Research on Performance Optimization for Power Big Data Storage based on HBase

Li Zhu 1,*, Xiang Cai 2 and Yuanxin Le 3

1 Hubei Key Laboratory for High-efficiency Utilization of Solar Energy and Operation Control of Energy Storage System, Hubei University of technology, Wuhan, 430068, China
2 School of Electrical and Electronic Engineering, Hubei University of technology, Wuhan, 430068, China
3 Ineering Bureau Co., Ltd. China Communications Construction Third Engineering Bureau Co., Ltd., China

* juliabiding@hbut.edu.cn

Abstract. With the development of smart grid, the online monitoring of power equipment has become the focus. Due to the increasing of power data, data storage in power grid is faced with great challenge. HBase is widely used for storing and managing power data due to its excellent performance in distributed access to data. However, HBase is prone to write a hot spot problem when storing data. To address this problem, this paper proposes an optimization method for data storage performance based on a dynamic load balancing strategy, which is implemented by pre-partitioning processing, RowKey substitution, and WLC (Weighted Least Connection) algorithm adapted to HBase storage mechanism. Through the experimental analysis of insulator current data storage, the results show that the method improves the performance of HBase cluster parallel data storage.

Keywords: Power data, HBase, Dynamic load balancing, Weighted Least Connection algorithm.

1. Introduction

With the rapid growth of the grid scale and the continuous advancement of intelligence, the online monitoring of power equipment has been greatly developed and has become a trend. Exponential growth in the monitoring data of power data acquired and transmitted in the device [1]. At the same time, the storage of power data faces the challenges of low latency and low blocking, and the power data has the characteristics of real-time, continuity, and large data volume [2]. How to store time sequence waveform signal data efficiently and reliably is an urgent problem in the field of power processing and big data processing.

HBase is an open source implementation module for BigTable in the Hadoop ecosystem. HBase is compatible with the distributed file system HDFS and the distributed data processing framework MapReduce and Spark [3], while retaining the high concurrency, high reliability, and high availability characteristics of BigTable when managing semi-structured and structured data [4]. As a result, HBase
is now widely recognized and used in the industry [5]. However, HBase, as a distributed database, is limited by the index structure and Region split mechanism, when storing streaming data such as power data, it is prone to write hot spot problem, that is, any period data storage tasks are handled only by a small number of RegionServer in the cluster. This leads to the blocking of certain nodes in the cluster due to high resource utilization, which severely slows down the rate of concurrent storage in the cluster [6]. Therefore, this paper proposes a dynamic load balancing strategy to solve the write hotspot problem of data in storage.

2. Related Work

The widely used storage systems in Hadoop ecology are file systems and databases, and the components corresponding to the above two categories are HDFS and HBase. HDFS and HBase components as a distributed system, how to make full use of its nodes is an urgent problem to be solved. Currently, load balancing techniques are often used to solve the aforementioned problem. Since HDFS and HBase storage mechanisms are similar and HBase is implemented with HDFS as the underlying layer, load balancing strategies for both of them can be mutually referenced. The load balancing strategies of HDFS and HBase can be divided into data migration and task allocation according to their implementation methods.

The load balancing strategy for data migration is to move some of the data on the overloaded nodes to the underloaded nodes. This approach is mainly applicable to scenarios where data is already stored in the cluster and the access mode of the scenario is to read more write less. Currently, most of the load balancing methods for HDFS and HBase are implemented in this way.

The load balancing policy for task allocation is consistent with the meaning of the load balancing policy defined in the domain of distributed systems, which is to allocate the service request tasks currently received by the cluster to each node in the cluster according to the load situation of each node to achieve load balancing for each node. The load balancing policy implemented in the form of load allocation is mainly applicable when the data access pattern is to write more read less.

3. Dynamic Load Balancing Strategy

Based on the research of existing data storage problems, this paper combines pre-partitioning, RowKey design and WLC algorithm adapted to the HBase storage mechanism to propose a dynamic load balancing strategy as shown in Fig 1.

![Fig. 1 Dynamic load balancing strategy](image)

3.1. Pre-partitioning.

When a RegionServer holds too many Regions, the CPU will consume a lot of time to switch back and forth between the threads corresponding to each Region according to the demand of data processing [7]. Therefore, the number of Regions is selected to be equal to the number of RegionServer in order to prevent an unreasonable number of Regions from being allocated in advance for a Table of unknown
size. Under the influence of the mechanism of polling and dispatching the Regions to the RegionServer, each RegionServer holds one Region during the initialization phase of Table creation. Considering that HBase needs to have the function of dispatching the current data storage request to the destination RegionServer, the combined data is arranged in the Rowkey dictionary order, and the highest order of the range of each region can be guaranteed to be different.

