International Journal of Scientific and Technical Advancements

ISSN: 2454-1532

A Survey on the Data Management in IoT

M. Shona[1], B. N. Arathi[2]
[1, 2]Department of CSE, SVCE, Bangalore, India
Email address: [1]sonasuresh04@gmail.com, [2]aradvg@yahoo.com

***Abstract*—**In the recent years, the Internet of Things (IoT) is considered as a part of the Internet of future and makes it possible for connecting various smart objects together through the Internet. The use of IoT technology in applications has spurred the increase of real-time data, which makes the information storage and accessing more difficult and challenging. This paper discusses the challenges for Data Management systems in the Internet of Things to manage the massive volume of operational data gener-ated by sensors and devices.

***Keywords*—**NoSQL; hadoop; time series daemon (TSD); memSQL.

I. Introduction

A network comprised of physical objects capable of gathering and sharing electronic information. The Internet of Things includes a wide variety of "smart" devices, from industrial machines that transmit data about the production process to sensors that track information about the human body. Often, these devices use Internet Protocol (IP), the same protocol that identifies computers over the world wide web and allows them to communicate with one another.

The connection of *physical things* to the Internet makes it possible to access remote sensor data and to control the physical world from a distance. All of these things are creating a "perfect storm" for the IoT. It is estimated that by 2020 there will be over 25 billion devices wirelessly connected to the Internet of Things, including embedded and wearable computing devices. At the same time, IoT imposes fewer data quality and integrity constraints. Although IoT sensors generate data rapidly, they do not entail the same kinds of transactions one finds in traditional enterprise business applications.

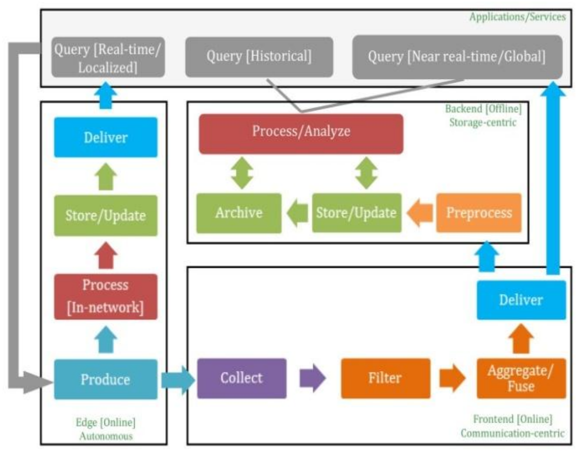

Fig. 1. IoT data lifecycle and data management.

The lifecycle of data within an IoT system—illustrated in figure 1—proceeds from data production to aggregation, transfer, optional filtering and preprocessing, and finally to storage and archiving. Querying and analysis are the end points that initiate (request) and consume data production, but data production can be set to be "pushed" to the IoT consuming services [5]. Production, collection, aggregation, filtering, and some basic querying and preliminary processing functionalities are considered online, communication-intensive operations. Intensive preprocessing, long-term storage and archival and in-depth processing/analysis are considered offline storage-intensive operations.

Storage operations aim at making data available on the long term for constant access/updates, while archival is concerned with read-only data. Since some IoT systems may generate, process, and store data in-network for real-time and localized services, with no need to propagate this data further up to concentration points in the system, "edges" that combine both processing and storage elements may exist as autonomous units in the cycle. In the following paragraphs, each of the elements in the IoT data lifecycle is explained.

Querying: Data-intensive systems rely on querying as the core process to access and retrieve data. In the context of IoT, a query can be issued either to request real-time data to be collected for temporal monitoring purposes or to retrieve a certain view of the data stored within the system. The first case is typical when a (mostly localized) real-time request for data is needed. The second case represents more globalized views of data and in-depth analysis of trends and patterns.

Production: Data production involves sensing and transfer of data by the "Things" within the IoT framework and reporting this data to interested parties periodically (as in a subscribe/notify model), pushing it up the network to aggregation points and subsequently to database servers, or sending it as a response triggered by queries that request the data from sensors and smart objects. Data is usually time-stamped and possibly geo-stamped, and can be in the form of simple key-value pairs, or it may contain rich audio/image/video content, with varying degrees of complexity in-between.

Collection: The sensors and smart objects within the IoT may store the data for a certain time interval or report it to governing components. Data may be collected at concentration points or gateways within the network where it is further filtered and processed, and possibly fused into compact forms for efficient transmission. Wireless communication technologies such as Zigbee, Wi-Fi and cellular are used by objects to send data to collection points.

Aggregation/Fusion: Transmitting all the raw data out of the network in real-time is often prohibitively expensive given the increasing data streaming rates and the limited bandwidth. Aggregation and fusion techniques deploy summarization and merging operations in real-time to compress the volume of data to be stored and transmitted [1].

