Apache IoTDB: Time-series Database for Internet of Things

Chen Wang[1,2], Xiangdong Huang[1*], Jialin Qiao[1], Tian Jiang[1], Lei Rui[1], Jinrui Zhang[3], Rong Kang[1], Julian Feinauer[4], Kevin A. McGrail[5], Peng Wang[6], Diaohan Luo[1], Jun Yuan[1], Jianmin Wang[1], Jiaguang Sun[1]

[1] School of Software, Tsinghua University, [2] EIRI, Tsinghua University, [3] Microsoft, [4] Pragmatic Industries GmbH, [5]InfraShield.com, [6] Fudan University

huangxdong@tsinghua.edu.cn

ABSTRACT

The amount of time-series data that is generated has exploded due to the growing popularity of Internet of Things (IoT) devices and applications. These applications require efficient management of the time-series data on both the edge and cloud side that support high throughput ingestion, low latency query and advanced time series analysis. In this demonstration, we present Apache IoTDB managing time-series data to enable new classes of IoT applications. IoTDB has both edge and cloud versions, provides an optimized columnar file format for efficient time-series data storage, and time-series database with high ingestion rate, low latency queries and data analysis support. It is specially optimized for time-series oriented operations like aggregations query, down-sampling and sub-sequence similarity search. An edge-to-cloud time-series data management application is chosen to demonstrate how IoTDB handles time-series data in real-time and supports advanced analytics by integrating with Hadoop and Spark. An end-to-end IoT data management solution is shown by integrating IoTDB with PLC4x, Calcite, and Grafana.

PVLDB Reference Format:

Chen Wang[1,2], Xiangdong Huang[1*], Jialin Qiao[1], Tian Jiang[1], Lei Rui[1], Jinrui Zhang[3], Rong Kang[1], Julian Feinauer[4], Kevin A. McGrail[5], Peng Wang[6], Diaohan Luo[1], Jun Yuan[1], Jianmin Wang[1], Jiaguang Sun[1]. Apache IoTDB: Time-series Database for Internet of Things. PVLDB, 13(12): 2901 - 2904, 2020.

doi:10.14778/3415478.3415504

1 INTRODUCTION

Nowadays, IoT applications are becoming increasingly popular in many areas. Examples can be found in consumer electronics including smart home devices, wearables and connected healthcare as well as in industrial applications with the rise of Industrial IoT (IIoT). Compared to traditional time-series usage for IT such as infrastructure monitoring, the major characteristics of these IoT applications are real-time data management with lower latency and more advanced analytics on the time-series datasets. Furthermore, when IoT is used in industrial applications, intelligent equipment usually produces one to two orders of magnitude more data than consumer-oriented IoT devices. This makes it even harder for analytics to produce valuable insights in a reasonable amount of time. As an illustrative example, a single wind turbine can generate hundreds of data points every 20 ms [7] to monitor conditions, detect faults and make decisions. Future operations can then be decided by a set of sophisticated queries against the acquired time-series by data scientists. Typical uses are signal decomposition and filtration, segmentation for different working conditions, and failure pattern matching.

Consequently, the IoT-related service market has spawned new workloads on time-series processing blended by:

Edge computing: As edge devices have gained more computational power and edge computing has grown more popular, managing time-series data and supporting advanced analysis on the edge side is trending. It requires the time-series database to be capable of running on both edge and cloud side, while remaining well organized for data synchronization.

Long-life, large volume historical data: The volume of data in IIoT is large. For example, the sensors on a Boeing model 787 airliner produce upwards of half a terabyte of data per flight [8]. Compared with data center monitoring applications where the data is kept for a week or month, industrial users usually choose to keep all historical data for audit and statistical analysis of the whole life cycle of devices.

High throughput data ingestion: As illustrated in the wind turbine example, the database needs to handle the ingestion of tens of millions of time-series data points per second stably in a 24×7×365 manner. It becomes more challenging when the arrival of time-series data cannot be guaranteed to be in order due to various device and network problems including device failure, weak communication signal or network congestion.

Low latency and complex queries: Queries are typically used in three scenarios. (1) The value of the latest data point is required for real-time monitoring with a short on boarding interval. (2) Applications, like those for fault detection, regularly retrieve time-series data having a timestamp or time window filter for given time-series IDs, and the results are ordered by time. (3) The interactive, exploratory queries by data scientists are more complicated and unpredictable, where conditions on value and similarity of sub-sequence are applied on arbitrary lengths of historical time-series.

