DataFog: Towards a Holistic Data Management Platform for the IoT Age at the Network Edge

Harshit Gupta
Georgia Institute of Technology

Zhuangdi Xu
Georgia Institute of Technology

Umakishore Ramachandran
Georgia Institute of Technology

Abstract

The increasing adoption of the Internet of Things (IoT) paradigm in structuring services for "smart cities" has led to an explosion of data generated, ingested, processed, and stored in the system. Traditional time-series databases being Cloud-based are not suitable for meeting the low-latency requirement of such geo-distributed smart services. We propose *DataFog*, a geo-distributed data-management platform at the edge of the network to cater to the needs of smart services for the IoT age. We identify key challenges towards building such a database over a widely geo-distributed and heterogeneous edge computing environment. Preliminary evaluations show the performance potential of *DataFog* in comparison to state-of-the-art distributed data-stores. *DataFog* is the first system, to the best of the authors' knowledge, that is meant for data-management at the network edge.

1 Introduction

Smart services based on the Internet of Things (IoT) infrastructure has increased the need for scalable data management platforms. Recently there have been industry proposals for managing IoT data using (cloud-based) platforms, for efficient analytics and also for monetizing the use of data by 3rd parties [3]. Situation awareness applications running on edge computing platforms utilizing the IoT infrastructure have tight low-latency requirements between sensing and actuation. Often such applications need timely data about the state of the physical environment - that too in the critical path of the application logic. Cloud-based data management platforms are incapable of meeting the timeliness requirements of such applications due to the inevitable high latency incurred due to wide-area network traversals. Further, with the ever increasing deployment of bandwidth intensive IoT platforms (especially cameras), there is an increasing pressure on the backhaul bandwidth to transport data back and forth between the edge and the Cloud. Both these factors point to the need for more efficient management of the data and the computation on the data at the edge of the network (i.e., close to the IoT platforms that are the sources for such data).

Building a datastore on an edge computing infrastructure, however, has its own set of peculiar challenges. The wide geo-distribution and heterogeneous nature of this infrastructure requires data-partitioning and replication policies that are commensurate with the latency requirements of applications. Traditional cloud-based datastores (such as Cassandra) do not differentiate between data items and partition them evenly across all the nodes for better load-balancing. In edge computing environments, where inter-node communication may have to traverse the Internet, an even partitioning of data is detrimental to the latency of queries. Moreover, the servers at the network edge are not as resource rich as their Cloud counterparts. Thus a data management platform that is edge friendly should simultaneously take advantage of the edge servers for low latency and the Cloud for abundance of resources. In this paper, we propose *DataFog*, a solution to these issues by designing a system that performs data partitioning between the edge and the Cloud based on contextual relevance of data-items in space and time. In the scope of this paper, we assume that all datastore nodes would be owned or rented by the same organization. Extending DataFog for federated operation across clusters managed by multiple parties is a prospective future work. In this paper, we make the following contributions as algorithmic insights of *DataFog*:

- A distributed indexing mechanism that performs data placement (both among edge nodes, and between the edge and the Cloud) based on spatio-temporal attributes to support efficient queries involving multiple edge nodes.
- A replica placement approach that provides both spatial proximity and resilience to geographically-

correlated failures [7].

- A location-aware load balancing policy to deal with sudden storage usage surges and hotspots by utilizing nearby edge nodes.
- A data-eviction and data-compression strategy based on temporal relevance that takes into account the low-storage capacity of edge nodes

The rest of paper is organized as follows. Section 2 describes the requirements of an edge-friendly data management platform using two motivating use cases. Section 3 describes the design of *DataFog*. Section 4 presents performance improvements of *DataFog* through results of preliminary evaluations. Section 5 summarizes the related work on data management for IoT and Section 6 concludes the paper.

2 Requirements of situation-awareness applications

Smart-cities equipped with multi-modal sensing functionalities generate data continuously by sensing the environment. Raw data streams, e.g. video frames from smart cameras, need to be pre-processed to detect interesting objects or patterns, e.g. pedestrians or cars, to generate streams of events, which could in turn be used by applications as well. We describe two candidate applications to motivate the necessity of a data-management platform at the network edge.

