Database-as-a-Service for IoT

Anand Gogawale
V.C.E.T.
Mumbai, India
Email ID: anandgogawale2395@gmail.com

Fasih Khatib
V.C.E.T.
Mumbai, India
Email ID: fasihxkhatib@gmail.com

Pratik Sontakke
V.C.E.T.
Mumbai, India
Email ID: pratiksontakke22@gmail.com

Swati Saigaonkar
Asst. Prof., Dept. Of IT, V.C.E.T.
Mumbai, India
Email ID: swatiavarma@gmail.com

Abstract – The Internet of Things, often abbreviated as IoT, is a network of "things" like software, sensors, electronics, etc. that have the ability to exchange data with their operator, manufacturer, and/or other devices using an existing communication infrastructure. IoT is still in early stages. However, it is developing at a rapid rate. As more "things" start communicating, more data will be generated. This paper outlines the requirements of a general-purpose, open source platform that would enable these "things" to store and retrieve their data reliably over the network.

Keywords – Database-as-a-Service; Internet of Things; Web Service

NOMENCLATURE

IoT – network of physical objects that are able to communicate using existing communication infrastructure.

HTTP – Hypertext Transfer Protocol is an application layer protocol used for transmitting hypermedia information over the web.

CoAP – Constrained Application Protocol is an application layer protocol designed for use in resource-constrained devices.

UDP – User Datagram Protocol is a connection-less transport layer protocol designed to be used with network later protocols.

TCP – Transmission Control Protocol is a connection-oriented transport layer protocol designed to be used with network later protocols..

I. INTRODUCTION

We are living in a new era of connectedness. Our world is more connected than before. Humans are connected in unprecedented ways. In the last few years, these connections have extended beyond human realm. A lot of "things" from our physical world are now able to communicate with each other or with us through sensors, actuators, etc. without any human involvement. These "things" could be cars, factory equipment, personal electronics, etc. The intelligence embedded in these things generates vast amounts of data that can be collected, networked, and analyzed for a variety of purposes.

Humans are quickly being outnumbered by these internet-connected devices that constantly transmit data. It is estimated that by 2020 there will be atleast 50 billion devices connected to the internet [1].

II. IMPACT ON STORAGE

The impact on storage is fairly obvious - there is more data to store. However, the less obvious impact is that the data is highly unstructured. As time passes, IoT will find a more prominent role in telemetry. Telemetry involves wireless transmission of small amounts of data such as environment conditions. Telemetry data is often small and unstructured. The same is true not just about telemetry, but about other forms of data as well. As it is shown in Figure 1, most of the data that is transmitted over the internet is now unstructured. [2]

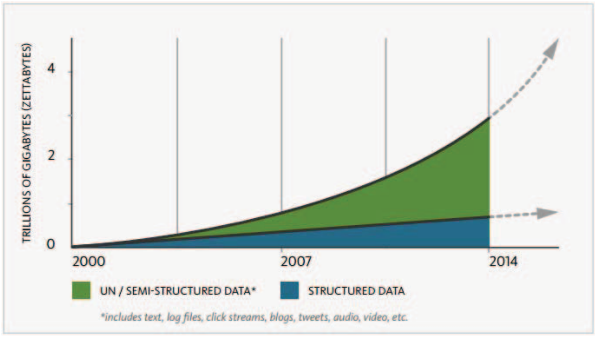

Fig 1. Amount of structured vs unstructured data

Another characteristic of the data transmitted by these devices is that it is small. In contrast with big data, which contains petabytes of information that can be mined for patterns, small data contains very specific attributes like temperature, humidity, wetness, pressure, etc.

As an example, consider a wind turbine with a variety of sensors on it which report the velocity of wind, temperature, vibration, and other such information. We can then remotely change the direction of the blades based on the transmitted small data.

Currently, the data storage solutions for such scenarios is provided by SAP and AT&T. Although reliable, the closed-source nature of these solutions hinders innovation that would be needed as IoT progresses rapidly. The following sections outline how an open source platform can be created to store this data.

III. REQUIREMENTS OF THE SYSTEM

While creating a system to support IoT data storage, it is necessary that it meets a few requirements.

First, it should be able to uniquely identify each device uniquely. This name maybe a GUID or a user-defined name.

Second, it should allow the data to be transferred using a variety of protocols. IoT devices can communicate using a number of protocols like Constrainted Application Protocol (CoAP) or Hypertext Transfer Protocol (HTTP). The choice of protocol depends upon the amount of memory that the device has.

Third, it should be a REST-ful web service. The REST architecture allows for a uniform way to access and modify a resource.

Fourth, it should allow for realtime triggers. As IoT is about realtime data, there should be a way to trigger actions when certain conditions are met. For example, send an email when the temperature drops to freezing.

Fifth, it should provide a security mechanism like password to protect the data.

Finally, it should be able to handle a large number of concurrent, I/O-bound requests.

IV. STORING THE DATA

IoT will produce a lot of unstructured data. This poses a question of how the data should be stored. Traditional relational databases cannot handle unstructured data as they require a strict schema. Also, any database that holds the data incoming from a variety of devices should be able to scale horizontally when the load increases. Again, traditional relational databases fail to scale well horizontally.

Relational databases store the data in the form of tables which need to be created prior to inserting any data into the database. While creating the tables, information about the number of columns and their datatypes need to be specified. Althought this helps by providing a sort of consistency check, it is not flexible enough to handle the myriad forms of data sent by the IoT devices.

For example, consider two temperature sensors from two different manufactures that send their data periodically. One sensor might send simply one temperature field while other may send the temperature and the time at which the temperature was recorded. It is not possible beforehand to know how many fields will be sent. So the tradiotional relational databases cannot be used.

