International Journal of Web Applications

DLINE JOURNALS

Print ISSN: 0974-7710
Online ISSN: 0974-7729

IJWA 2025: 17 (4)
https://doi.org/10.6025/ijwa/2025/17/4/145-152

A Cloud- based Data Processing Platform for the Internet of Th ings: Architecture, Implementation, and Performance Evaluation

Xia Peng
Normal College, Xinxiang Vocational and Technical College
453006, Xinxiang, Henan. China
funing20031317@163.com

ABSTRACT

The paper presents a cloud-based data processing platform for the Internet of Things (IoT), designed to handle the collection, storage, processing, and visualization of sensor data. It leverages cloud computing to address the challenges posed by the massive volumes of data generated by IoT devices, utilizing scalable technologies like Hadoop and Hive for distributed storage and Map Reduce based analytics. The system architecture includes a TCP server module built with PHP and the Swoole extension to manage real time sensor data ingestion, and a middleware layer based on Redis to decouple components and enable efficient inter module communication. The platform offers two primary external interfaces: one for task execution (via Hive) and another for data uploading to HDFS. Performance tests on the TCP server show high data processing efficiency at low concurrency, with performance degrading beyond 500 concurrent connections still sufficient for most real world IoT scenarios where data acquisition rates are lower. The Hadoop cluster demonstrates robust functionality, including dynamic node addition/removal without service interruption. While domestic IoT analytics platforms in China lag behind global counterparts, this work contributes a customizable, open framework that supports secondary development and real time monitoring. The authors conclude that the platform meets typical production needs and suggest future improvements in API design and developer usability to lower the barrier for customization and deployment.

Keywords: Internet of Things (IoT), Cloud Computing, Big Data Processing, Hadoop, Hive, TCP Server, Redis Middleware, Data Visualization

Received: 13 July 2025, Revised 2 October 2025, Accepted 11 October 2025

1. Introduction

The concept of the Internet of Things was initially introduced by professors from American universities during their research on RFID in 1999. Building on the idea of the Internet, the Internet of Things extends its client

interface to encompass any item or object through technologies such as radio frequency identification (RFID), infrared sensors, global positioning systems, laser scanners, and other information gathering devices. This framework connects anything to the Internet through mutual agreement, enabling information exchange and communication, and thereby enabling intelligent identification, location tracking, monitoring, and management. Cloud computing has consistently played a crucial role in the evolution of Internet of Things technologies. As a foundational and vital approach to processing big data, cloud computing provides reliable management and analytical support for the vast amounts of data generated by the Internet of Things. It is anticipated that the global number of sensor nodes will reach 20 billion by 2020, creating unprecedented data volumes. In contrast to traditional data handling, the high lateral scalability of cloud computing enables flexible expansion of system computing power to meet the processing demands of big data from the Internet of Things, making it well suited to these demands. Therefore, leveraging current cloud computing technologies to establish a data processing platform for the Internet of Things is of significant practical importance and essential for advancing the field.

2. Background

Par Stream, a German company, provides online analytical services for big data in the Internet of Things, being the first industry class platform designed to manage massive, high speed data in this domain. This platform enables companies to obtain analytical results from big data more quickly and comprehensively, thereby aiding in decision making and system analysis. Due to its strong performance and positive user experience, the company's products have gained traction across Europe, the United States, and various other regions. Conversely, the domestic landscape remains relatively underdeveloped, lacking national level projects in this area. There are online data analysis platforms such as Baidu, Music Federation, and U cloud that provide standard functions, including sensor data reception, online analysis, and result display via HTTP. However, the performance and user experience in our country still lag behind, with no relevant open source technology frameworks available. In December 2013, Alibaba, the largest cloud computing enterprise in the country, introduced the "flying platform." This platform represents innovative exploration across product development, pricing, services, and third party collaboration. Significant advancements in related technologies have been achieved, providing Chinese companies with a universal computing platform that rivals the capabilities of a single cluster of 5,000 servers a feat typically achieved only by leading companies like Google and Facebook. [4]. In addition, because of its technical accumulation in search, Baidu Inc. also has strong strength in the cloud computing infrastructure and massive data processing and has gradually opened the IssS, PaaS SaaS and other multi level platform services. Baidu Cloud OS, with individuals at the centre, organises data and applications, supports Web apps, and provides App pages [5].

