the existing version of TorqueDB. The existing version of TorqueDB uses ElfStore, and this paper tries to integrate the combination of IPFS and elastic search with TorqueDB. IPFS will replace the ElfStore and elastic search is used for indexing and querying purpose on the IPFS.

It is expected that the modified version of TorqueDB performs better in query planning and execution. Because here we are using elastic search for querying and elastic search is used for fast queries.

On the other hand, ElfStore has an in-built mechanism for searching.

Next, in *section-2*, we discuss the background of ElfStore and related works; in *section-3*, we will discuss the Background and Technical Approach, where IPFS, Elasticsearch InfluxDB are discussed; in *section-4* we discuss the proposed architecture and the approaches that we tried.

## II. GAPS/RELATED WORK

The above discussed motivation is where TorqueDB developed by Dhruv Garg, et al. comes in handy. It leverages

ElfStore distributed edge local storage and InfluxDB to offer distributed time-series querying over the edge and fog devices

We intend to make changes to these leverages by replacing ElfStore with IPFS and use ElasticSearch (ELK Stack) as an indexing mechanism in an effort to improve the efficiency and usability of the database.

#### A. ElfStore

Elfstore is an edge-local federated store for streams of data blocks. i.e., it is stream-based, block-oriented distributed storage service over unreliable edges. Reliable fog devices are used as a super-peer overlay to monitor the edge resources.

Elfstore provides federated metadata indexing using Bloom filters. It maintains approximate global statistics about the reliability and storage capacity of edges.

In ElfStore there are two resources, edge and fog. The fog resources are reliable and they connect with each other through a WAN/MAN. ElfStore supports some of the service API like, CreateStream(), UpdateBlock(), PutBlock(), UpdateStreamMeta(), FindStream(), FindBlock(), etc. The data model for ElfStore is a stream of blocks.

ElfStore performs device management and search using a P2P model, where the fog acts as super-peer and edges act as peers among them. ElfStore will host the actual data and to guarantee reliability and balance storage utilization it performs replication.

### III. BACKGROUND AND TECHNICAL APPROACH

**IPFS:** The Interplanetary File System is a protocol and peer-to-peer network for storing and sharing data in a distributed file system.

A content addressed storage is employed by IPFS. i.e., for accessing data through IPFS, contents of data is being, instead of location based addressing

The mechanism is like BitTorrent. Instead of having a centrally located server, IPFS is built around a decentralized system of multiple host-operator nodes where each node holds a part of the total data, resulting in a resilient system of file storage and sharing. Any node in the network can serve the required file using the content as an address, and other peers can find and request that content from any host having that content using Distributed Hash Table textcomp

Some commands of IPFS include, *ipfs add x*, which adds file x to the filesystem and print the resulting hash of the file. *ipfs get <hash>* retrieves a file from the filesystem, where <hash> is the IPFS hash of the required file

In IPFS, data is stored using IPFS objects. The IPFS object format is,

```
type IPFSObject struct {
    links [] IPFSLink
    //array of links

    data [] byte
    // opaque content data
}
```

A single IPFS object can store 256KB of data. For storing files with size greater than 256KB, splitting of file is performed with each split being less than 256KB

In the case of splitting of file, an empty parent object links all the pieces of the file together using the link field.

commit objects are used to support versioning in IPFS. parent object is linked to the updated object and so all updates are tracked as well as the older versions are also maintained

For our project, we'll have an IPFS network consisting of multiple fogs and edge resources. Client will send a query to the fog co-ordinator and the fog co-ordinator node will publish the query on the network.

**InfluxDB:** InfluxDB is an open-source TSDB optimized for high read and write throughput. It stores data in buckets (databases) that contain measurements (tables). Each row in a table has a timestamp and columns that are either tags, which are indexed, or fields, which can be aggregated on. It has a native Flux query language that allows SQL-like queries over time-series data, with support for Select, Project, Aggregate, Window-aggregates and Joins. Besides network APIs provided for data insertion and querying, data can also be bulk-loaded into an InfluxDB table using a line-protocol CSV format

**Elasticsearch:** Elasticsearch is distributed, open-source search and analytics engine built using Apache Lucene and developed in Java. It is a scalable version of the Lucene search framework and has the added ability to horizontally scale Lucene indices. Elasticsearch allows us to store, search and analyse huge volumes of data quickly and efficiently by searching the text directly. In simpler terms, we can think of Elasticsearch as a server that receives JSON requests and gives back JSON data, all of it being leveraged with extensive REST APIs

It uses inverted index. i.e., a hashmap-like data structure that directs from a word to a document. Elastic search distributes the tasks of searching and indexing across all the nodes in the cluster

ElasticSearch ensures redundancy by implementing sharding and replication techniques.

