

## RESEARCH ARTICLE

# Simulation Research on Fast Matching of Big Data Based on Spark

GUOJIAN XU, MINGYANG SONG, ZHENGGANG LENG, AND ZHENHONG JIA 

College of Information Science and Engineering, Xinjiang University, Ürümqi 830046, China

Corresponding author: Zhenhong Jia (jzhh@xju.edu.cn)

This work was supported in part by the Research and Development Foundation of China Mobile Communications Group Xinjiang Company Ltd., under Grant CMXJ-202001279.

**ABSTRACT** To solve the problem of low efficiency in real-time processing and matching of CNAME records in massive DNS log data, a parallel AC automaton enhancement method based on Spark was proposed. The method is based on the Spark distributed cluster computing engine of Hadoop, which ensures the stability of massive DNS log data storage with high fault tolerance and 24-hour real-time processing. At the same time, the Spark distributed cluster uses the multi-thread parallel computing method combined with the improved AC automaton algorithm, which not only reduces the memory occupied by trie construction, but also improves the efficiency of rapid matching of CNAME records of massive DNS logs. Simulation results show that the proposed method can quickly match CNAME records of massive DNS log data. Compared with the original AC algorithm, the efficiency is significantly improved, and the time complexity and storage space are reduced.


**INDEX TERMS** DNS, matching, CNAME, spark, parallel, distributed cluster.

## I. INTRODUCTION

Telecom operators need to dig out their core value through in-depth analysis of massive DNS [24] log data, and the alias record called CNAME in the log is often used for server IP address analysis and CDN (Content Delivery Network) [25] distribution. CNAME is very important to operators' business optimization, so it has become an important requirement for telecom operators to quickly extract and quickly match with the target domain name library to establish a dynamic CNAME library [1]. Currently, the mainstream frameworks for large-scale and high-throughput data processing include distributed intensive data processing platforms Apache Hadoop, HPCC (high performance computing cluster), Spark, a micro-batch processing model based on data slice collections, and Flink, an operator-based continuous flow model, etc. [2], [3], [4]. The Hadoop distributed computing framework mainly includes three parts: HDFS (Hadoop Distributed File System) [27], MapReduce [27] and YARN (Yet Another Resource Negotiator) [28]. The Spark computing framework with Spark Core as the core includes

local mode and cluster mode [3], [5], [6]. The distributed cluster parallel computing MapReduce in traditional Hadoop has the problem of excessive disk I/O overhead, which is not suitable for iterative calculation and it is easy to cause a single point of failure. However, Spark is much faster than MapReduce in performance, benefiting from its in-memory data processing. Moreover, as a general system, it can support batch, interactive, iterative, and streaming computations in the same runtime, which is useful for complex applications that have different computation modes [29], [30], [31], [32]. Currently common matching algorithms include single pattern matching and multi-pattern matching. Common single pattern matching algorithms include BF (Bruce Force) algorithm, KMP (Knuth Morris Pratt) algorithm, and suffix BM (Boyer-Moore) algorithm [7], [8], [9]. Multi-pattern matching algorithms include AC (Aho-Corasick automaton) algorithm [10], [11], and improved AC-BM (Aho-Corasick Boyer-Moore) algorithm based on AC algorithm [12], [13].

The core idea of the BF matching algorithm proposed in [7] is to match the text string and the pattern string character by character, and the efficiency is very low. The core idea of the KMP algorithm proposed in [8] is to construct the next array in the pattern string and jump through the next

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Marozzo .

array when the pattern string mismatches, but the text string and the pattern string have multiple identical characters that will be matched repeatedly, which will affect the efficiency. Literature [9] proposed a more efficient algorithm - BM algorithm, the core idea is to increase the pattern string jump distance by introducing good suffix rules and bad character rules. The AC automaton multi-pattern matching algorithm proposed in literature [10], [11] achieves multiple pattern string matching with one input by constructing a trie tree, which is more efficient. However, the trie construction of the AC algorithm wastes a lot of memory and requires multiple backtracking to transfer to the effective successor state when there is a mismatch, which affects the performance of the algorithm. Literature [12] and [13] is based on the AC algorithm, combined with the idea of BM algorithm, and adopts bad characters and good suffix rules to realize the jump matching of the pattern tree. The major disadvantage of this algorithm is that the good prefix rule preprocessing is complex and difficult to achieve.

