Distributed Method for the Backup of Massive Unstructured Data

Junhong Duan* and Bo Yang

State Grid Gansu Electric Power Information and Communication Company, Lanzhou, China

*Corresponding author: duanjh@gs.sgcc.com.cn

Abstract. In order to solve the existing problems for the backup of massive unstructured data with its performance bottleneck of single server, in this paper we propose a distributed method for the backup of massive unstructured data. In the backup method, through using a combination of load balancing algorithm and distributed algorithm, the backup system assign task of backup to each production server. Multiple servers backup massive unstructured data stored in shared storage to the backend server. The backup system can make full use of the performance resources of servers in environment production, reduce the single server performance bottlenecks, to avoid the single server hunger, increased the speed of backup.

Keywords: Massive unstructured data, Backup, Distributed algorithm, Load balancing algorithm

1. Introduction

With the rapid development of information technology, the amount of data is increasing rapidly. According to Intel's prediction [1-2], the global data volume will reach 44ZB (1ZB=10^9TB=10¹²GB) in 2020, and the annual growth rate of data volume has been around 50% since 2012. The data is divided into structured data and unstructured data. Unstructured data is data with irregular or incomplete data structure and no predefined data model. It is not convenient to represent data with two-dimensional logical tables of the database, including office documents, text, pictures, XML, HTML, various reports, images, audio/video information, etc. Its format is very diverse, and the standard is also diverse, and technically unstructured information is more difficult to standardize and understand than structured information. In the backup scheme, unstructured data is more irregular than traditional structured data, so customized backup software is needed to make long-term archival backup. If the amount of data is close to massive, when the backup efficiency is high, there is also a corresponding increase in the demand for the production server's own resources (such as memory, CPU, storage IO). How to ensure the production server has a loose concurrent space while efficiently backing up data puts forward higher requirements for the backup method.

In order to optimize the backup performance of massive unstructured data, at present, it is mainly through parallel, multi-channel and multi-process cooperative execution of a backup task to improve the data flow that can be operated at the same time to improve the overall backup efficiency. The existing backup method for production server data in mass unstructured data environment is mainly to
create a single process (multiple working sub-threads can be used internally) to execute a backup task; Or a backup task is executed in parallel by creating a plurality of execution processes (in parallel mode); These two methods have single-point server performance bottleneck [3], that is, although in a single server environment, hardware resources such as memory, CPU, disk and the like can be fully utilized to complete specific backup tasks, the concurrent performance of the entire server in a certain period of time will not be fully utilized. If the entire server only performs a specific backup task, the concurrent performance is acceptable. If multiple backup tasks or multiple types of tasks need to be performed, the entire server will experience severe performance overload. Each task frequently requests time slice, CPU, memory, and IO will also be frequently scheduled for switching, thus causing the overall performance of the server to decline.

To overcome the shortcomings in the prior art, a distributed massive unstructured data backup method and system are proposed to solve the technical problem of performance bottleneck of the existing single point server and ensure load balancing of backup clusters [4].

2. System Structure
The whole system consists of five modules: performance weight calculation module, backup task creation module, backup task decomposition module, backup task execution module and resource recovery module. The system structure diagram is shown in fig. 1.

![System structure diagram](image)

**Figure 1. System structure diagram**

The performance weight calculation module is used for calculating and determining the performance weight corresponding to each production server according to the performance indexes of each production server; The backup task creation module is use for creating a backup proces for that backup task to generate a corresponding backup strategy when one of the production servers receives the backup task of backing up the data in the share storage to the back-end server; The backup task decomposition module is used for decomposing the backup strategy into a plurality of sub-processes for execution and distributing all the sub-processes to each production server according to the performance weight of each production server; The backup task execution module is use for executing all subprocess in each production server at that same time so as to backup the data in the shared storage to the back-end server; The resource recovery module is use for releasing that resource of each sub-process and the backup process after the backup task is completed.
2.1. Performance weighting module
The performance weight module is responsible for calculating the performance weight of the production server and providing basis for subsequent task decomposition. The performance weight module load balancing server adopts the client-based load balancing technology [8] and a specific value calculated by the load balancing service weighted polling algorithm [5-7] according to the performance index of the production server, wherein the larger the performance weight value, the better the performance of the production server.