Elastic Stack (ELK Stack) is a set of open-source tools for

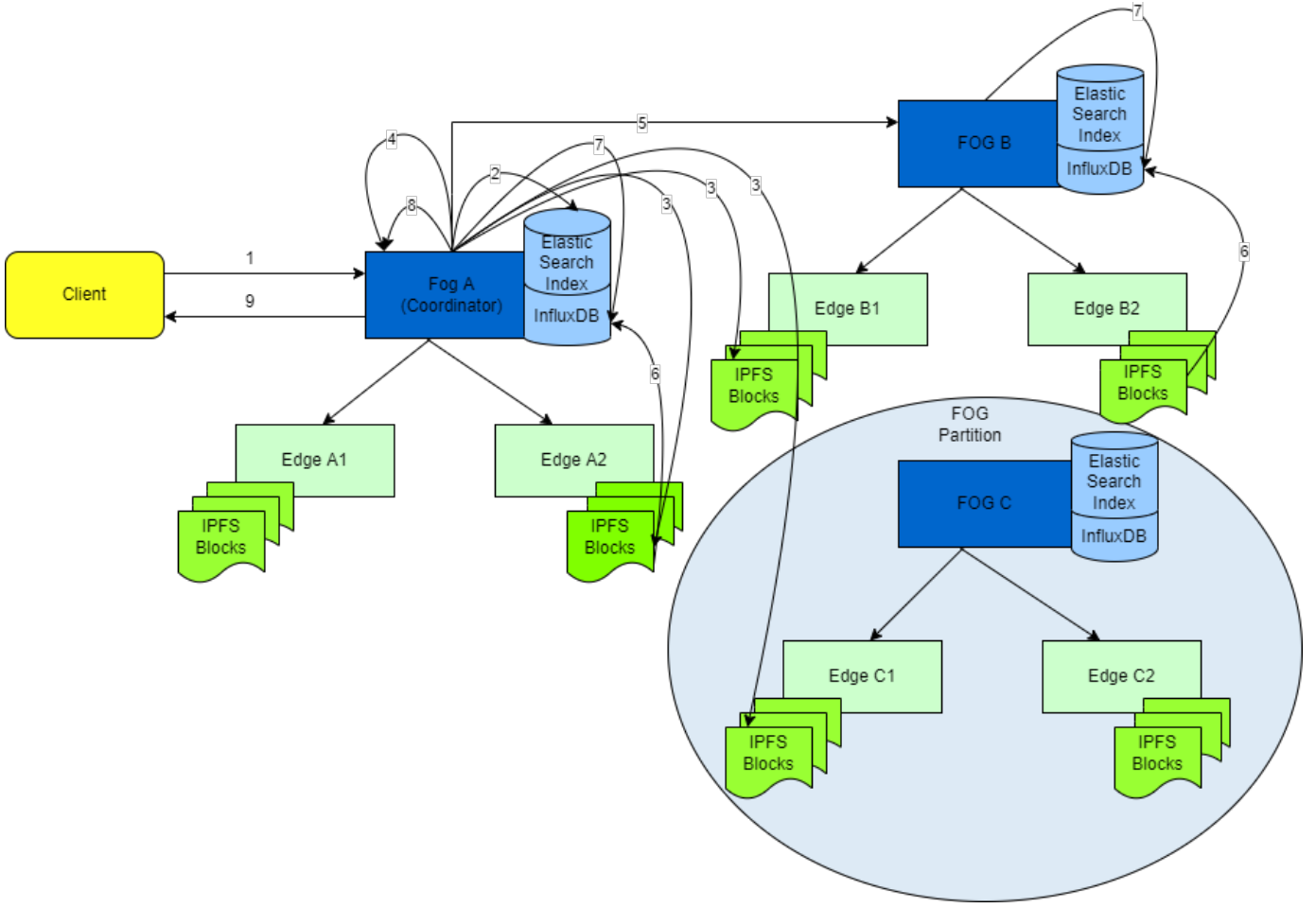


Fig. 1. High level Architecture

data ingestion, enrichment, storage, analysis and visualization. And elastic search is a component of ELK stack

So in our case ElasticSearch indexes based on the meta-data. And when a client queries, the searching is done based on the meta-data. First step will be to elastic search the index using the meta-data requested.

#### IV. PROBLEM DEFINITION

The focus of this project is to modify the existing architecture of TorqueDB.

The existing architecture of TorqueDB contains edge and fog resources. Each edge is associated with one parent fog and that parent fog acts as a network gateway to other fogs and Internet.

TorqueDB uses ElfStore to manage the input data accumulated in blocks hosted by Edge devices. ElfStore stores data in blocks, where each block contains rows of time-series data, typically from one or more sensors and for a specific timing range. And it allows application-specific metadata properties to be stored for these blocks and searched

upon.

ElfStore services manage the edge devices, replication and block placement and also maintain indexes on the metadata for blocks each fog resource to execute Flux queries.