Advanced data analytics: Besides queries, advanced IoT data analytics like signal processing and machine learning algorithms are also necessary for data scientists to process the historical data. However, the support by big data ecosystems such as Apache Spark requires ETL from time-series database and keeping two costly copies of huge historical data respectively.

Time-series databases, like OpenTSDB [9] and KairosDB [2], are built on top of existing NoSQL stores but suffer from insufficient

Proceedings of the VLDB Endowment, Vol. 13, No. 12 ISSN 2150-8097.
doi:10.14778/3415478.3415504

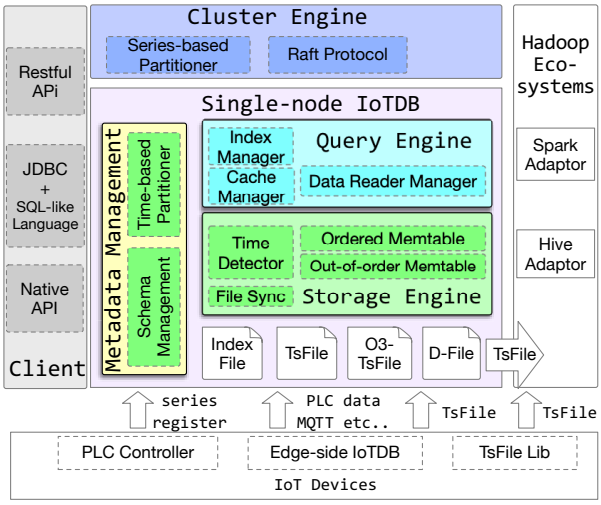

Figure 1: Main modules of IoTDB

performance and poor compression for IoT workloads. InfluxDB has about 10x improvements on these aspects [5], but still has a gap on query performance especially when aggregating large amounts of historical data. In practice, Parquet [3] is the choice for time-series storage for analytics and OLAP workloads with Hadoop ecosystem, but it requires ETL from the time-series database. Moreover, optimization is needed to support native time-series data for both storage and query efficiency beyond columnar storage.

In this demo, we introduce Apache IoTDB [1], a native time-series storage format and database for both edge and cloud computing. It has the following key features: (1) IoTDB has a lightweight architecture running on the edge appliances, and a cluster version for the data center under the same code base, as well as efficient data synchronization from edge to cloud. (2) IoTDB provides a novel columnar file format, called TsFile (Time-series File), as the main storage format to optimize the data organization, size reduction, and query performance with time-series data. (3) IoTDB supports high throughput ingestion by an elaborate buffering design and storage strategy to handle frequent out-of-order data ingestion and sorted data query. (4) IoTDB leverages metadata in TsFiles and index files to support low latency queries and complex similarity search. (5) TsFile, as the file format of IoTDB, can also be accessed directly in Hadoop ecosystem by Spark and Hive for data analysis.

Currently, IoTDB supports the ingestion rate up to 30 million data points per second on a single node, and the latency of hundreds of milliseconds for raw data queries and tens of milliseconds for aggregation queries on billions of data points. A more comprehensive functional and experimental evaluation can be accessed in public [4]. IoTDB has been deployed in the production environment by several industry users.

2 SYSTEM OVERVIEW

The architecture of IoTDB is shown in Figure 1. IoTDB is designed to manage huge volumes of time-series data points from IoT devices, where one data point is logically depicted as (<*device, sensor, timestamp, value*>. Herein the device and sensor identifiers together present a unique time-series ID. The *Metadata Management* module manages the naming space of devices with a tree structure. For instance, Location1.Windfarm2.Manufactuer3.Turbine4 is a full path to describe a single wind turbine. The design of IoTDB chooses to store the data in an open native time-series file format for both database access with *Query/Storage Engine* and Hadoop/Spark access against a single copy of the data. It also serves as a distributed time-series database, where data is partitioned by grouping of time-series in *Cluster Engine* among different nodes while time-based data slicing is implemented on each node to improve the performances. IoTDB provides an SQL-like language, native API, and restful API to access the data. We then introduce the main features in the following subsections.

2.1 Uniform Edge-Cloud Design

In IoT scenario, edge computing and cloud side deployment are equally important. Therefore, IoTDB is designed to fit three deployment models: 1) file-based storage or embedded time-series database on edge appliance like Raspberry PI, 2) standalone time-series database on Industrial PC and 3) distributed time-series database or Hadoop cluster with TsFile storage format.