3.2. Rowkey Replacement.
RowKey replacement is to modify the structure and content of RowKey. As we all know, the CRUD (Create, Retrieve, Update, Delete) function of any database system is realized by a storage engine that relies on a sort structure and its algorithm. The HBase system is implemented with a storage engine based on the LSM-Tree (Log Structured-Merge Tree) sorting structure, and the RowKey is the only reference for this storage engine. Therefore, the good or bad design of RowKey will directly affect the distribution of data in each RegionServer and the performance of data read and write. Based on the consideration of the characteristics of the power data and the RowKey sorting method, this paper uses the prefix and hash method to replace the original RowKey of the power data.

3.3. WLC Algorithm Adapted to HBase Storage Mechanism.
This paper combines with HBase’s own data storage mechanism to improve the WLC algorithm from three aspects: weight setting, load feedback mechanism, and task allocation mechanism.

3.3.1. Weight Setting. Since there is some bias in the artificially set weights $W_{pre}(S_i)$ for measuring the processing capability of nodes, a set of functions for quantifying the processing performance of machine nodes is used to replace $W_{pre}(S_i)$ in this paper, and the expression of the function is as follows [7].

$$P_{cpu}(S_i) = \begin{cases} F_{cpu}(S_i) \\ (1 + \sum_{S_{i}=2}^{S} (\rho + \delta \times e^{2-N_{core}(S_i)}) \times F_{cpu}(S_i) N_{core}(S_i) > 1 \\ P_{mem}(S_i) = N_{mem}(S_i) \times F_{mem}(S_i) \\ W_{post}(S_i) = [r_i, r_j] - \frac{P_{cpu}(S_i)}{\max(P_{cpu}(S_i))} \times \frac{P_{mem}(S_i)}{\max(P_{mem}(S_i))} \end{cases} \quad (1)$$

Where $P_{cpu}(S_i)$ is the node CPU processing performance function; In formula (1), $F_{cpu}(S_i)$ represents the main frequency of the CPU configured by the node $S_i$, in GHz; $N_{core}(S_i)$ represents the number of cores of the CPU configured by the $S_i$. In addition, in a multi-core CPU, the performance of each core is only about 0.8 to 0.9 times that of a single-core CPU. Therefore, this article sets the values of coefficients $\rho$ and $\delta$ to 0.8 and 0.1, respectively.

$P_{mem}(S_i)$ is the memory processing performance function of the node. In formula (2), $N_{mem}(S_i)$ represents the size of the memory space configured by the node $S_i$, in MB; $F_{mem}(S_i)$ represents the memory frequency configured by node $S_i$, in GHz.

$W_{post}(S_i)$ is the comprehensive processing performance function of the node. In formula (3), $\max(P_{cpu}(S_i))$ is the maximum function value of the CPU processing performance of the node in the cluster, and $\max(P_{mem}(S_i))$ is the maximum function value of the memory processing performance of
the node in the cluster; parameter $r_1$ to $r_6$ represents the degree of influence of
\[
\frac{P_{\text{cpu}}(S_c)}{\max(P_{\text{cpu}}(S_c))}
\]
and
\[
\frac{P_{\text{mem}}(S_c)}{\max(P_{\text{mem}}(S_c))}
\]
on function $W_{\text{post}}(S_c)$ respectively.

3.3.2. Load Feedback Mechanism. Regarding the definition of load, referring to the definition of load
in distributed systems, it is divided into task-oriented load and workload-oriented load. By considering
the actual production environment, this paper chooses one of the work-oriented node loads, which has
the specific meaning of the consumption of node resources.

Regarding the evaluation of load, this paper uses a comprehensive load function $l(S_c, t)$ to quantify
the load size of a node, which selects four metrics: CPU usage, memory usage, disk I/O occupancy, and
disk usage as references, and the comprehensive load function $l(S_c, t)$ expression is as follows.

\[
l(S_c, t) = \lfloor r_1, r_2, r_3, r_4, \max(l_1(S_c, t), l_2(S_c, t), l_3(S_c, t), l_4(S_c, t)) \rfloor
\]

Where $l_1(S_c, t)$, $l_2(S_c, t)$, $l_3(S_c, t)$, and $l_4(S_c, t)$ represent the numerical values of the above-
mentioned four indicators of the node $S_c$ in period $t$, and parameter $r_1$ to $r_4$ represents the degree of
influence of the four indicators on function $l(S_c, t)$, and $\sum r_i = 1$.