Aiming at the characteristics of common pattern matching algorithms, this paper proposes a PPEACS (Parallel Python Enhanced Aho-Corasick automaton with highly available hadoop-based Spark) algorithm. The algorithm introduces an improved double-array trie method to enhance the original AC algorithm trie, which effectively reduces the space occupied by data storage, and combines the parallel computing method on the Spark distributed cluster based on Hadoop, which can analyze the CNAME domain names of massive DNS logs. Real-time processing and fast matching, better performance than other matching algorithms.

## II. SPARK DISTRIBUTED CLUSTER BASED ON HIGHLY AVAILABLE HADOOP

Since Hadoop is optimized for high data throughput and sacrifices the delay of obtaining data, it has no advantage for low-latency data access, but this problem can be effectively solved by introducing HBASE [33], [34] for massive data storage. In addition, compared with the Spark computing framework, Hadoop's computing engine MapReduce has disadvantages such as poor real-time computing, unsuitable for streaming computing and directed acyclic graph computing. Lastly, to solve the single point of failure problem of Hadoop and improve the stability of the 24-hour real-time computing of the cluster, the paper builds a highly available distributed cluster on the Linux server, as shown in TABLE 1. In Hadoop, because the NameNode manages the meta-data information of the entire HDFS file system, once a single NameNode stops working in the cluster, it will affect the real-time operation of the entire cluster [14], [15], making the cluster unable to process data in real time, and it is also not conducive to the cluster upgrade and maintenance, the NameNode is designed in a high-availability mode. Using the high-availability cluster design, even if a master node fails, the standby node can be switched to the working mode immediately through ZKFC (ZKFailoverController) [35], [36] process to perform failover and ensure the normal operation of the cluster. Similarly, the

TABLE 1. Cluster node configuration.

Server IP	10.235.31.241	10.235.31.242	10.235.31.243	10.235.31.244	10.235.31.245
Hostname	master01	master02	slave01	slave02	slave03
NameNode	1	1	0	0	0
JournalNode	1	1	1	1	1
DFSZKFailoverController	1	1	0	0	0
DataNode	1	1	1	1	1
ResourceManager	1	1	0	0	0
NodeManager	1	1	1	1	1
SparkQuorumPeerMain	1	0	0	0	0
HMaster	1	1	0	0	0
HRegionServer	0	0	1	1	1
PP	1	1	1	1	1
CPU	32	32	16	16	16
Memory	32G	32G	16G	16G	16G
Hard drive	2TB	2TB	1TB	1TB	1TB

TABLE 2. Software environment.

Software name	Version	Introduction
JDK	1.8.0_251	java runtime environment
Hadoop	3.1.3	distributed platform
Spark	3.0.3	distributed parallel computing platform
Python	3.6	python language depends on the environment
HBASE	2.4.11	distributed database
PP	1.6.4	parallel computing
Zookeeper	3.7.1	cluster coordination

ResourceManager, which manages cluster resources, is also designed for high availability in this paper. By simulating killing the NameNode of the background node master01, the NameNode of the node master02 immediately changes from standby to active mode, and the cluster still runs normally in real time. The software environment is shown in TABLE 2.

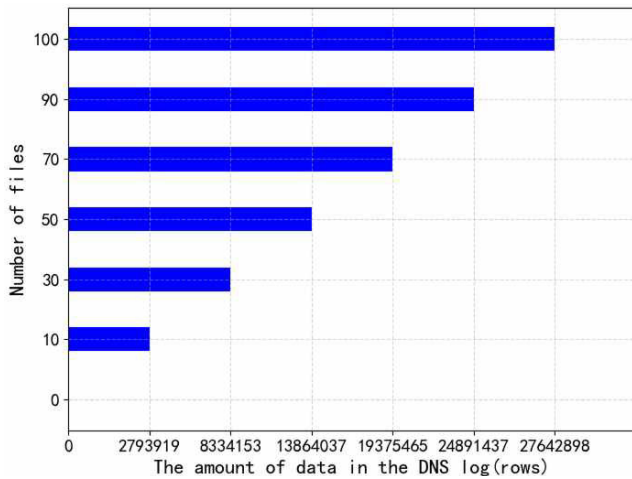


FIGURE 1. Changes in DNS log file data volume.