3. Methodology

The platform is a data processing framework for the Internet of Things that utilizes cloud computing. It carries out the procedures of data collection, storage, processing, and visualization, along with two development interfaces. Users can monitor the status of sensor nodes and perform basic configuration through a web-based community, leveraging the robust storage and computational capabilities of cloud computing [6]. Simultaneously, after thoroughly assessing users' specific requirements, the system integrates various core libraries used in the platform's development. Further more, leveraging the PHP programming language, which is user friendly, individuals can swiftly incorporate customized functional modules into the system based on their distinct needs. The system primarily delivers services through the Web server and the TCP server; among these, the Web server facilitates

cross-platform application configuration and data visualization for users [7]. Conversely, the TCP server manages the reception of sensor data. Cloud computing clusters handle data processing within the system. The system's "middle layer" oversees the communication between the var- ious modules, acting as middleware for each component and implementing a Redis based message queue. This message queue enables data sharing and service invocation among the modules while simultaneously reducing coupling. It is sufficient to define the interaction protocol and data interface without needing to understand the underlying implementation specifics, thereby improving the system's maintainability. The architecture for data storage and processing resides at the core of the system, tasked with storing and computing essential data [8]. The intermediate layer interacts directly with this core layer. It delegates computing tasks to the Hive module within the cluster module; once the Hive module receives the task details, it compiles and parses them, transforming them into Map Reduce computations executed in Hadoop, resulting in execution results. Another crucial function of this layer is data uploading [9]. The intermediate layer appropriately issues the data upload signal to the cluster layer, with the data upload module responsible for managing this message. Upon receiving the message, the module retrieves data from the intermediate layer's cache and uploads it to the cluster's HDFS file system, thereby achieving data persistence and facilitating calculations for the Hive module. The specific architecture diagram is depicted in Figure 1.

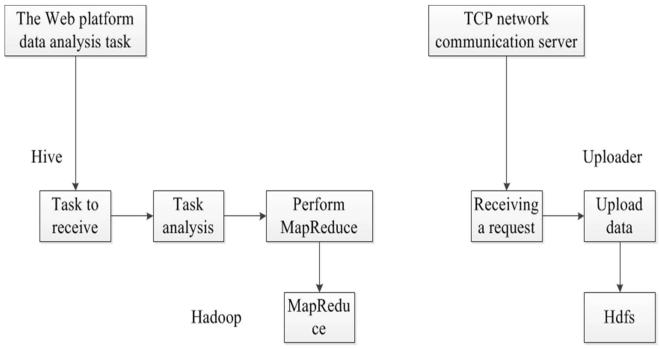

Figure 1. System architecture design

The layer reveals just two external interfaces to the upper level: the task delivery interface and the data upload interface, corresponding to two modules, specifically the Hive module and the uploader module. Within the Hive module, there are three sub modules: Task acceptance. This interface call is available externally to retrieve comprehensive details about the task. Task analysis: the primary function of this task is to transform SQL queries into Map Reduce programs [10]. Task execution: the Map Reduce program is dispatched to Hadoop for processing, and the task results are collected to update the intermediate layer with the task status. In the Uploader module, there are two sub modules: Request reception, which provides interfaces for capturing data upload events. Data upload: once triggered, the data is automatically retrieved from the middle layer cache and transferred to the specified HDFS location for long term storage [11]. The TCP server module was implemented in PHP. Although PHP is not ideally suited for developing high performance network communications servers, the recent extension library Swoole allows PHP to undertake more complex large scale network application development. The primary role of this module is to manage terminal node requests, establish TCP connections, and handle terminal data uploads, serving as a crucial functional module of the platform [12]. This module is segmented into connection maintenance, engineering management, data frame management, and other sections. Effective encapsulation is applied across sub modules, enabling users to access the system's fundamental functions and create more advanced functional modules tailored to their specific needs. The module is primarily divided into three layers: the outermost layer is the Swoole TCP connection pool, the middle layer is the