3.3.3. Task Allocation Mechanism. In order to maximize the performance of HBase's concurrent
processing of power data storage tasks when the number of nodes is limited, all RegionServers need to
have the possibility of being assigned tasks at any moment. This paper considers changing the allocation
method of the optimal RegionServer in a specified period in the traditional WLC algorithm to the
allocation method of the proportion of the amount of data stored in each RegionServer in a specified
period. The aforementioned method changes the basis of task allocation from $m(S_c)$ to task allocation
function $v(S_c, t)$, the formula is as follows.

\[
v(S_c, t) = \frac{1}{n(S_c, t)} \sum_{S \in n} \left( \frac{1}{n(S_c, t)} \right)
\]

Where $v(S_c, t)$ represents the ratio of the amount of written data allocated $S_c$ in the period $t$ to
the total amount of written data.

In order to realize that in a certain load information update cycle, the cluster allocates the amount of
stored data to each RegionServer according to a specific ratio. This article considers establishing a
benchmark data set on the Region. At any time, the amount of each benchmark data set to hit by the data
storage task needs to be approximately equal, so as to ensure that the number of approximate benchmark data sets
allocated to each RegionServer can be determined by controlling the number of tasks allocated to each
RegionServer's data storage.

Consider the design part of RowKey, this article uses the MD5 algorithm to process the RowKey,
replacing the original RowKey with a 16-digit hexadecimal number. Since the HashCode obtained after
processing a piece of continuous data can be approximately uniformly mapped to the entire HashCode
value space, and HBase uses lexicographical order to organize the data. Therefore, the entire RowKey
value space is divided into 16 blocks according to the highest value of the HashCode obtained by the
MD5 algorithm, namely 0 to 15, each block corresponds to 1 reference data set, and the aforementioned
16 reference data sets are used as the logical task allocation unit. The task allocation algorithm
determines the number of logical task allocation units to be allocated according to the allocation function
to complete the task allocation control.
However, considering that the MD5 algorithm cannot guarantee that the highest bits of HashCode corresponding to the stored data RowKey in each load information update cycle can be uniformly distributed between 0 and \( f \). Therefore, regarding the selection method of logical task allocation unit, this paper adopts the random selection method, that is, \( S_e \) randomly selects the corresponding number of logical task allocation units with the highest bit of 0 to \( f \) according to \( v(S_e, t) \). The implementation of allocation is completed by adding the corresponding prefix of \( S_e \) to the HashCode.

4. Experimental Analysis
This paper takes insulator current data in the transmission line monitoring system as an example, and writes a program for the event of storing insulator current data by calling the client interface provided by HBase. Because the insulator current acquisition equipment performs 200 samples within each operating frequency, the insulator current data model is designed as shown in Table 1.

| Table 1. Insulator Current Data Model |
|-----------------|-----------------|-----------------|
| RowKey          | ColumnFamily: Climate | ColumnFamily: Signals |
| Time1+Mac_id+C_id | Temperature     | V_1             |
|                 | Humidity        | …               |
| Time1+Mac_id+C_id | t1              | V_1             |
|                 | h1              | …               |
| Time1+Mac_id+C_id | v_1             | …               |
|                 | …               | V_200           |

Considering that this experiment has three servers maintaining the RegionServer process, the data generated by the three insulator current acquisition devices are stored, and each device deposits data to the server through a single thread. The total amount of data deposited by the three client threads is selected as 1 million, 5 million, and 10 million lines.

In order to prevent the effects of the copy mechanism in HDFS on this paper, this experiment sets the number of copies of files generated in HBase to 1. The aforementioned impact is that each HFile has 1 primary copy and 2 secondary copies distributed on other nodes by default. Therefore, the storage of each piece of data will cause the resource consumption of other nodes except the allocation node. In addition, considering the limited amount of data stored in the experiment, the Split threshold is adjusted to 1 GB.

This experiment uses native HBase, HBase with pre-partitioning and RowKey replacement, and the method proposed in this paper for comparison, and the experimental results are shown in Fig 2.

