Cache Management of Big Data in Equipment Condition Assessment

Yan Ma[a], Yufeng Chen, Ying Lin, Chengqi Li, Yujie Geng

State Grid Shandong Electric Power Research Institute, Jinan, China

Abstract. Big data platform for equipment condition assessment is built for comprehensive analysis. The platform has various application demands. According to its response time, its application can be divided into offline, interactive and real-time types. For real-time application, its data processing efficiency is important. In general, data cache is one of the most efficient ways to improve query time. However, big data caching is different from the traditional data caching. In the paper we propose a distributed cache management framework of big data for equipment condition assessment. It consists of three parts: cache structure, cache replacement algorithm and cache placement algorithm. Cache structure is the basis of the latter two algorithms. Based on the framework and algorithms, we make full use of the characteristics of just accessing some valuable data during a period of time, and put relevant data on the neighborhood nodes, which largely reduce network transmission cost. We also validate the performance of our proposed approaches through extensive experiments. It demonstrates that the proposed cache replacement algorithm and cache management framework has higher hit rate or lower query time than LRU algorithm and round-robin algorithm.

1 Introduction

With the rapid development of power grid and the explosion of transmission and transformation equipment, massive and heterogeneous data are generated and separately collected into different business information systems during grid operation and equipment monitoring. In order to make full use of various data from different systems, we build a big data platform for equipment condition assessment. Now it connects production management system (PMS), energy management system (EMS), condition monitoring system of transmission and transformation equipment, geographic information system (GIS) and meteorlogical information system, and is used to research dynamic evaluation of load capacity, fault diagnosis, condition evaluation and risk assessment [1]. For the application with high requirements for response time such as dynamic evaluation of load capacity, users have requirements for processing performance and return time. Therefore, we need to optimize big data processing performance to improve data analyzing efficiency.

Cache is one of the most important means of improving big data processing speed [2]. Data are stored in cache, which largely improve I/O efficiency and then data processing performance. However, cache is more expensive than external storage devices such as disk. Moreover, big data is total sample data. It is uneconomic and infeasible that all the data are stored into cache. In general data accessing is

[a] Corresponding author : yanpony@126.com

frequent and real-time just for a portion of the data. Thus we could put important and frequently accessed data into cache.

Compared with traditional data caching, big data caching has its own characteristics.

1) Data is organized by key-value pair. The caching granularity, caching modality and cache replacement algorithm need discussing to adapt to storage structure of big data.

2) Big data processing relies on cloud computing platform. Data accessed in big data have a certain correlation. It can reduce data transferring cost to put the related data on adjacent positions. For example, a data processing needs data A and B. If data A and B are stored on two nodes, one data should be transferred to the other node. If data A and B are stored on one node, network transmission is avoided and processing efficiency is high. After we get the cached data, a method of placing them on proper nodes needs designing.

2 Related work

Database cache is widely studied to improve database performance [3-5]. Storm et al. [3] gives a self-adjusting cache management mechanism, which builds the model between consumed time and output profits for different users. Malik et al. [4] define hit output rate by unit of byte, which caches the caching objects with high output to reduce network transmission cost. Brown et al. [5] propose fragment fencing to solve cache assignment for different type of applications with QoS requirements.

With the rise of cloud computing, there are shortcomings for cache management of traditional database [6]. On one hand, cache resources are shared and competed by multiple users, which results in low hit rate and poor user experience. On the other hand, there lacks shared cache management strategy for multiple users, which lets users with low query rate have low hit rate. Thus, Yao et al. [6] propose a self-adaptive cache management based on chunk folding architecture, which dynamically generates a series of cache replacement units to consistently achieve users' SLA requirements.

As for cache replacement strategy, the common cache replacement algorithms have LRU, LFU, LRU-K [7]. All the algorithms use pages as replacing unit to minimize page lose rate. LRU algorithm sometimes pollutes cache pool when reading a series of infrequently used pages. The InnoDB [8] improves the above problem based on new and old link tables. However, they do not consider the data structure of big data, which is key-value pairs, which is different from caching the whole page.