module's application layer, and the innermost layer is the core layer. The primary responsibilities of each layer are as follows: Swoole maintains the Swoole TCP connection pool. When Swoole receives a connection request, it initiates a new connection in the connection pool by triggering the server's on Connect event. During the on Connect event, the connection details are provided and passed to the next layer based on the port number [13]. Another function of this layer is to update the TCP server's status to the intermediate layer in real time, allowing other system components to access the server's current state through the middle tier.

Application layer: When a connection module receives a connection request, it first stores the connection details to facilitate communication within the module. Next, the application module's frame resolution component is activated to interpret the data frame. The resulting object array from this parsing is then saved in the connection module's frame queue. Finally, the system signals the user that the main process has concluded [14]. At this point, the method available to external users can be executed. Users can enhance their application modules by overriding this method. Within the run method, users can link the array objects in the module's frame queue and transmit data to the sensor connection via the connection module. Additionally, to simplify user operations, this process incorporates many commonly utilized methods, such as encapsulating intermediate interaction methods, encapsulating methods for database interactions, and functionalities for sending emails and text messages. Essentially, all functions in this module revolve around the operational method, preparing data for the running method and streamlining overall system scheduling. The built in applications operate at the same level as standard applications, and they are fundamentally akin to user applications. The distinction lies in their protection, with access regulated by permissions. Among these applications, the most critical is the server management process, which allows remote starting, restarting, or shutting down the server by altering the application and issuing other significant commands to it. Furthermore, since Swoole is entirely implemented in PHP, the global variable memory can be easily retained. Consequently, a global registration tree is established in the system, integrating common factory design patterns and singleton design patterns to provide application level services, thus minimizing the overhead associated with constructing and destroying objects each time. When the application is initiated for the first time, the memory is retained, containing links and cached data frames for each link to facilitate object services and expedite processing. The middle layer serves as the segment that interfaces with the TCP server. The primary roles of this section include: caching the data frame submitted by the run method, triggering the uploaded *Hdfs* event once the configured threshold is exceeded; recording the server's running parameters and current status in real time; and serving as a means for data sharing between each connection module and application module [15].

4. Result Analysis and Discussion

TCP data receiving module is an essential module of the platform, which is responsible for the reception and pretreatment of the sensor data, and directly determines the network communication performance of the platform. Through testing, the performance transformation rule of the TCP server was explored, and then the best configuration of the TCP server in the current testing environment was summarized.

An important indicator for a network communication server is response time, the time it takes for the server to process user requests; in this platform, it is the time that the system processes the data frame. It is assumed that there are n sensors in the specified period; at the same time, the data frame of size s Byte is uploaded to the server. After receiving and processing the data, the server returns it as it is. The size is R Byte. Because the server consumes time on data processing: $r < n * s$. That is, the data returned by the server will be less than

the sum of data sent by the n sensors. It is not difficult to see that when n approaches infinity, the server's processing time approaches infinity, while the server returns the processed data, and the R approaches 0. So the server data processing ratio is $P = r / n * s$. The closer the P is to 1, the better the server performs. In the testing period, the data sent by the sensors is processed, and when P is close to 0, the server pressure is too enormous, causing congestion or even downtime.

The main performance parameters of the target server were: 4 cores CPU, power 1200MHZ, 64 bit processor, 128G memory. After receiving the data, the server can parse it and return it directly, resulting in better performance than in the actual production environment.