The architecture of load balancing technology is shown in Figure 2.

![Figure 2. Architecture diagram](image)

The production server registers its own performance index with the registration center in the load balancing server regularly, and the load balancing server calculates and determines the performance weight of each production server in the current state according to the performance index of each production server. The performance weight here is a specific value calculated by using load balancing service algorithm in combination with the performance indexes of specific production servers, such as memory, disk, CPU, etc.; for example, the range can be 1-100; The higher the numerical value, the higher the representative weight, and the better the performance.

2.2. Backup task creation module and decomposition module
The backup task creation module is responsible for creating the backup process. The backup task creation module includes a process manager. In the backup process, when the production server receives a backup task, it creates a backup process through the process manager and registers the backup task with the distributed collaborative scheduling server.

The backup task decomposition module includes a distributed collaborative scheduling server. The distributed cooperative scheduling server is responsible for storing resource weight information. When the backup task is decomposed, the decomposition backup module allocates task subprocesses according to the performance weight. The larger the performance weight value of the production server, the more subprocesses are allocated.

2.3. Backup task execution module
The backup task execution module adopts the zab(Zookeeper Atomic Broadcast) algorithm in Zookeeper [9-12]. Zab algorithm is an improved algorithm based on paxos [13] algorithm to ensure the consistency of distributed data.
The system uses zab algorithm to select one sub-process as the main sub-process among all sub-processes, and the other sub-processes are sub-processes to perform backup tasks in a master-slave manner.

There are four roles in Zab algorithm: leader, learner and client. Leader is responsible for initiating bids and resolutions and updating system status; Learn is divided into follower and observer, follower is responsible for accepting the client's request and returning the result, and participates in voting in the process of selecting leader; Although the observer is also connected with the client, it is different from the follower in that the observer is responsible for forwarding the request to the leader and does not participate in the voting to select the leader. Client is responsible for initiating the request.

The entire execution process of Zab algorithm is divided into two modes:

1. Recovery mode
When the backup task is started or the original master process or leader crashes, the zab algorithm enters the recovery mode to re-elect the leader, and the recovery mode ends when the master process and the slave process are synchronized.

2. The process of selecting the master sub-process is shown in Figure 3.

![Figure 3. Election main process flowchart](image-url)
Start all processes to read their own zid and other information, and recommend themselves as leader, and then broadcast (leader id, zid, and epoch) as broadcast information to all processes in the cluster. Then wait for the processes in the cluster to return information.

When a process receives information from other processes in the cluster, it is divided into two types: the process is in a looking state or in another state.

1) The process is in a looking state.
   First, judge the logical clock Epoch:
   a) If the received Epoch is larger than the current logic clock (indicating that the logic clock saved by oneself is outdated). Update the local logical clock Epoch and Clear the election data sent by other processes (these data are outdated). Then judge whether it is necessary to update the current election situation (the leader id selected at the beginning is yourself); According to the judgment rule: the stored zid maximum value and the main process id are used for judgment. First look at the data zid and select the process with large value. Secondly, judging the id of the main process and selecting a process with a large value; Then broadcast their latest election results (i.e. the above-mentioned three data (process id, Zxid, Epoch) to other processes)
   b) If the received Epoch is smaller than the current logic clock. This indicates that the other party is behind in the number of rounds of the election, and only needs to send his own (process id, Zxid, Epoch) to him.
   c) If the received Epoch is equal to the current logic clock. According to the judgment rule in a), the latest election results will be broadcast to other processes. At the same time, the process has to deal with two situations: 
      a) If the process receives the election information of all other processes, then according to the election information, it determines Following, Leading, ends Looking, and quits the election.
      b) Even if the election information of all the processes is not received, it can be judged whether the latest election leader after the above process is supported by more than half of the processes. If so, try to accept the latest data. If there is no latest data, it means that everyone has acquiesced in the result and has also set up a role to withdraw from the election process.