3 Cache management framework of big data

In the paper we propose a Distributed Cache Management Framework of big data (DCMF) to accelerate processing speed of big data. The proposed framework includes three parts: cache structure, cache replacement algorithm and cache placement algorithm. Cache structure is the basis of cache replacement algorithm and cache placement algorithm, and it provides the organization format for unit of cache. Cache replacement algorithm selects the cached data, and cache placement algorithm puts the cache data into the cache of cloud nodes.

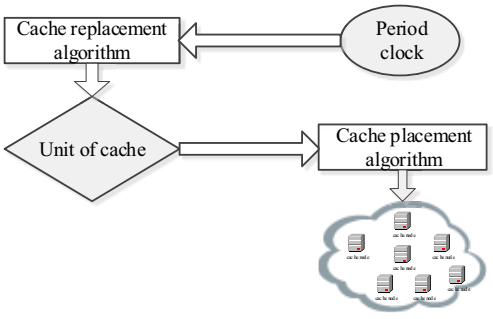

Figure 1. The flow chart of updating data cache

The proposed framework periodically updates big data cache. It first uses cache replacement algorithm to select the cached data from disk, then cache placement algorithm put the cache data on cloud nodes. The flow chart is shown in Fig. 1.

4 Distributed caching method of big data processing

In the section, we introduce the three part of proposed distributed cache management framework: cache structure, cache replacement algorithm and cache placement algorithm.

4.1 Cache structure

In disk, several stored key-value pairs constitute a unit of cache. Therefore, all the data in disk can be divided into certain unit of cache. Data are ordered by Rowkey. Assume the number of key-value pair in a unit of cache is k. We can set k to 100. As shown in Fig. 2, each dotted box is a unit of cache and has k key-value pairs. These units of cache are contiguously stored in disk.

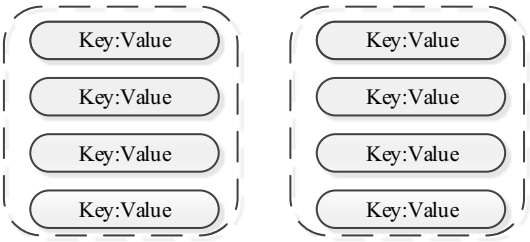

Figure 2. The structure of unit of cache

4.2 Cache replacement algorithm

Cache replacement algorithm [9] gives priority to the unit of cache with high value and accessing frequency and put them into caching set. Caching set is the chosen units of cache from all the units of cache and they will be put into cache on the nodes.

In the section we propose a Cache Replacement algorithm based on periodic Value (CRV). Its concrete steps are as follows.

1)Units of cache in caching set are updated each period.

2)For a period j, we record the accessing frequency of each unit of cache. For a unit of cache c_i, when a data access needs the data in c_i, its number of accesses in period j is computed as $n_i^j = n_i^j + 1$, where n_i^j is initialized as 0.

3)We compute the value of each unit of cache in period j, which is denoted as p。For the value of c_i is computed as $p_i^j = \alpha \cdot p_i^{j-1} + (1-\alpha) \cdot n_i^j \cdot \beta$, where p_i^{j-1} is the value of c_i in last period, α is periodic impact factor, β is the data value factor of c_i and $0 < \alpha, \beta < 1$. When c_i is accessed, its return time is tight and the β value is high.

4)We sort all the c_i in the descending order of p_i^j. Assuming that all the cache capacity is *h* and the capacity for a unit of cache is *b*, the number of units of cache that can be put in the cache is computed as $l = h / b$. We select the first *l* units of cache into caching set.

During data access we divide the degree of urgency into real-time, normal and loose according to the requirement on return time. The above three types correspond to different β value and β is selected from 0.9, 0.5 and 0.2.

Once data access is recorded as $<\cup\{c_i\},\beta>$, where $\cup\{c_i\}$ is the units of cache involved into the data access. We calculate the accessed frequency and the degree of urgency for each unit of cache in a period according to data access records.

4.3 Cache placement algorithm

Cache placement algorithm [10] is responsible for putting units of cache in caching set on cloud nodes. In the section we propose a Cache Placement algorithm based on Clustering (CPC). Its concrete steps are as follows.

1)Assuming the number of cloud caching node is n and the caching capacity of each cloud node is m, the total caching capacity is $h = m \cdot n$. The caching mechanism in clouds is organized by Memcache [11].

