A New Database Architecture to cope with frequency and amount of IoT data

Dr. Christian Roth
roth@transaction.de

Dr. Klaus Elhardt
elhardt@transaction.de

Simon Valentini
valentini@transaction.de

September 30, 2022

Abstract

The Internet of Things (IoT) generates data in a never observed frequency and amount. Although it is desired to access those data from a central server, it is by far not possible to transfer and store that data in such a single database server.

Therefore we describe a completely decentralized hierarchy of databases to cope with the amount and frequency of data as well as to provide a single overall view on the data at the same time.

This decentralized hierarchy has further characteristics that are needed for a scalable, extendible and flexible and secure system design. Last but not least, the privacy of data can be controlled individually.

This architecture is fully integrated into the relational database system Transbase and available for various platforms from IoT edge devices throughout large servers.

1 Introduction

The Internet of Things (IoT) has been the next disruptive technological, social and economic development. IoT means, that most objects in the world will be connected to the Internet and will produce gigantic amounts of data at extremely high transaction rates. To get a better understanding of IoT we sketch three scenarios briefly and estimate their technical challenges. As our model we use a city of 1 million people. From this base it is trivial to scale our numbers to a metropolitan area of 10 million people or to a country of 100 million people and even to a world population of 10 billion (or 10^{10}) people.

1.1 Smart Homes

We start with the scenario of smart homes or smart cities. Our model city has 500'000 homes or apartments, each having typically about 20 objects in the IoT. Such objects

are all electrical appliances, heating, water faucets, open-close positions of doors, temperature gauges per room, etc. For our model city this adds up to 10 million (10^7) objects in the IoT. If every such object produces only 100 measurements per second (m/s), we get a transaction stream of 1 billion transactions per second in our model city.

Let us look at the data volume: Assume that a typical measurement has 100 bytes. A day has about 100'000 or 10^5 seconds, resulting in a daily volume of 10^7 objects x 100 bytes x 100 x 10^5 seconds = 10^{16} bytes/day = 10 Petabytes/day or 10'000 hard disks of 1 Terabyte each, again only for our model city.

1.2 Smart Traffic

Let us assume that 50% of our city population drive 1 hour per day, each car measuring speed, fuel consumption, distances to cars ahead and behind, steering wheel actions, breaking and acceleration, adding up to 100 bytes/s. Spreading the cars on the road evenly over a 10 hour period we have 50'000 cars simultaneously driving, resulting in 50'000 transactions per second. This scenario could be handled with the top technology available today for our model city, but it would not scale up to a country like Germany, leaving a gap of a factor of 100.

1.3 Health and Fitness

Today, a surprising variety of health and fitness data can be measured in a non-invasive way in real time: pulse, temperature, blood pressure, blood oxygen, blood sugar, breathing frequency, sleep, steps, exact position and movements outdoors or indoors in a house or apartment. Assume this amounts to 100 bytes/measurement. Continuous monitoring for our model city would result in 1 million transactions per second and in a daily data volume of 100 bytes x 10^6 people x 10^5 seconds/day = 10^{13} bytes/day or 10 Terabytes/day.

1.4 Today and Near Future

Both the transaction rates and the data volumes in the above scenarios are by far unachievable by today's databases. It is obvious, that the problem cannot be solved by the conventional centralized approach. Therefore we propose an entirely new approach to the problem by introducing hierarchies of databases as described subsequently.

2 A distributed system architecture with Transbase as the key technology

The crucial technical obstacle in the centralized approach is that data and transactions are pushed to a central hub causing tremendous bottlenecks as quantified above.

As alternative we propose a hierarchical, highly distributed architecture, where data and the bulk of their processing remain on edge level, i.e. in or near the devices which produce them. To make such an architecture feasible, two requirements must be met:

1. Edge resources: The edge devices must have the storage and processing power to solve their local problem. The microcomputers of today such as a Raspberry Pi can achieve this easily as they have multi-core ARM processors with Gigabytes of RAM and as much flash memory as needed to store data persistently.
2. Software and programming: There must be a simple programming interface for all platforms from edge device up to large servers. Transbase provides this interface uniformly over all those platforms. SQL along with built-in SQL procedures can be used to select or aggregate queries over the database hierarchy. At edge level there is enough computing and storage power to cope with local insert or delete operations. This results in very high reuse of code, very reliable software and low cost.

We strongly believe that only distributed data storage and distributed computing will be able to solve the volume problems that arise with IoT. We will now discuss our architecture proposal under several technical aspects.

2.1 Resources

In a typical IoT device like a Raspberry Pi there will be sufficient memory and CPU resources even though it costs only a few dollars. A resource-efficient database system like Transbase can be used on the device. Insertion rates can be easily maintained and data aggregation can be performed locally, too. Even deleting or compressing data can be performed in the devices before storage capacity runs low. With a 8 GB flash storage on chip and a data rate of 100 Bytes/sec a history of more than 2 years could be stored (100 Bytes/sec = 10^7 Bytes/day = 400*10^7 Bytes/year = 4GB/year) on edge.