Typically, IoT devices collect data from sensors and industrial controllers, and send data to data center using customized or standard protocols like MQTT in real-time. However, in some cases, the edge intelligence requires real-time analytics, such as fault alerts, to retrieve data from a local data store. Therefore, IoTDB has a lightweight, embedded version to be deployed on the IoT devices, where the minimal runtime memory requirement is 32MB and computation is supported with an ARM7 processor. Local storage is also mandatory to prevent data loss in case of the temporary network outage. In this scenario, *TsFile Lib* allows the devices to persist data in TsFile format, and afterwards the generated TsFiles can be directly synchronized and merged with active IoTDB instance on the cloud using the *File Sync* module.

On the cloud side, using the *Cluster Engine*, a raft-based protocol is implemented to manage multiple IoTDB nodes. In the cluster mode, data partitions can be defined according to both time slice and time-series ID. The distribution of data and query operations are completely transparent to the end users.

2.2 TsFile Format

TsFile is the primary data file format for time-series data storage in IoTDB. Figure 2 shows the structure of the TsFile. TsFile is similar to Parquet but optimized for time-series data. A TsFile mainly consists of two parts: the data content (Chunks, Pages) and the index. Each chunk stores the data of a time-series for a certain time range. Inside a chunk, the data is split into several pages, which is the fundamental unit of the data storage on the disk. Each page stores data points in a pair of columns, i.e. the timestamps and the value. Timestamps are encoded by second order difference and the value field supports compression algorithms like bitmap, Gorilla, RLE, etc. to save disk space. Snappy is also employed for advanced compression on historical data.

To accelerate the query, the data in the chunks of each time-series is ordered by time in TsFile. In this way, queries with time range filters can quickly skip the chunks out of the given time window.

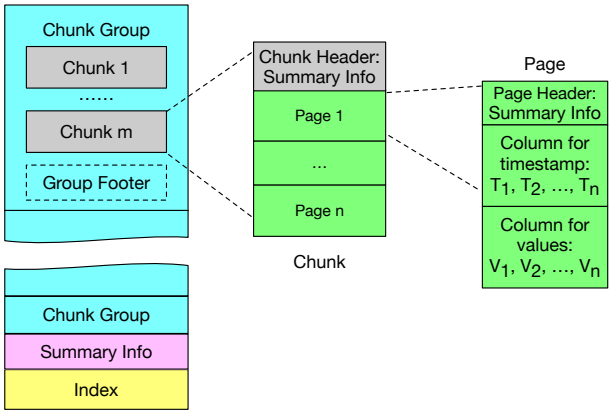

Figure 2: The structure of TsFile

For further query performance improvement, the max/min time, the max/min value, count and average of data points for each series are pre-calculated and stored as *Summary Info* in the level of the file, chunk, and page, which significantly accelerates the filtration and aggregation operations. The TsFile *Index* part is organized at the end of a TsFile, to quickly locate the data chunks of a given time-series. TsFile differs from Parquet and ORC by separately storing the metadata corresponding to each time-series to avoid accessing metadata for unmatched columns.

2.3 Data Ingestion

Apart from the batch file loading, the *Storage Engine* is the main component providing real-time data ingestion. The high throughput ingestion is a key requirement of an IoT database, but out-of-order data presents the major technical challenge. The arrival of data is not guaranteed in order due to many factors like re-transmission because of a temporary device or network failure. However, on the other hand, the time-ordered property of the data inside and among TsFiles is necessary for efficient query processing.

To achieve the high write throughput, the *Storage Engine* buffers data in memory as a memtable first with a write-ahead log. Since the time complexity of finding the ordered position is $O(\log n)$, IoTDB's design gives up the ordering of newly ingested data in the memtable. When flushing a memtable into a TsFile, the ordering and flushing operations are pipelined against different time-series data, therefore the utilization of available CPU time for ordering during I/O helps IoTDB improve the data ingestion throughput further. Another acceleration of ingestion is the vectorized write interfaces in *Storage Engine* in the case of batch arrival data. The pointer of the data array is saved in the memtable, hence memory copy and potential Garbage Collection hereafter is avoided.

In some cases, the late arrival data is out of the time range of memtable. The *Storage Engine* saves such out-of-order data separately into Out-of-Order (O3) TsFiles first and then merges them into TsFiles periodically in the background. Comparing with the normal TsFile, the only difference in O3-TsFile is that the time window of different chunks may overlap, which incurs a merge-sort of multiple chunks against a matched query.