- **Suspicious vehicle tracking** : A distributed camera network deployed on urban roadways generates streams of vehicle detections for continuous tracking of suspicious vehicles. Tracking vehicles in real time requires spatio-temporal range queries such as *select all vehicle detections within 5 km and 10 minutes* to be efficient. The distribution of workload is dependent on the distribution of vehicles in space, often leading to hotspots. Furthermore, for continuous operation, the camera network generates continuous streams of vehicle detections that need to be saved in a datastore.
- **IoT to improve quality of life for people with disabilities** : Information generated from smart cameras has the potential to help people with navigational challenges - who due to psychological issues prefer paths with less crowd density [8]. The navigation application needs to gather crowd density of nearby regions so as to navigate the person through the best possible path. Such a density estimate can be constructed using spatio-temporal range queries for recent measurements of crowd density at locations close to the user.

Note that the use-cases described above consume data generated by specific IoT subsystems (smart surveillance) for performing their functions. Based on these use-cases we present the key characteristics of applications that would benefit from *DataFog*.

1. **Spatio-temporal locality in range queries** : For each query, there is a spatio-temporal region such that data-items belonging to that region are *more relevant* than other data-items. For example, in the vehicle-tracking use-case mentioned earlier, the region of relevance for a range query would include all events within 5 km and within 10 minutes from the current detection. This region is relevant for real-time operations of an application, while on the other hand batch processing operations require data from a large area and over a long period of time.
2. **Data-model** : *DataFog* supports data-items having a field describing the type of data, location and timestamp fields and the value. Such a data-model allows *DataFog* to handle both continuous streams emitted by static sensors (e.g. crowd density monitoring) as well as streams concerning moving objects (e.g. vehicle tracking).
3. **Continuous generation of data** : Data is generated at all times, thus posing a challenge to the data management platform to cater to the low storage capacity on edge-based nodes.
4. **High availability requirements** : Applications in the IoT space interact with the environment and perform critical tasks. For reliable operation, the datastore needs to be tolerant to failures, especially geographically correlated failures.

3 Algorithmic Insights for a Geo-distributed Datastore

Based on the peculiar characteristics of a geo-distributed edge computing infrastructure and applications characteristics mentioned above, we take the following design decisions that will overcome the pitfalls of off-the-shelf platforms.

3.1 Locality-aware distributed indexing

Based on the requirements of spatio-temporal locality in range-queries, low-latency response is dependent on the placement of data-items on datastore nodes in proximity to clients. Data-items are indexed based on their spatio-temporal attributes. We propose to use spatio-temporal encoding techniques (e.g. ST-Hash [9]), or a combination of spatial encoding (e.g. Geohash, Hilbert's encoding [13]) and consistent hashing, for using the location

timestamp, and item-type attributes for partitioning data across nodes.

3.2 Replication policy

Typical data replication serves two main purposes : load-balancing and fault-tolerance. We intend to create multiple replicas both on edge nodes for low-latency as well as on remote datacenter nodes for tolerance from geographically correlated failures.

The replication policy of *DataFog* should take into account the requirements of replication policies in contemporary data-stores (e.g., Cassandra) like cross-rack and cross-datacenter replication (for fault-tolerance). Furthermore, due to the latency-sensitivity in edge computing environments, we incorporate a constraint that dictates the maximum distance between a replica node and the location of data-item and can be specified by the developer based on latency requirement of the application.

3.3 Handling workload skews

DataFog's data partitioning ensures good load-balancing against skews in application workload. Its design is a hybrid one, that provides both spatial-proximity and even distribution of data placement. *DataFog* allows the application developer to configure load-balancing regions, such that all nodes belonging to the same region would balance workload among each other. The application-developer is allowed to specify the size of this region, based on the suitable tradeoff between latency and load-balancing.