2.2 Hierarchies of databases

According to aggregation needs there will be hierarchies of databases. If we talk about a smart home, then we talk about the electricity meter which is installed in each home or apartment; all data from switches and plugs (called sensors in the following) will flow into that meter; actually we assume that this meter is the lowest-level database where data is persistently stored and call it edge database. We can further assume that there is another "crowd" database per building which collects data from all meters in that building, a crowd database per city that collects all data from all buildings, and a crowd database per country that collects all data from all cities in that country. Finally there may be a crowd database for the world, aggregating all country databases. Such a global database would be fascinating for energy trends, climate research, market analysis, etc.

For a home or a building the electricity meter could be a Raspberry Pi (fixed installation with permanent Internet connection), for health applications it would be the Smartphone of a person, online or offline, but mobile. Transbase runs on both devices with a footprint of significantly less than 10 MB. The remarkable unique feature of this hierarchical approach is that crowd databases on higher levels can be assumed to be virtual databases that do not even necessarily store data persistently. But obviously,

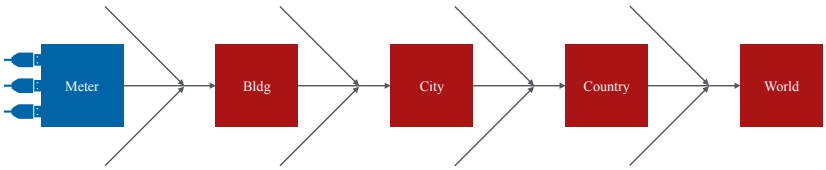

Figure 1: 5 level hierarchy

virtual databases on higher levels can store data persistently, too. In such an environment, the stored data may be seen as cached data which may help to reduce response times, as no query evaluation down the hierarchy is necessary.

In addition, cached data may also be used to solve big data problems with Data Warehouse (DWH), Neural Network (NN) or Artificial Intelligence (AI) techniques. NNs trained on a server could then even be pushed down to lower levels in the hierarchy to recognize critical situations locally, e.g. the pulse disorder ventricular heart fibrillation, in health applications.

For the smart home application the hierarchy is defined as follows:

sensor → meter → building → city → country → world.

Each element in the hierarchy has to know only its parent, a building knows to which city it belongs, the city to which country, and so on. It is not necessary that the parent knows all of its children. However, it is assumed that all databases use the same database schema, i.e. the same set of tables with the same columns. The database schema actually is defined on the lowest-level database and propagated up to the root. Children must register with their parent so that the parent database can push queries down and pull data up.

The following picture shows the hierarchy of databases in such a smart home application: Data is physically stored in the meters, all upper databases are assumed to be "crowd databases".

2.3 Hierachical Query Processing with Crowd SQL

The following schema illustrates how an SQL query can be formulated on the top hierarchy level and is automatically distributed on all lower levels in a special way. For simplicity, we restrict the example to three levels:

```
SELECT <aggregations>                                    -- World (level 0)
FROM CROWD <crowd specifications>
    (SELECT <aggregations>                               -- Country (level 1)
    FROM CROWD <crowd specifications>
        (SELECT <fields>                                 -- City (level 2)
        FROM <base tables>
        WHERE <level 2 restrictions>)
    ) WHERE <level 1 restrictions>
WHERE <level 0 restrictions>
```

Figure 2: Crowd SQL

Query processing starts at the top level and works as follows: each level automatically distributes the query part beginning with the first "CROWD" (indented in the example above) to the next lower level participants, receives their results and processes the results (as if they were a local table), finally sends the results to the next upper level. Beside this simple and schematic example, all SQL constructs (GROUP BY, HAVING, WINDOW functions etc.) are supported.

It is easy to see that this concept is extremely powerful and flexible. The query parts can be individually tailored to the single levels, but are integrated into a single construction. In particular, the queries for the next lower level could be formulated and stored as views.

If we assume a generous response time of 1 sec for one aggregation layer, we have a final result over all meters in the world after only 5 seconds. Such short response times could be used to recognize fluctuations in energy consumption (also blackouts) quickly and to use them on the market, e.g. on short term spot markets.

2.4 Database volume and scalability

If we assume 10^2 countries, 10^4 cities per country, 10^5 buildings per city and 10^3 meters per building as an average, then we have a (virtual) world database distributed into 10^{14} physical meter databases. If a meter database takes 1 megabyte of data, the virtual world database takes 10^{14} megabyte or 10^2 exabytes of data, a volume that currently could be neither transferred to nor stored in a single place.

Databases grow physically by adding meters (and their associated database) which is an isolated local operation. Only if too many meters are linked to the same building or too many buildings are linked to the same city there might be a need to split a node in the hierarchy and to restructure this intermediate level much like B-tree splits.

Also note that queries referring to a given city or building have to be propagated only to nodes below this city or building database and do not influence queries on other city or building databases.

2.5 Band-width

As data is stored at the edge level only, there is no need to transfer all measured data (although it would be possible). Band-width is needed only for transferring queries and results. Results in most cases are aggregations and as such are very small.

2.6 CPU power and storage capacity

As mentioned before, device computers provide enough resources for concurrently storing incoming data, evaluating data analysis queries (coming from outside or from within the device) and also for data maintenance.