Also, traditional relational databases are highly normalized. Normalization increases the performance of the database by avoiding insert and update anomalies. However, normalization prevents the databases from scaling horizontally; something that is a must-have for a system handling heavy loads. This is because it is not possible to split the table along rows and store different parts on different instances of the system after it has been scaled horizontally. Finding which row is located where based on its primary key becomes difficult.

In contrast, NoSQL databases are designed to handle large amounts of unstructured data with efficiency and speed. Document-oriented NoSQL databases, in particular, are an apt choice for storage of this data. These document databases use data structures that allow unstructured data to be stored efficiently. Also, these databases scale well horizontally.

Document databases allow every "row" to have any number of fields. Also, they make no assumptions about the datatypes of these fields. This suits the IoT scenario.

Also, NoSQL databases are highly denormalized. Although this comes at a cost of introducing slight inconsistency in the system, it allows the system to scale well horizontally.

V. UNIQUELY IDENTIFYING EACH DEVICE

When storing data, it is necessary to identify which device has sent this data. Devices can be identified in two ways.

First, they can be identified by using a GUID. Since GUIDs are used to uniquely identify computer hardware, this can help in avoding name collisions.

Second, users of the system can be allowed to choose a unique name that identifies their device. For example, "temp-sensor-03" may be used to identify a temperature sensor. However, this requires ensuring that no two devices have the same name. These names mean nothing to the system beyond unique identifiers. Whenever a device sends its data, it will identify itself using this name. The system will then store the data with the identity of the device.

VI. SUPPORTING VARIOUS PROTOCOLS

It is imperative that a system catering various types of devices support various protocols as well. Limiting to just one protocol would limit the number of devices that can be served.

Two of the most common protocols are HTTP and CoAP.

HTTP allows for existing servers and technlogies to be able to serve these devices. The HTTP verbs like GET, PUT, POST, DELETE provide a clear vocabulary for the actions to be performed by the IoT device. However, HTTP comsumes a lot

of bandwidth and memory. HTTP headers are a good example. As plain strings with no compression of any sort, they bloat the network protocol. This makes it unsuitable for devices with low power and intermittent connectivity.
CoAP overcomes the limitations of HTTP. It is designed for devices that run in resource-constrained environments. Its packets are much smaller than HTTP and it runs over UDP instead of TCP making it much more bandwidth efficient. Adding CoAP support is easy as it can interoperate with HTTP and the REST-ful web at large via simple proxies.

VII. CREATING A REST WEB SERVICE

Creating a REST service allows for numerous benefits. First, it allows us to expose a set of well-defined endpoints. Each of these endpoints would correspond to the functions provided by the system.
Second, it allows the system to be interoperable with various devices of different make and design. Any device with network connectivity would then be able to use the services offered by the system.
Third, using REST architectural style allows the communication to be stateless. This means less bandwidth and memory consumption.
Finally, Since Web services are based on open standards their cost is low and the associated learning curve is smaller than that of many proprietary solutions. A smaller learning curve means that the developers who write code for IoT devices can develop, test, and ship faster.

VIII. CREATING TRIGGERS

IoT is all about realtime data and the ability to react to realtime data. So, it is necessary that a system that stores this data have some form of mechanism that allows the owner of the device(s) to be notified.
The system should let the owner of the device be able to set the conditions that would generate a trigger. For example, “temp-sensor-03” may have a trigger to notify the owner when the temperature rises or falls below a certain threshold.

IX. PROVIDING SECURITY

Since these devices send data and the data may generate triggers, it is necessary to provide security for the data. Security should be provided in the form of encrypted connection which will protect the data in transit. Also, a password should be associated with each device. This will prevent malicious data from being stored in the database for a particular device.
For example, consider that a malicious third party has come to know the existence of “temp-sensor-03”. If there is no password mechanism, the system would simply store the data. Even worse, the malicious data can possibly generate a trigger. With password in place, every time the data is sent to the system, it would check if the data contains the right password and only then would the data be stored in the database.

X. HANDLING MULTIPLE CONNECTIONS

For any service that caters to clients, it is necessary to handle multiple, concurrent connections. In case of IoT, there will be a number of small, I/O-bound requests. There are two possible ways in which a request can be handled.
First, for every request the system may spawn a thread with some fixed amount of memory and this thread would cater to the request.
Second, the the system may be single threaded that adds request to a first-in-first-out queue. It then processes each request one-by-one, asycnhronously.
The first approach is good if the requests were CPU-bound instead of I/O bound. This is because I/O bound requests would put the thread in a blocked state, waiting for the I/O to finish. The memory that has been allocated to the thread is wasted as it is in a blocked state. This would reduce the number of concurrent connections.
In the second approach, the system would receive a request from the device and then schedule it for I/O. The request would be removed from the queue and added to the tail of the queue. The system would then proceed onto the next request, and so on. Here, the system can use the memory to its full capacity as the request does not waste memory while waiting for I/O to finish. This would allow for more concurrent connections.

XI. CONCLUSION AND FUTURE SCOPE

In the coming years, IoT will find a more prominent role in our lives. With newer devices there will be newer forms of data and the need to service requests differently.

The system which stores the data can also be expended to provide machine-to-machine communication. The triggers could be modified to communicate with other devices instead of just notifying the owner of the device.

REFERENCES

[1] Dave Evans, “*The Internet of Things: How the Next Evolution of the Internet Is Changing Everything”*, April 2011

[2] Couchbase Inc., “*Why NoSQL?”*, pp. 2-5