We plan to develop two mechanisms for adapting to hotspots. For long-lived hotspots, the administrator would launch and attach new datastore nodes to the running cluster. *DataFog* ensures that the newly added nodes balance load from the heavily loaded ones and bring down the average per-node traffic. For short-lived hotspots, we propose to develop an offloading mechanism that would allow heavily loaded nodes to offload a specific portion of its data-items to a lightly loaded node.

3.4 Handling scarce resources at the edge

Resource capacities at the network edge are typically smaller than those in the cloud. *DataFog* should leverage the proximal location of edge nodes, but also take into account their resource scarcity[1]. To this end, we make the following design decisions for lowering storage use on edge nodes.

[1]DataFog would be implemented by extending Cassandra, and thus inherits its resource requirements. Cassandra's throughput improves with higher cores and RAM and faster disks, and the developer can choose resource configuration based on the applications needs. However, a minimal edge node would require 2 cores and 8 GB of RAM.

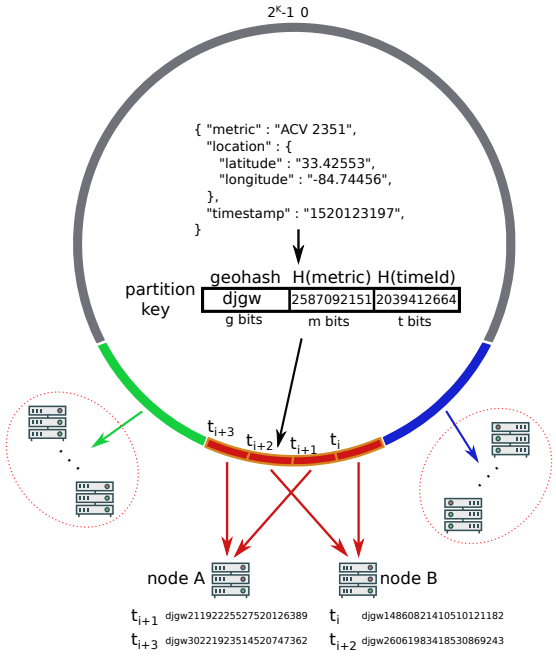

Figure 1: This figure exemplifies load-balancing in a Cassandra-specific setting. It shows three adjacent load-balancing regions, colored by green, red and blue on the token hash ring. Data-items falling in the red region are uniformly partitioned across 4 logical (virtual) nodes ($t_{i-1} - t_i$, $t_i - t_{i+1}$, $t_{i+1} - t_{i+2}$ and $t_{i+2} - t_{i+3}$), which are mapped to 2 physical nodes (A and B). Given a data item, we use the location to construct a spatial encoding and concatenate it with the hash of metric type and hash of timestamp to get the partition key. Then we locate the position of the partition key on the ring and store the key-value pair on the virtual node that covers the corresponding position. In this example, the data-item's partition-key falls between t_{i+1} and t_{i+2}, therefore it is stored on the datastore node B (which owns the region between t_{i+1} and t_{i+2}).

3.4.1 TTL-based data eviction

The dependence of real-time analytics on temporally local data renders records older than a certain application-dependent threshold irrelevant for those queries. However, batch-processing use-cases require data spanning over a large period of time to detect slow-moving patterns. *DataFog* uses a time-to-live based approach to evict stale data from the edge-nodes, while maintaining replicas in cloud-based nodes. Hence a range query spanning a large interval of time would have to retrieve all concerned data-items from the cloud-based replicas. This prevents accumulation of stale data on constrained edge-nodes, as well as makes complete time-series data available for batch-processing use-cases.

3.4.2 Data aggregation and compression

Time series data from sensors is generated periodically and piles up quickly. If each data item (for example, temperature) owns a unique token key, it is likely the metadata will occupy a large portion of storage space and lead to inefficient storage utilization. By aggregating data-items from the same data source, *DataFog* is able to save storage space by omitting redundant metadata and also improve the memory locality for range queries. Meanwhile, time series data items from the same data source are usually isomorphic, which leverages the opportunities to further improve the storage utilization by properly compressing both the metadata and the actual data items [12]. For example, the air temperature data only fluctuates within a small range, so there is no need to use a full 64-bit integer for each data item, and the difference between subsequent values can be stored.

