# Study on Distributed NoSQL Database on Embedded System for Low-power IoT

PAETHONG PORNPAT[†1] MITARO NAMIKI[†1]

***Abstract***: The world is moving toward the era of Internet of Things (IoT). Internet of things provided an internet-based platform for devices to communicate, exchange information, make decision, invoke action in order to provide services. The advancement of IoT will bring upon a new era of computing and communication to improve people's life. The most essential part of IoT is the database which collect and store information from different sensors on the device. In general cloud computing is uses to process these information, however, using cloud computing means every IoT devices need to have internet connection. The drawbacks of this method included high cost and high consumption of energy. To solve this problem a method of using credit-card sized computer is purposed. These devices are cheap and energy efficient, but still provide enough computing power. The purpose of this paper is to demonstrate and explain how to construct database server for IoT middleware using low energy computing devices such as Raspberry Pi as an alternative to the traditional method of cloud computing.

***Keywords***: Credit-card Sized Computer, Distributed Database, Embedded System, Internet of Things, MongoDB, NoSQL Database, Low-power, Raspberry Pi

## 1. Introduction

The most essential part of IoT is the database [1] which collect and store information from different sensors on the device. In general cloud computing is uses to process these information, however, using cloud computing means every IoT devices need to have internet connection. The drawbacks of this method included high cost and high consumption of energy. To solve this problem a method of using credit-card sized computer is purposed. These devices are cheap and energy efficient [3-5], but still provide enough computing power. The purpose of this paper is to demonstrate and explain how to construct NoSQL database server [6] for IoT middleware using low energy computing devices such as Raspberry Pi as an alternative to the traditional method of cloud computing.

## 2. Issues and Goals

Usually, IoT databases, which are based on cloud computing, have high reliability, redundancy, security, and processing power [2]. The drawbacks of the cloud database are the cost, required internet connection, limited bandwidth, and high power consumption. These drawbacks make cloud database not suitable for household, office, or farm. In such case the IoT server do not require a lot of processing power, an alternative database server such as Raspberry Pi can be implement. The benefits of database on Raspberry Pi included, local-base storage, affordability, and energy consumption, while also open new possibility for future IoT second database hardware.

## 3. System Structure and Design

The system contained two main parts: master node and data nodes. The two nodes work together as a single system; the master node is able to connect to other data nodes, while the data node is able to connect to master nodes as well as other nodes in the system as shown in Fig.1.

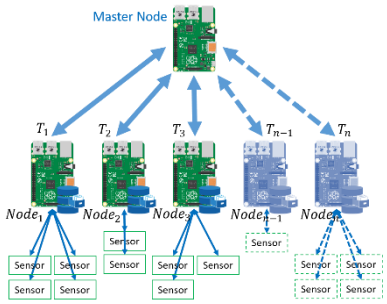


Fig.1: System Structure and Design

Every node, each runs MongoDB (which work as NoSQL database server), work completely independent from each other (individual running) to store and process information from the sensor hardware that it's connected to. For data distribution, each node has a metadata table, which contained information about the node's data such as temperature, pressure, humidity, and etc. The Fig.2 shows how the node distributes these data to other node, as a result creating an information network between itself and other nodes.

Master node, which act as a proxy server, when it receives query command from client, it will find alternative data nodes by looking at metadata table. Then, it forward that query to alternative data nodes. Finally, it will combine all data from data node and respond to client.

Data node, which act as a data storage, it will receive and store a data from connected sensor hardware. Moreover, it will execute a query command from client via master node.

## 4. Distribution Algorithm

### 4.1 MongoDB Sharding

Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meet the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves this problem with "horizontal scaling", by adding more

†1 Tokyo University of Agriculture and Technology

machines to support data growth and the demands of read and write operations.

- In replication, all writes go to master node
- Latency sensitive queries still go to master
- Single replica set has limitation of 12 nodes
- Memory can't be large enough when active dataset is big
- Local Disk is not big enough
- Vertical scaling is too expensive

### 4.2 Basic Distribution Algorithm

Due to that all node work independently, the problem is how to synchronize each node. This problem can be solved by using metadata synchronization (as config server component of normal MongoDB sharding). When each data nodes have metadata table on their own database that contains data key, IP address, last active, etc. to reference to each other data node. It can directly access and get the target data. Mover, MQTT protocol will use for synchronize all metadata on each data node, when some node has inserting, updating event.