Now in our project we are replacing the ElfStore with IPFS (Interplanetary File System). IPFS service will handle the replication, block placement of the input data. But to maintain indexes on the metadata for blocks, we need to implement ElasticSearch. ElasticSearch will handle the indexing and querying on the metadata for blocks.

So, we need to first decouple the ElfStore component from TorqueDB.

We need to use IPFS to store the input data generated by edge devices and then implement ElasticSearch for forming index on the metadata.

Then we need to design an efficient query model. So, when a client sends a query, the fog co-ordinator executes the queries using ElasticSearch and returns the result to the client.

## V. PROPOSED ARCHITECTURE

We have given a proposed high-level architecture diagram as shown in Figure 1.

Following are the label descriptions:

- 1- Flux Query.
- 2- Elastic search on metadata index.
- 3- Fetch and query metadata.
- 4- Plan L2 query.
- 5- Send L2 blocks and Sub-query.
- 6- Fetch blocks from IPFS and insert into InfluxDB.
- 7- Local Influx Query.
- 8- Aggregate results.
- 9- Return results.

The system model contains edge and fog devices, where each edge device is associated to a parent fog device. This parent fog acts a network gateway to other fogs and the Internet.

A fog partition is formed by all edge devices and the parent fog device. Also, devices in a partition are part of the same private network, with high bandwidth and low latency connectivity.

The input data accumulated from sensors is hosted by the Edge devices and is managed by storing in form of blocks across nodes using IPFS. Each block is identified by a unique block ID and contains a row of time-series data, typically from one or more sensors and for a specific time range. Over the time new blocks are added. IPFS allows to store all the metadata in a decentralized, distributed file system. Using IPFS, we can store application-specific metadata properties to be stored for these blocks and searched upon. These contains details such as the table name, sensor ID, sensor types, units, etc. A subset of these properties matches specific columns present in the timeseries data, e.g., the location and the sensor ID column values may be common to all rows in the block, which are surfaced as a property for that block, while the minimum and maximum timestamps for the rows in the block will form the time-range property for that block. In IPFS the content doesn't automatically get replicated across the entire network. But IPFS does provide replication feature, so the reliability requirements are achieved.

We also host an InfluxDB instance on each fog device to execute flux queries. InfluxDB is mainly used as a query engine. It is a transient store for the time-series data on which complex flux queries are executed.

Edge devices runs the IPFS services to manage replication and block placements. After block placement of data, indexing is performed based on a key-value pair using ElasticSearch, where the key is the metadata which is formed using the key contents of the blocks and value is the list of Block-ids

representing those keys.

### A. Query Lifecycle

Users submit their Flux query to a TorqueDB service that runs on all fog devices. Fog co-ordinator will receive the query and attempts to identify the IPFS blocks that contain the time-series data on which the query depends. Then the fog co-ordinator queries the metadata in the index and search for the blocks with the given property value. After getting the required Block-ids, the query is performed on the edges that contains that particular blocks and the content is fetched and the fog devices store the block contents into their local InfluxDB instance and execute the query submitted by the Client/User.

The mapping of blocks to fogs is done by the Query planner.

## VI. OUR APPROACH

1. We first install and configure IPFS
  2. Initialize IPFS using init command
  3. Load the directory to be stored into IPFS
- IPFS stores all the directory files by splitting each file into smaller chunks and returns a hash key called a content identifier
4. Retrieve the hash keys for each file and store in a txt file.
  5. Use LogStash to form an index on metadata by writing a config file
  6. Query the index using ElasticSearch
  7. Visualize using Kibana

After this, integrate the IPFS to TorqueDB and modify the functions integrate the query mechanism, *readEdge()*, *find-BlocksAndLocationsWithQuery()* and *getMetadataByBlockid()*

## VII. FUTURE WORK

- Now that we have the basics down of how to integrate Elasticsearch with IPFS, we can now make necessary changes in the logstash bul push config file.
- The first change is to make id of Elasticsearch as the metadata of the file itself i.e. instead of indexing the metadata, we will index with the metadata.
- Append the value field to have "ipfs get" in front of the hash key and make it auto execute.
- This means that the user can search files in Elastic search using file metadata and in return, the search will initiate a retrieve file with that metadata and get the required files ready for further work in InfluxDB.

## VIII. ACKNOWLEDGEMENT

This project has been proposed for the coursework DS256: Scalable Systems for Data Sciences under the guidance of Dr. Yogesh Simmhan (Associate Professor, Department of Computational and Data Sciences, IISc Bangalore, India).

- <https://www.knowi.com/blog/what-iselastic-search/>
- <https://en.wikipedia.org/wiki/InterPlanetaryFileSystem>
- Dhruv Garg, et al.: TorqueDB: Distributed Querying of Time-series Data from Edgelocal Storage, Euro-Par 2020, <https://doi.org/10.1007/978-3-030-57675-218>