3.5 Sketch of System Implementation

As a first approach towards implementing *DataFog*, we would extend Apache Cassandra with the aforementioned design decisions. Cassandra uses the hash of data-item's key to search the hash ring for replicas (see Fig 1). In *DataFog* the hash function is modified to incorporate spatio-temporal indexing (Section 3.1). A load-balancing region is composed of data nodes with the same GeoHash prefix (similar positions on hash ring) which forms an arc on the hash ring (Fig 1). When a load-balancing region is overloaded, application developer can choose to bring in new data nodes or distribute workload across existing nodes. In either case migration of data is involved, where we would leverage the built-in data migration mechanism of Cassandra [2]. A gossip protocol is used to exchange information between nodes and detect failures and network partitions. Network partitions are handled by caching the writes to unavailable nodes and later replaying them when those nodes become available (hinted-handoff). TTL-based data eviction extends Cassandra's existing mechanism to classify data as expired. Data aggregation and compression will be performed as in time-series database engines such as InfluxDB [1].

4 Preliminary Evaluation

The evaluation of *DataFog* is intended to delve into the design choices presented in Section 3 and study their effectiveness. In the interest of brevity, we present the results of a preliminary evaluation of a spatio-temporal context-aware data partitioning and replication policy. A more thorough evaluation of *DataFog* is part of future work and discussed in Section 6. We compare the performance of *DataFog*'s data partitioning policy against the consistent-hashing based policy of off-the-shelf Cassandra. We use the vehicle-tracking use-case described in Section 2 to generate query workload and quantify the performance of spatio-temporal range-queries for recent detections of a particular vehicle.

4.1 Workload description

The objective of the vehicle-tracking application is to construct trajectories of vehicles in real-time. We use SUMO [11] to simulate the movement of vehicles on the road network in Georgia Tech campus equipped with 35 smart cameras. These cameras, upon vehicle detection, generate an event with the location, timestamp and license number of the vehicle. For each vehicle detection by a camera, the application submits a range query for all vehicle detections of that vehicle within a 5 kilometre radius that happened less than 10 minutes ago. These previous detections are used to construct vehicle's recent trajectory. The application's performance is contingent on efficient range-query execution. The datastore to store these events and trajectories comprises of 4 nodes within the campus itself and remote nodes in 4 geographically distant regions of the USA (CA, WA, IL and FL). We emulate these nodes and corresponding network topology using MaxiNet [17] on Microsoft Azure.

4.2 Implementation details

We extend Cassandra to build this first prototype of *DataFog*. To enable context-aware data partitioning, we assign a token to each datastore node, which is composed of a Geohash encoding of the node's location at precision 3 followed by a random byte-sequence [3]. The choice of Geohash precision has been made that all the local nodes (inside Georgia Tech) fall in the same load-balancing region of the hash ring. For simplicity of implementation, we add an additional field called *part_key* to each data-item that is composed of the data-item's contextual information (as in Fig.1), and is used for data partitioning. Using this additional field can easily be avoided by implementing a custom context-aware partitioner [4].

```
CREATE TABLE vehicle_detections (
geohash VARCHAR, ts_chunk BIGINT,
metric_type VARCHAR, ts_offset BIGINT,
part_key VARCHAR,
PRIMARY KEY (part_key, geohash, ts_offset));
```

[2] The reader is referred to *repair* and *rebuild* operations of Cassandra. `https://docs.datastax.com/en/cassandra/3.0/cassandra/tools/toolsRepair.html`

[3] Auto-token generation is part of future work

[4] `https://docs.datastax.com/en/cassandra/2.1/cassandra/architecture/architecturePartitionerAbout_c.html`