Differential (D) files are used for infrequent update and deletion operations where updates are only for correction in case of data quality issues and deletion is for truncating the data older than a certain timestamp. The log-style records are appended in the versioned D-file, and will be similarly merged to TsFile as O3-TsFile. The query will sequentially scan the D-file to get the latest value, if the target time-series exists.

2.4 Queries

The *Query Engine* takes full advantage of the time-ordered property of TsFiles to reduce I/O and latency for queries with time and value predicates. TsFiles' file, chunk and page level *Summary Info* significantly prune the unnecessary access of TsFiles in IoTDB. For the potential page hit in TsFile, O3-TsFile and D-file, sequential scans will be applied to verify the results. Moreover, the latest data point query is retrieved from the memory *Cache* without accessing a file.

To accelerate interactive and exploratory queries, we proposed PISA [6] index for aggregation and down-sampling queries and KV-match [10] indexes for sub-sequence similarity search. All indexes are managed by *Index Manager*. In IoTDB, the PISA index is built on top of the *Summary Info* of pages by constructing variant of segment trees and forming a forest to support $O(\log n)$ time complexity for aggregation and down-sampling queries while minimizing the loss of the insertion throughput. The leaf and internal nodes in the tree-based index are considered as data points of a time-series and thereby the index file of PISA is in the TsFile format. KV-match index supports sub-sequence matching under either ED or DTW distance while avoiding scanning full time series. When creating a KV-match index on a time-series, the related TsFiles will attach index files one to one while the index file is in the key-value format. Then, users can specify "similar(target time-series, distance bound, start time, end time)" sub-clause to find similar sub-sequences.

Using the above design, the latest data query, which is the most importance for device monitoring, has the extreme low latency as the data is cached in memory. Query against raw data with time filter could also be achieved with low latency with our optimizations, but, for the case of out-of-order data, the query latency may increase duo to the unfinished data compaction.

2.5 Integration with Big Data Ecosystem

IoTDB supports TsFiles persistence on the local file system and HDFS directly. We implement TsFile's HDFS interface for MapReduce, SerDe interface for Hive, and the Data Sources API of Spark to integrate TsFile with MapReduce, Hive, and SparkSQL respectively. TsFile can tell Hive and Spark where the data is for a given time range, what the file schema is, and how to decode the data. In this manner, SparkSQL can push filter conditions down to the *TsFile Lib* to reduce unnecessary disk I/O.

Moreover, IoTDB has been integrated with many other big data open source systems including Flink, RocketMQ, Calcite, PLC4X, and Grafana. The integration in these systems helps to extend IoTDB's capabilities to manage the life cycle for time-series data from data collection to data visualization.

Chen Wang[1,2], Xiangdong Huang[1*], Jialin Qiao[1], Tian Jiang[1], Lei Rui[1], Jinrui Zhang[3], Rong Kang[1], Julian Feinauer[4], Kevin A. McGrail[5], Peng Wang[6], Diaohan Luo[1], Jun Yuan[1], Jianmin Wang[1], Jiaguang Sun[1]