Delivery: As data is filtered, aggregated, and possibly processed either at the concentration points or at the autonomous virtual units within the IoT, the results of these processes may need to be sent further up the system, either as final responses, or for storage and in-depth analysis. Wired or wireless broadband communications may be used there to transfer data to permanent data stores.

Preprocessing: IoT data will come from different sources with varying formats and structures. Data may need to be preprocessed to handle missing data, remove redundancies and integrate data from different sources into a unified schema before being committed to storage. This preprocessing is a known procedure in data mining called data cleaning. Schema integration does not imply brute-force fitting of all the data into a fixed relational (tables) schema, but rather a more abstract definition of a consistent way to access the data without having to customize access for each source's data format(s). Probabilities at different levels in the schema may be added at this phase to IoT data items in order to handle uncertainty that may be present in data or to deal with the lack of trust that may exist in data sources [2].

Storage/Update—Archiving: This phase handles the efficient storage and organization of data as well as the continuous update of data with new information as it becomes available. Archiving refers to the offline long-term storage of data that is not immediately needed for the system's ongoing operations. The core of centralized storage is the deployment of storage structures that adapt to the various data types and the frequency of data capture. Relational database management systems are a popular choice that involves the organization of data into a table schema with predefined interrelationships and metadata for efficient retrieval at later stages [3]. NoSQL key-value stores are gaining popularity as storage technologies for their support of big data storage with no reliance on relational schema or strong consistency requirements typical of relational database systems [4]. Storage can also be decentralized for autonomous IoT systems, where data is kept at the objects that generate it and is not sent up the system. However, due to the limited capabilities of such objects, storage capacity remains limited in comparison to the centralized storage model.

Processing/Analysis: This phase involves the ongoing retrieval and analysis operations performed and stored and archived data in order to gain insights into historical data and predict future trends, or to detect abnormalities in the data that may trigger further investigation or action. Task-specific preprocessing may be needed to filter and clean data before meaningful operations take place. When an IoT subsystem is autonomous and does not require permanent storage of its data, but rather keeps the processing and storage in the network, then in-network processing may be performed in response to real-time or localized queries.

Looking back at figure 1, the flow of data may take one of three paths: a path for autonomous systems within the IoT that proceeds from query to production to in-network processing and then delivery, a path that starts from production and proceeds to collection and filtering/aggregation/fusion and ends with data delivery to initiating (possibly global or near real-time) queries, and finally a path that extends the production to aggregation further and includes preprocessing, permanent data storage and archival, and in-depth processing and analysis. In the next section, the need for data management solutions that surpass the current capabilities of traditional data management is highlighted in light of the previously outlined life cycle.

II. Requirements of IoT

The requirements of IoT fall into three general categories are, and virtually all applications will require that at least two are satisfied by your database platformsimultaneously:

1. Continuous machine-scale ingestion, indexing, and storage. A *modest* data source may generate millions of complex records per second on a continuous basis. You will need to parse formats like GeoJSON (surprisingly common) at this data rate. The velocity implies a volume that is too large to fit in memory but it is simple to store data on 10 GbE networks at wire speed using commodity disks.
2. Operational ("real-time") queries and analytics. Extracting value from IoT data is all about minimizing the latency from data ingestion to online queries and actionable analytics. For many applications, the value of the data is highly perishable, with an exponential decay on timeframes measured in seconds. IoT queries and analytics are rarely summarizations. Stream processing rarely works, you need to support ad hoc queries in something like SQL.
3. IoT data is all about spatiotemporal relationships and join operations. To support the speed and scale of the first two bullets this means you need at least a true time-series database for very simple uses and a true spatial database for the more general case. Spatiotemporal (or just temporal) must be a fundamental organizing principle of the database internals or it will not scale; you cannot modify a text-and-numbers database with extensions for this purpose.

There are, in practice, two types of databases: relational and non-relational [6] (better known as NoSQL). There are pros and cons to each.

Relational

The relational model organizes data into multiple tables and assigns a value to attributes in each row and column, with a unique key for each row. Other tables can use these keys to access the data without reorganizing the table.

- *The pros*: Relational databases are simple, structured and ¬flexible. They're often used when processing speed is not a factor. They use Structured Query Language (SQL), a commonly understood process for manipulating data. Relational databases are often used in industries such as banking and financial services; because the data is not divisible, data integrity is preserved.
- *The cons*: Relational databases can be slow. If there are many tables utilizing relationships, the responsiveness of data queries can be delayed. In addition, relational databases scale up well, but do not scale out well, making storage expensive.