![Fig. 2 Comparison of Insulator Current Data Storage Time in Heterogeneous](image)

As can be seen from Fig 2 that compared to native HBase, HBase that uses pre-partitioning processing and RowKey replacement have greatly reduced the total time consumed for data storage, and the difference increases with the increase in the amount of stored data. The HBase that uses the time series waveform information number data storage performance optimization method based on the dynamic load balancing strategy mentioned in this article, compared with the HBase that only uses the
pre-partition and RowKey design, the data storage consumption time is slightly reduced, and the difference varies with storage. As the amount of data increases, it can be seen that when storing insulator current data, using the optimization method proposed in this article can increase the data storage rate of HBase compared to the traditional pre-partitioning process and RowKey replacement and only RowKey replacement.

In order to solve the reasons for the rate increase, this experiment measures the number of data storage tasks processed by each RegionServer in each period during the storage process, and further analyzes the working status of each RegionServer. In this experiment, the storage of 5 million pieces of data is taken as an example, and the number of data storage tasks processed by each RegionServer in the current cycle is collected with a period of 100 seconds as shown in Fig 3 to Fig 5.

As can be seen in Fig 3, Fig 4 and Fig 5, that when native HBase stores insulator current data, only 1 to 2 RegionServers work in any period of time. The foregoing phenomenon verifies that HBase has the problem of writing hot spots when storing insulator current data. When only the pre-partitioning process and the HBase storage insulator current data replaced by RowKey are used, the ratio of storage tasks processed by $S_1$ and $S_2$ within 0-200s is $6/5$. The reason for the foregoing phenomenon is that the proportion that the data storage task submitted by the client thread hits each RegionServer is near $6/5$. The difference in the number of storage tasks processed by $S_i$ between periods is very small, and the amount of storage tasks processed in any period is much different than that of $S_1$ and $S_2$. The reason for the foregoing phenomenon is that $S_1$ is limited by hardware performance, and the amount of storage tasks processed per unit time is limited.

When the HBase storage insulator current data with the optimization method proposed in this article is adopted, the duration of $S_1$, $S_2$, and $S_i$ processing data storage tasks is approximately the same, and the amount of processing storage tasks in each period is the same as when the original HBase stores insulator current data. The peaks are approximately equal. The reason for the foregoing phenomenon is that after the optimization method proposed in this article is adopted, data storage tasks are allocated...
according to the processing performance of each RegionServer at the current time, resulting in $S_1$, $S_2$, and $S_3$ being able to give full play to their processing task capabilities at any time.

5. Summary

This paper proposes a method to optimize the storage performance of power data based on a dynamic load balancing strategy that combines the three technologies of pre-partition processing, RowKey replacement, and dynamic load balancing. In order to enable the dynamic feedback load balancing technology to be implemented under the storage mechanism of HBase, this paper carries out the corresponding pre-partition processing and RowKey replacement design. At the same time, according to the HBase operating mechanism, the WLC algorithm has been improved in three aspects: weight setting, load feedback mechanism, and distribution mechanism. Through the aforementioned method, the effect of assigning an appropriate amount of data storage tasks to each working node according to the busyness of each RegionServer is realized, so that the processing capabilities of each node in the cluster are fully utilized, and the performance of the cluster parallel processing power data storage tasks is improved.

References

[1] S. J. Sarkar, P. K. Kundu and G. Sarkar. Development of lossless compression algorithms for power system operational data. in IET Generation, Transmission & Distribution, vol. 12, no. 17, pp. 4045-4052, 2018.
[2] F. Lixin, X. Chao, Y. Chao, J. Chen and Y. Dahai. Synchronous Condenser Vibration Signal Acquisition and Storage Method. 2019 IEEE 3rd Conference on Energy Internet and Energy System Integration (EI2), Changsha, China, 2019, pp. 2197-2200.
[3] A. Verma, A. H. Mansuri, N. Jain. Big data management processing with Hadoop MapReduce and spark technology: A comparison. Colossal Data Analysis & Networking. IEEE, pp. 1-4, 2016.
[4] George, Lars. HBase: The Definitive Guide. Andre, vol. 1, no. 12, pp. 1-4, 2011.
[5] J. Jin, et al. Distributed storage system for electric power data based on Hbase. in Big Data Mining and Analytics, vol. 1, no. 4, pp. 324-334, December 2018.
[6] Z. Xiong, G. Zhu, W. Yu, S. Wang and Z. Chong. Load-Balanced Cluster for Scale-Out Storage of Knowledge. 2018 Sixth International Conference on Advanced Cloud and Big Data (CBD), Lanzhou, pp. 1-5, 2018.
[7] E. Zdravevski, P. Lameski, A. Kulakov. Row Key Designs of NoSQL Database Tables and Their Impact on Write Performance. Euromicro International Conference on Parallel. IEEE, 2016.