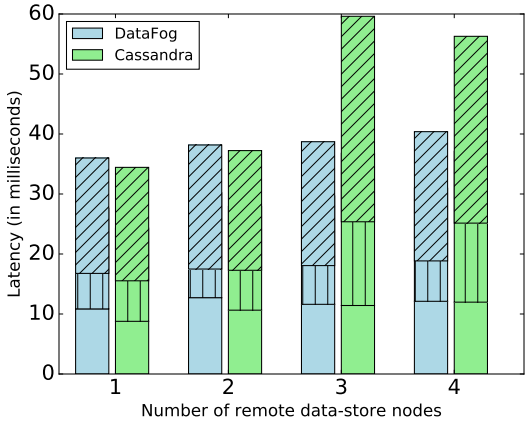

Figure 2: The 50^{th}, 95^{th} and 99^{th} percentile range-query latencies with varying number of remote datastore nodes.

The datastore implementation allows millisecond precision for timestamps, and for partitioning events, divides time into 1 hr long chunks - such that any timestamp can be broken down into a chunk with ID *ts_chunk* and an offset ts_offset. The partition key field *part_key* is composed of the GeoHash of record's location, followed by a consistent hash [5] of the *ts_chunk* field followed by a hash of the *metric_type* field (see Fig. 1).

We compare the performance of the preliminary location-based indexing approach to a location-agnostic indexing done by off-the-shelf Cassandra. The replication factor is set to 3. To show the drawback of location-agnostic indexing, we increase the size of the cluster by adding more remote data-store nodes. The latency of spatio-temporal range queries against these replication settings is shown in Figure 2. We expect the range-query latency to increase with an increase in the number of remote nodes, due to off-the-shelf Cassandra's even data partitioning. When the number of remote nodes is less than the replication factor (3), one of the replicas would always be placed on a local node. However, when the number of remote nodes is 3 or greater, some data-items end up having replicas only on remote nodes making the higher percentiles of latencies becoming higher. *DataFog*, on the other hand, performs location-aware placement of replicas, and hence is not affected by the increase in cluster size.

5 Related Work

The continuous increase in the amount of telemetry data has led design of time-series databases becoming an active area both in research and industry, with a plethora of solutions [15, 4, 2, 14]. Hughes et al. [10] designed GeoMesa, a distributed architecture for spatio-temporal data, which indexes data based on GeoHash and has been built by extending Apache AccumuloDB. The design of indexing keys is similar to *DataFog*, wherein they incorporate spatial encoding and timestamp by interleaving them. Von et al. [16] discuss the design of a database for storing events concerning objects that change state in space and time using a key-value store HBase. They present the indexing design and highlight important future work. These solutions, however, are built for operation in cloud-based environments, where inter-node latency is significantly smaller than that in edge-computing environments. The concern for having edge-based datastores has been pointed out by Confais et al. [6]. They conduct a performance evaluation of three off-the-shelf object stores (Cassandra, Rados and Inter-Planetary File System) in deployed over an edge-computing infrastructure. However, their analysis compares off-the-shelf object stores and they do not make any design decisions suited for the edge computing environment.

6 Conclusion

In this paper we present the case for a holistic management platform for IoT data on at the network edge. We identify the challenges of edge infrastructure and come up with algorithmic insights for addressing them so as to leverage the benefits of edge-based deployment. Implementation of solutions leveraging those insights and comparing the possible choices from the solution space forms an immediate future work. *DataFog*'s potential is demonstrated by the performance improvement due to a replica placement approach based on spatial locality. Our immediate future work is a thorough quantitative evaluation of the design decisions in comparison to state-of-the-art Cloud-based datastores.

The overheads of context-aware partitioning and replication needs to be analyzed. We plan to measure the ability of our load-balancing solutions to manage workloads with inherent skews. We would demonstrate the benefit of eviction-based strategy on utilization of storage resources at the edge. The reduction of storage consumption through data compression and aggregation is another metric that would quantify the performance of *DataFog* for constrained edge devices. Finally, we would tune the parameters in the aforementioned policies (e.g. replication distance, spatial encoding precision, etc.) and measure their impact on overall performance. Another important issue is the interaction between datastore platforms owned by different stakeholders [18]. There is a need for communication protocols and business models for sharing data across multiple edge administrative domains.