Under the above assumptions device data can be kept for a window of one year. Data outside that window is deleted or kept in compressed form. The device database can be expected to provide concurrent rates of 20'000 inserts/sec, 5'000 deletes/sec and 10 evaluation queries/sec which seems more than sufficient.

2.7 Data privacy

Data kept in devices sometimes has to be considered private or partially private, e.g. health data. In those cases, each edge database can define views or privileges to restrict remote access from crowd databases. E.g. queries that retrieve some averages might be allowed, while queries that retrieve personalized or not anonymized data records might be refused.

Thus, data privacy can be controlled and enforced locally and individually for each edge device which is very much in contrast to current implementations where data is transferred to Google, Apple or other tech giants where they can be exploited without permission or hacked.

2.8 Fault tolerance

For highly distributed aggregation queries there must be a sort of fault tolerance. Seen from a crowd server who performs such a distributed aggregation, the following situations with satellite databases must be considered:

- An edge database might be "down" or offline.
- An edge database might be not reacting on aggregation queries (controlled by a query timeout).
- An edge database might have a database schema that does not provide the data requested (query syntax error).
- An edge database might not have privileges to perform the requested query.

In all of these situations the aggregator should simply ignore the corresponding edge databases and perform the aggregation over the remaining databases. The crowd server easily determines the percentage of ignored databases. Based on this, it can decide whether to deliver an aggregation result itself.

Our SQL example above could e.g. specify in the crowd specifications:

```
RETURN AFTER 2 SECONDS OR 1000 MEMBERS
```

The first result record will be delivered after the time window has expired or, if specified, after the desired number of crowd members have returned a complete result. If the time window expires without a member delivering a complete result, an empty result set is delivered as the overall result.

2.9 Reverse connect

Classical distributed databases must know its child databases and connect to them when needed. For the architecture proposed here, we propose to revert the connection initiation: a child database connects to its parent and registers there, and the parent can distribute queries from then on to all registered child and grand child databases.

3 Transbase as IoT database

The following important properties of Transbase make it a powerful platform to analyse data over the Internet of Things and draw conclusions.

3.1 Platforms

Due to its low resource consumption, Transbase is available on all relevant platforms for devices, e.g. Raspberry Pi based systems. It is important to note that the programming interfaces and the functionality of Transbase are identical on all platforms, even on large servers. Access from outside crowd databases is handled through TCP/IP connections that are initiated from the edge device. So no incoming TCP/IP connections must be permitted on the device.

3.2 Performance

Transbase uses clustering B+-trees as its primary data structure on disk (and in memory), thereby enabling set-oriented queries defined by primary key intervals to be processed very quickly. Inserts have to preserve the clustering and therefore are slightly, but not significantly slower as compared to many other database systems. In addition B-trees provide excellent scalability for growing databases. Thanks to its efficient storage architecture, Transbase often performs better not only on evaluation queries, but also on pure insert profiles. Delete operations also make use of clustering and thus are performed efficiently. If a database fits completely into main memory, Transbase performs minimal IO as only log records are written persistently to disk for update transactions.

3.3 Security

Data stored in Transbase can be blockwise encrypted so that it cannot be hacked by accessing the database files. Data transferred between a database and a client is also encrypted by TLS/SSL. In addition, privileges of SQL and user or application logins guarantee allow further protection of information.

3.4 Multidimensional features

An extension for B+-trees for multidimensional data makes Transbase an ideal database system for multidimensional problems, e.g. described by geographic coordinates combined with a time dimension. In this case, clustering storage means that records belonging to a geographic region (a two- or threedimensional interval) and a time interval are stored on few adjacent disk pages and therefore can be retrieved with minimal IO.

3.5 Ledger functionality

Transbase supports so-called ledger tables which can be used to verify for each record its tamper-freeness. A unique Merkle tree is used to compute a single database hash value that reflects the update status of the database. For financial, logging, supply chain and many other applications this extension can be utilised easily with almost unchanged performance. The user simply has to define a table as "ledger table" and delegates all hashing and chaining to the database system.

Depending on the application it may be desired or even mandatory to guarantee tamper-freeness of data. Think on autonomous driving, event logging, supply chain management, accounting systems or others where such verifiable data integrity is required.

Transbase Ledger is capable to link all records in ledger tables into a single hash value from which data integrity of a single record or of the whole database can be proofed later.

4 Summary

The database architecture presented here uses local resources on low-end, inexpensive edge devices to store data and analyze it using highly distributed queries received from outside. Therefore, the big data problem which arises when data is concentrated on servers is solved by simply using the local resources in edge devices and distributing queries and aggregating their results in a hierarchical structure.

This architecture scales perfectly with the number of devices. Performance is predictable as each device has to handle its own data. The very flat hierarchy of servers (only 5 levels in our sample case of smart homes cover the hole world) guarantees an excellent performance with response times of < 2 sec per city and < 5 sec for the entire planet.

In addition, data privacy can be guaranteed by local restrictions that can model different levels of privacy, from no privacy to strict privacy including end-to-end encryption if needed.