NoSQL

NoSQL was developed in response to the shortcoming of relational databases, and was deigned to be more open source, more ¬flexible, and horizontally scalable. Unlike relational databases, NoSQL databases are not set up to have tables with linked relationships. There are several types of NoSQL databases, each with slightly different attributes defining how the information is stored and displayed to the user.
The pros: NoSQL databases are generally more scalable than relational ones and performance is generally not an issue. They are designed to expand transparently and horizontally using low-cost hardware.
The cons: NoSQL databases generally cannot handle the analytic processing of the data or joins, which are common requirements for IoT applications. They employ low-level query languages, and do not accommodate transactions where data integrity needs to be preserved (such as in the banking example above).

The reality is, the IoT requires characteristics of both relational [6] and NoSQL databases; the fl¬exibility of NoSQL, which allows different types of data to be stored, and the agility to adapt the underlying data models to specific business requirements and applications, and the data integrity aspects of the relational approach.

A database for IoT applications must be scalable. Ideally, IoT databases are linearly scalable so adding one more server to a 10 node cluster increases throughput by 10%. IoT databases will usually be distributed unless the application collects only a small amount of data that will not grow substantially. Distributed databases [5] can run on commodity hardware and scale by adding new servers instead of swapping out a server for a larger one. Distributed databases are especially well suited for IaaS clouds since it is relatively easy to add and remove servers from the database cluster as needed. An IoT database should also be fault tolerant and highly available. If a node in the database cluster is down, it should still be able to accept read and write requests. Distributed databases make copies, or replicas, of data and write them to multiple servers. If one of the servers storing a particular data set fails, then one of the other servers storing a replica of the data set can respond to the read query. Write requests can be handled in a couple of ways. If the server that would normally accept a write request is down, another node in the server can accept the write request and forward it to the target server when it is back online.

III. Databases in IoT

A. An approach to ensuring high availability with regards to writes is to use a distributed messaging system such as Apache Kafka or Amazon Kinesis, which is based on Apache Kafka. These systems can accept writes at high volumes and store them persistently in a publish-and-subscribe system. If a server is down or the volume of writes is too high for the distributed database to ingest in real time, data can be stored in the messaging system until the database processes the backlog of data or additional nodes are added to the database cluster.
B. IoT databases should be as flexible as required by the application. NoSQL [7] databases -- especially key-value, document and column family databases -- easily accommodate different data types and structures without the need for predefined, fixed schemas. NoSQL databases are good options when an organization has multiple data types and those data types will likely change over time. In other cases, applications that collect a fixed set of data -- such as data on weather conditions -- may benefit from a relational model. In-memory SQL databases, such as MemSQL, offer this benefit.
C. Managing a database for IoT applications in-house For those organizations choosing to manage their own databases, DataStax Cassandra is a highly scalable distributed database that supports a flexible big table schema and fast writes and scales to large volumes of data. Riak IoT is a distributed, highly scalable key-value data store which integrates with Apache Spark, a big data analytics platform that enables stream analytic processing.
D. Readying the data center for the IoT era OpenTSDB is an open source database capable of running on Hadoop and HBase. The database is made up of command line interfaces and a Time Series Daemon (TSD). TSDs, which are responsible for processing all database requests, run independently of one another. Even though TSDs use HBase to store time-series data, TSD users have little to no contact with HBase itself.

MemSQL [8] is a relational database tuned for real-time data streaming. With MemSQL, streamed data, transactions and historical data can be kept within the same database. The database also has the capacity to work well with geospatial data out of the box, which could be useful for location-based IoT applications. MemSQL supports integration with Hadoop Distributed File System and Apache Spark, as well as other data warehousing solutions.

Acknowledgment

This paper consider the challenges brought by the need to manage vast quantities of data across heterogeneous systems. In particular, it consider the factors to keep in mind when choosing a database for an IoT application.

REFERENCES

[1] C. Luo, F. Wu, J. Sun, and C. W. Chen, "Compressive data gathering for large-scale wireless sensor networks," *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking (MobiCom 2009*, Beijing, China, pp. 145–156, 2009.

[2] L. Chen, M. Tseng, and X. Lian, "Development of foundation models for Internet of Things," Front. Comput. Sci. China, vol. 4, pp. 376–385, 2010.

[3] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed. McGraw-Hill; New York, NY, USA: 2002.

[4] R. Cattell, "Scalable SQL and NoSQL data stores," A*CM SIGMOD Record*, vol. 39, pp. 12–27, 2010.

[5] M. T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, 3rd ed. Springer, New York, NY, USA, 2011.

[6] N. Jatana, S. Puri, M. Ahuja, I. Kathuria, and D. Gosain, "A survey and comparison of relational and non-relational database," *International Journal of Engineering Research & Technology (IJERT)*, vol. I, no. 6, 2012.

[7] C. Nance and T. Losser, "NOSQL VS RDBMS -Why there is room for both," *in Proceedings of the Southern Association for Information Systems Conference*, Savannah, GA,USA, 2013

[8] D. Harris, Ex-Facebookers Launch MemSQL to Make your Database Fly, 2012.