In the experiment, when n was minimal, each time increased 10 times, when the n value was bigger, the increment slowed down, 1, 10, 100, 200, 500, 600, 700, 800, 900, and 1000 were selected. To minimise the impact of each set of data connections and server fluctuations, the test time was 30 each time. The test results are shown in Table 1.

Test number	Concurrent quantity n	Send data (Byte)	The server returns data (Byte)	Data processing ratio P
1	1	798	798	100.00%
2	10	8445	8445	100.00%
3	100	81297	81279	99.97%
4	200	162057	159924	98.685
5	500	264516	246390	93.14%
6	600	251553	226368	89.98%
7	700	275304	202108	73.41%
8	800	277338	196623	70.89%
9	900	424434	219447	51.70%
10	1000	2118258	1002174	47.31%

Table 1. Test results for the server

The server's processing data rate during the sending time interval (100ms) decreased gradually as the concurrent amount increased. When concurrency increased to 1000, less than half of the data was processed in time. So, in the current environment, the best control of server concurrency was within 500. However, it was necessary to point out that sensor data acquisition rates rarely reached 100ms levels and thousands of

concurrent amounts simultaneously in the actual generation environment. When the acquisition frequency increases to 1s level, in theory, the amount of contemporary can reach about 5000. In addition, the data processing ratio parameter selected in this test was not particularly appropriate, which described the processing ratio at "the same time". In the actual production environment, a 100ms delay is acceptable, and the concurrency can be doubled to around 5000 in theory. To sum up, although TCP server performance gradually declines with increasing concurrency, it is sufficient to handle most actual production environments.

Hadoop distributed cluster is the base of data storage and processing in this platform. To reflect the system's actual operational environment, an experiment was designed and conducted, and various characteristics under a fully distributed Hadoop cluster were verified. The functions of data upload, view, dynamic additions and deletions of the server nodes and cluster status in the fully distributed mode of the Hadoop cluster were tested.

For convenience, in the experiment, the server as the central node was the entity machine, and the other two machines as the slave nodes were all virtual machines, among which, the host master (192.168.1.129), the two Slaves; Slave 1 (192.168.1.116), and the slave 2 (192.168.1.116). There was only qualitative analysis, so the virtual machine's configuration was relatively low.

After installing and configuring the three machines, the command *start dfs.sh* was entered on the master node to start the cluster. After the process starts, the web operation interface via the master node is shown in Figure 2. As can be seen, there were two active nodes, slave1 nodes and slave2 nodes, respectively. The data was uploaded to the cluster. The test text file test was uploaded to Hadoop, the command bin/Hadoop fs-put test/test was entered, and then, through the web interface, the file was seen.

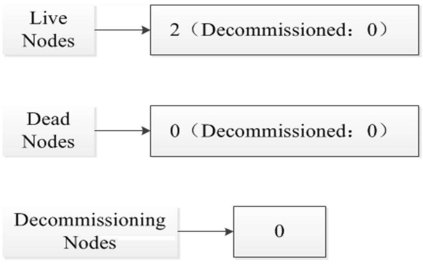

Figure 2. Hadoop cluster status

One of the key benefits of Hadoop is its flexible capacity scaling, which allows seamless addition and removal of nodes within the cluster. In the experiment, the Slave1 node was placed on the master node's blacklist and subsequently removed from the cluster. After this, it was taken off the blacklist and reintroduced to the cluster. Throughout this entire procedure, there was no requirement to restart or shut down the cluster, and the other nodes remained unaffected. The slave1 node was added to the master node's blacklist, specifically the data node deny list file. Following this, the command bin/hdfs dfsadmin refreshNodes was executed to prompt the cluster to reevaluate each node, and the web interface was refreshed. The slave2 remained in the In Service state, while the slave1 was labeled as Decommissioned. For recovery, it was removed from the blacklist, and the cluster section was refreshed as before. Tests were conducted, and nodes were incorporated into the cluster. To simplify testing, the cluster was started with only two nodes: master and slave1. Subsequently, slave2 was added to the master's list of slaves, and the data node process was activated on slave2. The cluster automatically checked the slave2 node and utilized its resources.