2)We set a unit of cache as a point. If two units of cache are accessed by a data processing, we add an edge weighted by 1. The weight can be added many times, i.e., if there exists an edgy and there is another data processing, the weight is increased by another 1. All the units of cache in caching set form a connected graph.

3)We divide all the c_i in caching set into n clusters using clustering algorithm. One cluster is denoted as C and the data volume of C is denoted as d_C. In the paper we propose a clustering algorithm called equal-capacity clustering algorithm based on k-means [12]. The algorithm divides all units of cache into the number of nodes (n), and each node has the same data volume.

Next we introduce our proposed equal-capacity clustering algorithm. We put all the l units of cache on all the n cloud nodes with the same caching capacity. Each node has $e=l/n$ units of cache.

1)We compute a weight for each unit of cache. The weight of unit of cache j is computed as $f_j = \sum_{i=1}^{q} w_{ij}$, where w_{ij} is the weight between unit of cache i and unit of cache j.

2)We sort all the units of cache in non-ascending order of f.

3)We first randomly select n units of cache as initial centers.

4)For each unit of cache, we obtain its edge weights with each initial center. The unit of cache is put into the same cluster with the initial center that has the largest weight.

5)When each unit of cache is put into clusters, there appears a situation that the number of units of cache is larger than e. In this case, we choose the unit of cache that has the smallest edge weight to shift out of the cluster, and the removed unit of cache re-chooses another center.

6)The algorithm stops until all the units of cache find their centers.

The proposed CPC algorithm largely reduces network transmission cost during big data access, because data processing only works on one node or several neighboring nodes.

5 Performance evaluation

At present, there are 21 nodes on the big data platform of equipment condition evaluation. Except two name nodes, maximum cache capacity of the 19 data nodes is 76GB. The main applications are searching data in PMS, EMS, GIS, computing the correspondence between equipment ID in EMS and PMS, the Chargeability of surge protectors, calculate the reject rate and time of duration of each attributes. We choose 10 days as a period. The size of each unit of cache is set to 1MB. We use CRV algorithm to choose several units of cache and use CPC algorithm to put clusters on 19 nodes.

In order to validate the performance of CRV and CPC algorithm, we use Least Recently Used algorithm (LRU) and round-robin scheduling algorithm as comparison objects. The performance indexes are hit rate and query time. LRU algorithm is to select the least used unit of cache to replace each time, which is most used in replacement algorithm. It implementation mechanism is simple and it adapts to the high local data access. Round-robin scheduling algorithm is to put units of cache on

different nodes in turn, i.e., each time the placed node is computed as $i = (i + 1) \bmod n$, and we choose the ith servers. The algorithm is simple and easy to realize.

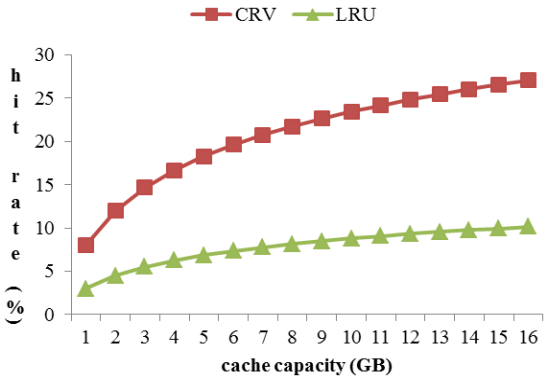

Figure 3. The comparison on hit rate of cache replacement algorithms

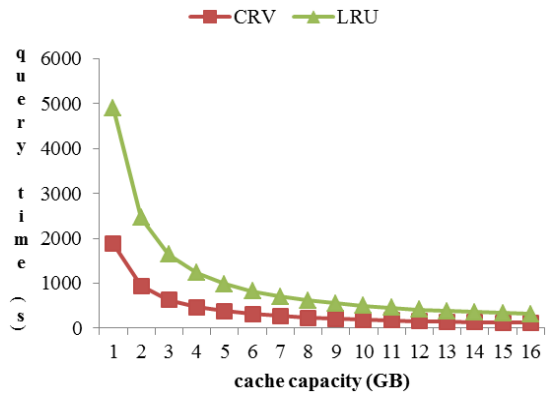

Figure 4. The comparison on query time of cache replacement algorithms

First we just use one node as cache placement node, and use LRU and CRV algorithm to run data accesses in a period. We record the hit rate and query time, as shown in Fig. 3 and Fig. 4. CRV has 13% higher hit rate than LRU algorithm. With the increase of cache capacity, the hit rate also ascends. The query time of CRV is on average 46.5% lower than LRU. Since there is only one cache placement node, the downtrend of query time gradually becomes slow with the increase of cache capacity.