References

[1] Influxdb. https://www.influxdata.com/. Accessed: 2018-05-21.

[2] Influxdb: The good, the bad, and the ugly. https://bluefox.io/2017/09/07/influxdb-good-bad-ugly/. Accessed: 2018-03-19.

[3] Nokia Sensing as a Service existing assets, new revenue streams. https://networks.nokia.com/services/sensing-as-a-service. Accessed: 2018-03-14.

[4] Spotify : The heroic time series database. https://spotify.github.io/heroic. Accessed: 2018-03-25.

[5] Murmurhash — wikipedia, the free encyclopedia, 2018. [Online; accessed 28-March-2018].

[6] CONFAIS, B., LEBRE, A., AND PARREIN, B. Performance analysis of object store systems in a fog and edge computing infrastructure. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXIII*. Springer, 2017, pp. 40–79.

[7] COUTO, R. S., SECCI, S., CAMPISTA, M. E. M., AND COSTA, L. H. M. Latency versus survivability in geo-distributed data center design. In *Global Communications Conference (GLOBECOM), 2014 IEEE* (2014), IEEE, pp. 1102–1107.

[8] GONG, J., FEELEY, C., TANG, H., OLMSCHENK, G., NAIR, V., ZHOU, Z., YU, Y., YAMAMOTO, K., AND ZHU, Z. Building smart transportation hubs with internet of things to improve services to people with disabilities. In *Computing in Civil Engineering 2017*. pp. 458–466.

[9] GUAN, X., BO, C., LI, Z., AND YU, Y. St-hash: An efficient spatiotemporal index for massive trajectory data in a nosql database. In *Geoinformatics, 2017 25th International Conference on* (2017), IEEE, pp. 1–7.

[10] HUGHES, J. N., ANNEX, A., EICHELBERGER, C. N., FOX, A., HULBERT, A., AND RONQUEST, M. Geomesa: a distributed architecture for spatio-temporal fusion. In *Geospatial Informatics, Fusion, and Motion Video Analytics V* (2015), vol. 9473, International Society for Optics and Photonics, p. 94730F.

[11] KRAJZEWICZ, D., HERTKORN, G., RÖSSEL, C., AND WAGNER, P. Sumo (simulation of urban mobility)-an open-source traffic simulation. In *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)* (2002), pp. 183–187.

[12] PELKONEN, T., FRANKLIN, S., TELLER, J., CAVALLARO, P., HUANG, Q., MEZA, J., AND VEERARAGHAVAN, K. Gorilla: A fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment 8*, 12 (2015), 1816–1827.

[13] SAGAN, H. Hilberts space-filling curve. In *Space-filling curves*. Springer, 1994, pp. 9–30.

[14] SIGOURE, B. Opentsdb: The distributed, scalable time series database. *Proc. OSCON 11* (2010).

[15] SIOW, E., TIROPANIS, T., WANG, X., AND HALL, W. Tritandb: Time-series rapid internet of things analytics. *arXiv preprint arXiv:1801.07947* (2018).

[16] VAN LE, H. Distributed moving objects database based on key-value stores. In *PhD@ VLDB* (2016).

[17] WETTE, P., DRAXLER, M., SCHWABE, A., WALLASCHEK, F., ZAHRAEE, M. H., AND KARL, H. Maxinet: Distributed emulation of software-defined networks. In *Networking Conference, 2014 IFIP* (2014), IEEE, pp. 1–9.

[18] ZHANG, Q., ZHANG, X., AND SHI, W. Firework: Big data processing in collaborative edge environment. In *Edge Computing (SEC), IEEE/ACM Symposium on* (2016), IEEE, pp. 81–82.