2) The process is in another state (Following, Leading)
   a) If the logical clock Epoch is the same, save the data to recvset. If the process claims to be leader, it will determine whether more than half of the processes elect it, and if so, set the election status to exit the election process.
   b) Otherwise, this is a message that does not conform to the current logic clock, which indicates that the election result has already been obtained in another election process. Then the election result is added to the outofelection set, and whether the election can be ended is judged according to the outofelection. If so, the logic clock is saved, the election state is set, and the election process is exited.

Broadcast mode

After the election of a new Leader and completion of state synchronization, zab algorithm enters broadcast mode. The broadcast mode flow is shown in fig. 4.
1) The client initiates a request.
2) Leader converts the client's request into a transaction Proposal and assigns a global ID, zxid, to each Proposal.
3) Leader assigns a separate queue for each Follower, and then puts the Proposal to be broadcast into the queue in turn, and sends messages according to the FIFO strategy.
4) After the Follower receives the Proposal, it will first write it to the local disk in the form of a transaction log. After successful writing, it will feedback an Ack response message to the Leader.
5) After the Leader receives more than half of the Follower's Ack response message, it believes that the message was sent successfully and can send a commit message.
6) Leader broadcasts the commit message to all Follower and completes the transaction commit itself. Follower commits the previous transaction after receiving the commit message.

2.4. Resource recovery module
The resource recovery module is responsible for releasing the resources of each sub-process and backup process after the backup task is completed.

3. Backup Process
The complete backup process is shown in fig. 5, assuming that there are 2 production servers in the entire production environment defined as production server 1 and production server 2 respectively, the backup task is to backup the massive unstructured data in distributed shared storage to the back-end server storage.
1) The production server 1 and the production server 2 register their own performance indexes with the load balancing server regularly, and the load balancing server calculates and determines the performance weight of each production server in the current state according to the performance indexes of each production server. After calculation, the performance weight of the production server 1 is 1, and the performance weight of the production server 2 is 3.

2) When the production server 1 receives a backup task, it creates a backup process 1 through its process manager, and the process manager registers this backup task with the distributed collaborative scheduling server; This backup task is to backup massive unstructured data from shared storage NAS/SAN to back-end servers. When the production server receives the backup task, the process manager sends the backup strategy information (such as backup source, etc.) of the backup task to the distributed collaborative scheduling server; The distributed server keeps records of tasks and is managed uniformly by the server. After the distributed collaborative scheduling server stores the task registration information, it manages the subsequent distributed subprocesses and notifies the process manager of the progress of the entire task.

3) The distributed collaborative scheduling server obtains respective performance weights of registered production servers by querying the load balancing server, and organizes a task decomposition plan according to the performance weights: The higher the weight, the more subprocesses the production server gets, and the backup strategy of backup process 1 is divided into 5 subprocesses for execution (1-1, 1-2, 1-3, 1-4, 1-5 subprocesses, which can also be called the 1st, 2nd ... 5th subprocesses respectively); Send 5 sub-processes to production servers 1 and 2 respectively for execution (at this time, server 1 is assigned to 1-1 and 1-2 task sub-processes; Three sub-processes of tasks 1-3, 1-4 and 1-5 are allocated on server 2);

On the one hand, the number of decomposition subprocesses is controlled by the weight of the production machine. On the other hand, backup strategies are generally distributed according to directories or multiple file paths. You can split the directory into multiple subprocesses. Can also assign different file paths to different subprocesses.

4) The distributed collaborative scheduling server selects one sub-process and other sub-processes as sub-processes of the assigned tasks by combining the distributed consistency election algorithm (because the entire decomposed backup task is executed in a distributed environment, the consistency

---

**Figure 5. Backup flowchart**
of the entire backup task needs to be ensured); The main sub-process and each sub-process are executed synchronously, and the backup data is read from the shared storage and stored to the back-end server according to the backup policy.