**4.2.1** Compression with MongoDB Sharding

First, the client must connect to a router (mongos), but in this case the client can connect directly to data nodes (mongod) (In fact, Rasperry Pi have to connected with sensor hardware, via GPIO, to get sensor data and store it into database). Second, MongoDB sharding will use metadata on config server for accessing to shard node, but in this case I use metadata table on its database. Last, Traditional MongoDB sharding have to always synchronize and balance the data on each shard node, which make it impossible to reduce energy consumption. My proposed solution of using database event and MQTT will make this possible.

**4.2.2** Consistency Model

Firstly, the data node will self-manage its data. Second, each data nodes will have metadata table on their own database. It is used for mapping and accessing data from other nodes. Last, when some data nodes fire event of inserting or updating a new type of sensor (new key), they will broadcast a message to each node to updates its metadata table and checksum value.

**4.2.3** Methods

Frist, Handling data changed events by modifying MongoDB core (mongod) and/or MongoDB driver libraries. Second, MQTT with QoS 2 is used for message exchanging in bi-directional communication as shown in Fig.3.

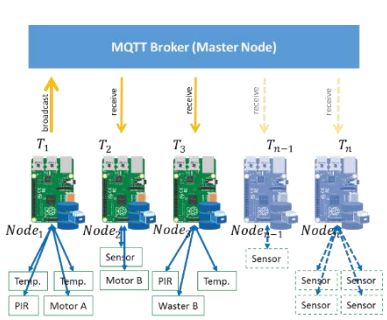


Fig.3: Metadata with MQTT Message Exchanging

## 5. Current progress and Conclusion

We compared x86 machine and Raspberry Pi for the operation time and power which showed that our solution looks promising. With evaluation environment, MongoDB version 2.4.10 for Raspberry Pi 2 Model B of 900MHz quad-core ARM Cortex-A7 CPU, 1GB of memory, 16GB of SD card as local-base storage and Raspbian of operating system.

We evaluated the MongoDB MongoDB (Individual Running) and MongoDB Sharding on 4 x Raspberry Pi for database performance and energy usage.

Table 5.1: Database Performance Comparison

| **Database Performance** | **MongoDB (Individual Running)** | **MongoDB Sharding on 4 x Raspberry Pi** |
|---|---|---|
| Inserting Time | **475** seconds | 4,875 seconds |
| Insert per Second (IPS) | **3,363.17** | 328.14 |
| Online Transaction Time | 580 seconds | 600 seconds |
| Transactions per Second (TPS) | 12.83 | 11.74 |

Table 5.2: Energy Usage Comparison

| **Energy** | **MongoDB (Individual Running)** | **MongoDB Sharding on 4 x Raspberry Pi** |
|---|---|---|
| Idle mA | 250 mA (1,000 mA) | 1,000 mA |
| Avg. Execution mA | 300 mA (1,200 mA) | 1,250 mA |
| Inserting Energy (J) | 142.5 kJ (**570** kJ) | 6,093.75 kJ |
| Transaction Energy (J) | 174 kJ (700 kJ) | 750 kJ |

Table 5.1 shown that inserting performance of MongoDB (individual running) was faster than MongoDB sharding, and insert per second (IPS) performance was higher than MongoDB sharding. For online transaction and transactions per second (TPS) performances, they are almost the same value. Moreover, table 5.2 shown about MongoDB (individual running) was used energy less than MongoDB sharding.

In conclusion, MongoDB as individual running on Raspberry Pi is faster and low-power than traditional sharding. The credit-card sized computer hardware and MongoDB database engine can be implemented as a simple, flexible and affordable database server for IoT middleware in the future.

## References

[1] Gubbi, Jayavardhana, et al. “Internet of Things (IoT): A vision, architectural elements, and future directions.” Future Generation Computer Systems 29.7, pp. 1645-1660, 2014.

[2] Zarghami, Shirin. “Middleware for Internet of things.”, 2013.

[3] Brock, J. Dean, Rebecca F. Bruce, and Marietta E. Cameron. “Changing the world with a Raspberry Pi.” Journal of Computing Sciences in Colleges 29.2, pp. 151-153, 2013.

[4] Anwaar, Waqas, and Munam Ali Shah. “Energy Efficient Computing: A Comparison of Raspberry PI with Modern Devices.” Energy 4.02, 2015.

[5] Maksimović, Mirjana, et al. "Raspberry Pi as Internet of things hardware: performnces and constraints." Design Issues 3, pp. 8, 2014.

[6] Parker, Zachary, Scott Poe, and Susan V. Vrbsky. “Comparing NoSQL MongoDB to an SQL DB.” Proceedings of the 51st ACM Southeast Conference. ACM, 2013.