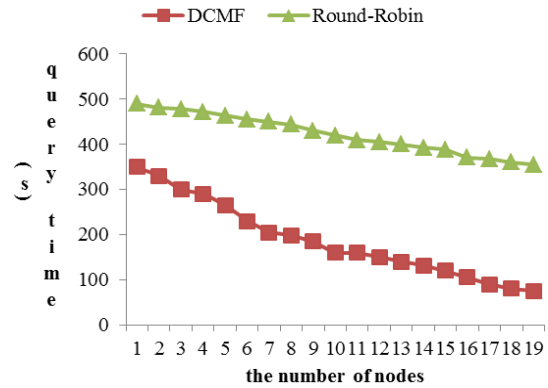

Figure 5. The comparison on query time of cache management frameworks

Second we validate the whole performance of proposed framework. In cache replacement phase we use our CRV algorithm and in cache placement phase we separately use CPC algorithm and round-robin algorithm. We run data access on multiple nodes in a period and record data query time, as shown in Fig. 5. The query time decrease with the increase of the number of caching nodes. Our proposed DCMF framework has 53.3% higher query efficiency than that of round-robin.

6 Conclusions

In the paper we design a distributed cache management Framework of big data for equipment condition assessment, which efficiently improves big data processing performance. Meanwhile, we propose a cache replacement algorithm based on value of units of cache in the last period and cache placement algorithm based on clustering method. All the approaches reduce network transmission cost and data processing time. Our extensive experiments demonstrate that the proposed CRV algorithm has 13% higher hit rate and 46.5% lower query time than LRU algorithm; the whole framework has 53.3% lower query time than round-robin algorithm.

Acknowledgement

This work was supported by the National High Technology Research and Development Program of China (863 Program) under grant no. 2015AA050204.

References

1. X.S. Peng, D.Y Deng, S.J Cheng, J.Y. Wen, Z.H. Li, L. Niu, *Proceedings of the CSEE*, **35**, 1, pp. 503-511 (2015).
2. X.L. Qin, W.B. Zhang, J. Wei, W. Wang, H. Zhong, T. Huang, Journal of Software, **24**,1, pp. 50-66 (2013).
3. Storm A J, Garcia-Arellano C, Lightstone S S, Diao Y, surendra M, *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*, Seoul, Korea, pp. 108-1092 (2006).
4. Malik T, Burns R, Chaudhary A, *Proceedings of the 35th International Conference on Data Engineering (ICDE)*, Tokyo, Japan, pp. 94-105 (2005).
5. Brown K P, Carey M J, Livny M, *Proceedings of the 19th International Conference on Very Large Data Bases (VLDB)*, Dublin, Ireland, pp. 328-341(1993).
6. Jincheng Yao, Shidong Zhang, Shiyu Liang, Liqing Zhong, Chinese Journal of Computers, **34**, 12, pp. 2319-2331(2011).
7. O'Neil Elizabeth J, O'Neil Patrick E, Weikum Gerhard, *Proceedings of the 19th International Conference on Management of Data (SIGMOD)*. Washington, DC, USA, pp. 297-306 (1993).
8. The InnoDB Buffer Pool. http://dev.mysql.com/doc/mysql-monitor/3.0/en/mem-graphs-innodb-buffer-pool.html. (2015-10.08).
9. Chaoyu Wang, The study on cach replacement strategy, harbin Engineering University (2012).
10. Wenzhong Li, Daoxu Chen, Sanglu Lu, Journal of Software, **21**, 7, pp. 1524-1535 (2010).
11. MemCache. http://www.danga.com/memcached, (2015-09-30).
12. Ng R.T., Han J, *Proc. 20th Int. Conf. on Very Large Data Bases*, pp. 144–155(1994).