Through the consistent election method in the distributed algorithm, each process that the whole task is decomposed is always executed in the master-slave mode (in case of abnormality in the process, the master-slave election is re-conducted), thus achieving the consistency of distributed execution of the whole decomposed task.

The data backed up by each sub-process is independent, and the data generated by each process will be stored in the back-end server in the form of storage data objects. The index library in the back-end server records metadata information of backup data, such as file name, size, and data offset in the back-end server. According to the records in the index library, the data in the storage can be read and written.

5) During the whole backup task execution process, each sub-process reports the current backup progress to the main sub-process (e.g. the progress ratio of completed backup data, the number of files, the number of directories, etc.), and the main sub-process is responsible for reporting the current overall backup task progress to the process manager of the production server 1 (mainly the data progress ratio of backup, and some abnormal errors);

6) Finally, with the completion of the backup task of each sub-process, the entire backup task also ends.

7) Production server 2 reclaims resources from subprocesses 1-3, 1-4, 1-5;
8) Production server 1 reclaims subprocess 1-1, 1-2 resources.
9) Generation server 1 reclaims backup process 1 resources.

4. System Simulation Experiment
The backup system consists of server and client, and adopts C/S architecture. The experiment plans to test the speed and performance of the system in data backup and compare it with traditional backup software.

4.1. Test environment configuration
The whole experiment consists of 3 testing machines. The configuration of the tester is shown in Table 1, Table 2 and Table 3.

**Table 1. Test machine configuration**

| Category               | Projects            | Specifications                                      |
|------------------------|---------------------|----------------------------------------------------|
| Hardware environment   |                     |                                                    |
| Main board             | S3420GPV            |                                                    |
| CPU                    | Intel(R) Xeon(R) CPU X3430 @ 2.40GHz |                      |
| Hard disk              | One SATA3T          |                                                    |
| Network card           | Yu Shuo Killer E2500 Gigabit Ethernet Controller / Jijia |    |
| Memory                 | 24G                 |                                                    |
| Graphics               | Nuclear display     |                                                    |
| Software environment   | Operating system    | Windows 2012 r2(64bit)                           |
| Network environment    | Switch              | Gigabit                                           |
| Ip address             | 10.10.1.63          |                                                    |
Table 2. Test machine configuration

| Category           | Projects       | Specifications                          |
|--------------------|----------------|-----------------------------------------|
| Hardware environment | Main board     | Z270 gaming 3                          |
|                    | CPU            | I7-7700K CPU @ 4.20GHz               |
|                    | Hard disk      | One SSD 120GB                         |
|                    | Network card   | Yu Shuo Killer E2500 Gigabit Ethernet Controller / Jijia |
|                    | Memory         | 32G                                     |
| Software environment | Operating system | Windows 2012 r2(64bit)      |
| Network environment | Switch        | Gigabit                                |
|                    | Ip address     | 10.10.6.171                            |

The test agent configuration is shown in Table 3.

Table 3. Test machine configuration

| Category           | Projects       | Specifications                          |
|--------------------|----------------|-----------------------------------------|
| Hardware environment | Main board     | Z270 gaming 3                          |
|                    | CPU            | I7-7700K CPU @ 4.20GHz               |
|                    | Hard disk      | One Sata1T                             |
|                    | Network card   | Yu Shuo Killer E2500 Gigabit Ethernet Controller / Jijia |
|                    | Memory         | 32G                                     |
| Software environment | Operating system | Windows 2012 r2(64bit)      |
| Network environment | Switch        | Gigabit                                |
|                    | Ip address     | 10.10.7.177                            |

4.2. Data backup speed and system performance test
The backup and recovery test results of 20w files under Window system are shown in Table 4.

Table 4. Backup result

| Backup | Backup time | Backup set size | Average speed | Total execution time |
|--------|-------------|-----------------|---------------|----------------------|
| The optimized method | 1012s | 96.96GB | 98.1MB/s | 1015s |
|         | 1061s  | 96.96GB | 93.6MB/s | 1063s  |
| Traditional method | 1956s | 96.96GB | 50.76MB/s | 1959  |
|         | 2035s  | 96.96GB | 48.7MB/s | 2037  |

The system performance test results are shown in Table 5.