5. Conclusion

This paper explored the development of a universal management platform for Internet of Things (IoT) data. The functions and features of several essential technologies were examined. By combining analysis and abstraction of the primary components of IoT data management, configurable management for IoT data was achieved across data access, storage, processing, and display, facilitating further development of the underlying API. Simultaneously, the platform's main functionalities were tested, leading to the implementation of an actual project. The platform's primary functions and the characteristics of its secondary development were presented in detail. Through this research, several conclusions were drawn: a high performance TCP server was developed using PHP, and the functionalities for data analysis, caching, and preprocessing were designed and executed; to address the effective sharing of data across different servers and layers, an intermediate adaptation module was created; to allow users to construct IoT applications and access sensor data through straightforward configurations, the platform provided online service capabilities, including real time data monitoring, data analysis, and results display modules. To further improve the platform's functional cha- racteristics and performance, future research and optimization could explore various avenues, such as restructuring the API and minimizing the learning costs associated with secondary development.

References

[1] Sun, Q. B., Liu, J., Li, B., et al. (2010). Internet of things: concepts, architecture and key technology research review. *Journal of Beijing University of Posts and Telecommunications,* 33 (3), 1-9.

[2] Liu, Y. (2013). Internet of things in the era of big data. *High technology and industrialization,* 5, 10-15.

[3] Shvachko, K., Kuang, H., Radia, S., et al. (2010). The hadoop distributed file system. *IEEE Symposium on Mass Storage Systems Technologies,* 3, 1-10.

[4] Thusoo, A., Sarma, J. S., Jain, N., et al. (2009). Hive A Warehousing Solution Over a Map Reduce Framework. *Vidb Proceedings of the Vldb Endowment,* 2 (2), 162-1629.

[5] Lloyd, A. D., Sloan, T. M., Antonioletti, M., et al. (2013). Embedded systems for global e-Social Science: Moving computation rather than data. *Future Generation Computer Systems,* 29 (5), 1120-1129.

[6] HDFS. (2007). The hadoop distributed file system: Architecture and design. *Hdaoop Project Website,* 11, 1-10.

[7] Song, S. D., Guo, F. (2002). Java based WEB database connection pool technology research. *Computer engineering and application,* 38 (2), 201-203.

[8] Ren, Z. F., Zhang, H., Yan, M. S., et al. (2004). It summarized research of MVC pattern. *Computer application research,* 21 (10), 1-4.

[9] Cheng, C. R., Liu, W. J. (2009). High cohesion and low said in software architecture to build. *Computer system application,* 18 (7), 19-22.

[10] Thusoo, A., Sarma, J. S., Jain, N., et al. (2009). Hive A Warehousing Solution Over a Map Reduce Framework. *Vldb Proceedings of the Vldb Endowment,* 2 (2), 1626-1629.

[11] Lin, C., Hu, J., Kong, X Z. (2012). User quality of experience (QoE) model and the evaluation method of review. *Journal of computers,* 35 (1), 1-15.

[12] Castro, O., Ferreira, H. S., Sousa, T. B. (2014). Collaborative Web Platform for UNIX-Based Big Data Processing. *Lecture Notes in Computer Science,* 3, 199-202.

[13] Shan, J. H., Jiang, Y., Sun, P. (2014). Software testing research progress. *Journal of Beijing university: natural science edition, x,* 41 (1), 134-145.

[14] Dong, Y. Z., Zhou, Z. W. (2006). Based on X86 architecture system virtual machine technology and application. *Computer engineering,* 32 (13), 71-73.

[15] Li, D. L., Chen, R. (2009). WebSocket in Web research in the field of real time communication. *Computer knowledge and technology,* 9, 19-20.