Figure 3: Data management from the edge to the cloud

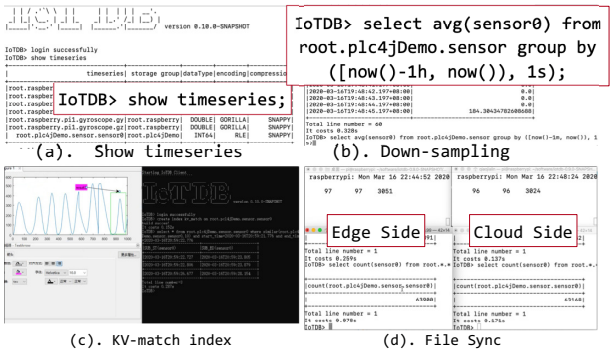

Figure 4: Some basic queries in IoTDB

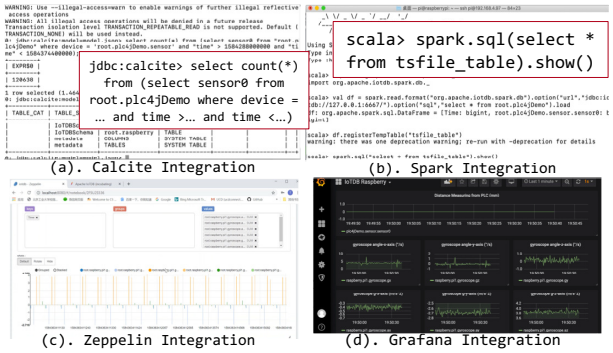

Figure 5: IoTDB integration for advanced functions

3 DEMONSTRATION

IoTDB System: We first demonstrate IoTDB's usage on the edge side. We install a Raspberry PI with a Mitsubishi programmable logic controller (PLC) , an industrial distance measuring sensor and a gyroscope sensor as an intelligent IoT device. An IoTDB is deployed on the Raspberry PI to manage the time-series data locally. The Raspberry PI collects distance-measuring data at a frequency of 100 Hz from the PLC and the data is ingested into IoTDB locally. The edge IoTDB synchronizes the generated TsFiles to the cloud every 10 seconds. The angle changes (x, y, z, accelerated-x, accelerated-y, and accelerated-z) from the gyroscope are collected at a frequency of 5 Hz, IoTDB JDBC is used to send the data to the cloud in real-time. Figure 3 shows the real sensors, the PLC and the Raspberry Pi with IoTDB. When the sensors are moved, we can see the visual time series being updated on the left top screen.

On the cloud, an IoTDB instance receives both the batch TsFiles and the streaming data points in real-time. The effect of the *File Sync* is shown in Figure 4 (d). In the IoTDB-CLI console (Figure 4 (a)), we can see there are 7 time-series in IoTDB. Figure 4 (b) shows that an aggregation query to down-sample the distance measuring data from 100 Hz to 1 Hz. Figure 4 (c) shows using KV-match index to get the most similar sub-sequence from the distance time-series curve when given a sample curve. Figure 4 (a) to (c) are all finished by using IoTDB SQL, while Figure 4 (d) requires running two IoTDB instances and setting the receiver's IP in IoTDB's configuration.

To support advanced analysis, e.g., interactive data exploring and signal computing, we show how to integrate IoTDB with other systems. Figure 5 (a) shows the integration with Calcite to use a nested query to query data from IoTDB. Figure 5 (b) shows using SparkSQL to translate the data in IoTDB to DataFrame and leveraging the capability of DataFrame for complex analysis. In Figure 5 (c), we integrate Zeppelin with IoTDB for exploratory analysis. Figure 5 (d) shows using Grafana to visualize time-series data in IoTDB.

4 CONCLUSION

In this paper, we present Apache IoTDB, a high performance database for time-series data management on the edge and cloud. A native time-series oriented columnar file format, TsFile, is introduced for improved query performance and storage efficiency. More write and query optimizations are also discussed. These contributions are demonstrated with a proof of concept application. This illustrates how time-series data is managed, visualized, explored and analyzed in IoT world using Apache IoTDB.

ACKNOWLEDGMENTS

As an incubating Apache project, there are many contributors devoted to IoTDB. We thank all those who contribute to the community.

REFERENCES

[1] http://iotdb.apache.org. [n.d.]. *Apache IoTDB (incubating)*. http://iotdb.apache.org
[2] http://kairosdb.github.io. [n.d.]. *KairosDB*. http://kairosdb.github.io
[3] http://parquet.apache.org. [n.d.]. *Apache Parquet*. http://parquet.apache.org
[4] https://s.apache.org/tsdb-comparison. [n.d.]. *TSDB Comparison*. https://s.apache.org/tsdb-comparison
[5] https://www.influxdata.com/blog/influxdb-markedly-outperforms-opentsdb-in-time-series-data-metrics benchmark. [n.d.]. *InfluxDB vs OpenTSDB*. https://www.influxdata.com/blog/influxdb-markedly-outperforms-opentsdb-in-time-series-data-metrics-benchmark/
[6] Xiangdong Huang, Jianmin Wang, and et.al. 2016. PISA: An Index for Aggregating Big Time Series Data. In *CIKM*.
[7] IEC 61400-25-6:2016 2016. *Wind energy generation systems - Part 25-6: Communications for monitoring and control of wind power plants - Logical node classes and data classes for condition monitoring*. Standard. International Electrotechnical Commission, Switzerland.
[8] Jussi Ronkainen and Antti Iivari. 2015. Designing a Data Management Pipeline for Pervasive Sensor Communication Systems. In *FNC/MobiSPC*.
[9] B Sigoure. 2010. OpenTSDB: The distributed, scalable time series database. *Proc. OSCON* 11 (2010).
[10] Jiaye Wu, Peng Wang, Chen Wang, Wei Wang, and Jianmin Wang. 2019. KV-match: A Subsequence Matching Approach Supporting Normalization and Time Warping. In *ICDE*.