Table 5. System performance test results

| Category           | Testing machine | Memory usage | CPU occupancy |
|--------------------|-----------------|--------------|--------------|
| The optimized method | 1              | 489MB        | 27%          |
|                    | 2              | 878MB        | 32%          |
|                    | 3              | 914MB        | 35%          |
| Traditional method  | 1              | 1381MB       | 76%          |
|                    | 2              | 358MB        | 14%          |
|                    | 3              | 826MB        | 29%          |
It can be seen from the experiment that the distributed backup method based on load balancing is obviously improved compared with the traditional backup method in terms of backup speed. Similarly, three test machines are used for collaborative backup. The optimized method effectively balances the test machines and avoids the problem of excessive consumption of single machine performance.

5. Summary
Aiming at the single-point performance bottleneck in traditional massive unstructured data backup methods, this paper proposes a distributed massive unstructured data backup method. By adopting client-based load balancing technology and zab distributed consistency algorithm, when facing multiple backup tasks, the task is divided into multiple subprocesses. The system allocates subprocesses according to server performance, and all servers cooperate to complete the backup tasks, avoiding starvation or overload of a single server, thus improving the overall backup speed.

Acknowledgments
This work is funded by the State Grid Corporation headquarters science and technology project (research on key technologies of highly available self-service database cloud based on physical resources).

References
[1] GANTZ J, REINSEL D. The digital universe in 2020: big data, bigger digital shadows, biggest growth in the far east [C/OL]. IDC iView: IDC Analyze the FutureIDC, 2012 [2015-05-20]. http://www.emc.com/collateral/analystreports/idc-the-digital-universe-in-2020.pdf
[2] Lillibridge M, Eshghi K, Bhagwat D, et al. Sparse indexing: Large scale, inline deduplication using sampling and locality//Proc of the 7th Conf on File and Storage Technologies (FAST’09). Berkeley, USA: USENIX Association, 2009: 111-123
[3] NI Bing, CHEN Yun-qin. Research and Test on P2P Traffic Monitoring and Control Technology[J]. Journal of Jilin University: Information Science Edition, 2009, 27(5):544-549.
[4] ADDISIE A, BERTACCO V. Collaborative accelerators for in-memory MapReduce on scale-up machines [C]//Proceedings of the 24th Asia and South Pacific Design Automation Conference. New York: ACM, 2019: 747-753.
[5] The guide of load balancing (Part One) [EB/OL]. http://cisco.chinaitlab.com/case/12415.html. 2003-05.
[6] The guide of load balancing (Part Two) [EB/OL]. http://cisco.chinaitlab.com/case/12416. html.2003-05.
[7] Cai Weihua, Ma Le, Wang Hua, et al. Numerical Simulation Model of Ship Navigation Performance Based on IGES Document [J]. Computer applications and software, 2012, 29 (2): 192-194, 236.
[8] Mao Kejia. Research on Client Load Balancing Algorithm and Its Application and Implementation in Instant Messaging Architecture [D]. Chengdu: University of Electronic Science and Technology, 2018.
[9] Prisco R D, Lampson B W, Lynch N A. Revisiting the Paxos algorithm [C] //Proceeding of the 11th International Workshop on Distributed Algorithm. Spring-Verlag, 1997: 111-125.
[10] Tang Hai-dong, Wu Yanjun. Optimization of Zookeeper Distributed Synchronization System [J]. Computer Engineering, 2014, 40 (4): 53-56.
[11] Apache Software Foundation. Apache Zookeeper [EB/OL]. (2013-02-01). http://Zookeeper.apache.org/.
[12] Hunt P, Konar M, Junqueria F P, et al. Zookeeper: Wait- free Coordination for Internet-scale Systems[C]//Proceedings of the 2010 USENIX conference on USENIX annual technical conference. USENIX Association, 2010: 11.
[13] Lamport L. The Part-time Parliament [J]. ACM Transaction on Computer Systems, 1998, 16 (2): 133-169.