Compared to the other Hadoop distributed cluster, the highly available hadoop-based distributed Spark cluster computing design not only avoids the repeated disk I/O overhead and cluster single point of failure caused by MapReduce computing in the Hadoop framework, but also ensures that the cluster can continuously process large-scale clusters in real time and access data with low latency. Additionally, this cluster adds a parallel computing module to enhance computing power. For batch processing of massive DNS log data, it can perform high fault-tolerant storage and fast iterative calculation based on memory.

#### A. A. API SELECTION FOR SPARK TO CREATE RDD

RDD (Resilient Distributed Datasets) operations refer to functions that act on RDDs to generate new RDDs. These operations can be divided into transfer operations and action operations [16]. The transfer operation here is a lazy operation, which means that the operation between two RDDs will not be executed immediately, till wait for the action operation to be triggered. Spark supports multiple APIs (Application Programming Interface) to create RDDs, such as `sc.parallelize` (local collection, number of partitions), `sc.makeRDD` (local collection, number of partitions), `sc.textFile` (local/HDFS file/folder, number of partitions) and `sc.wholeTextFiles` (local/HDFS folder, number of partitions), etc. During this experiment, our data source is an external file. FIGURE 1 shows the data volume change of the DNS log file, and FIGURE 2 compares the performance of two APIs to create RDD. Obviously, in the API for creating RDD, reading `wholeTextFiles` for a large number of small files has better performance than `textFile` and consumes less time.

#### B. RDD OPERATOR OPTIMIZATION AND PERSISTENCE

Although operators such as `mapPartitions` are used in Spark to replace ordinary Maps, one function call will process all the data of a partition instead of one, which is more efficient.

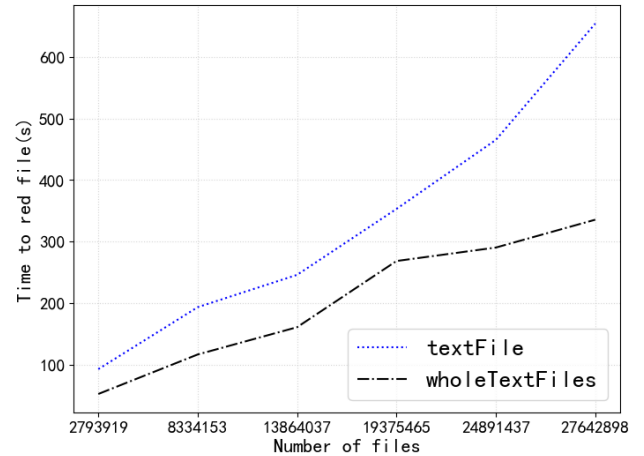


FIGURE 2. Performance comparison of two APIs for creating RDD.

However, when the amount of processed data is very large, once the memory is insufficient, it will cause OOM, that is, memory overflow, and the job will fail. For this reason, the computing performance of the `mapPartitions` operator can be fully improved by filtering and repartitioning data. Pre-aggregate on the map side before aggregation and use the `reduceByKey/aggregateByKey` operator instead of the `groupByKey` operator, which can reduce the aggregation memory usage and network transmission data volume in the reduce phase. However, this is easy to cause data skew. This paper greatly alleviates data skew by adopting local aggregation plus global aggregation method. Its specific method steps are in the first aggregation, first assign a random number to each key-value key, then perform aggregation operations such as `reduceByKey` and de-key prefixing on the data with random numbers, and finally perform the full aggregation operation again.

In Spark, RDD adopts a lazy evaluation mechanism, and every time an action operation is encountered, the calculation will be performed from scratch [17], [18]. This means that each call to an action triggers a calculation from scratch. This is very expensive for iterative calculations, which often need to reuse the same set of data. Using a persistence strategy for RDD can improve the data reuse rate. Using a single persistence level cannot balance memory and CPU. This paper implements differentiated persistence levels for different RDD operators, which effectively improves the performance of Spark jobs.

#### C. SPARK PARAMETER SELECTION

Spark parameters can be set by the user to control the job behavior during the execution of the Spark job. These parameters can change the amount of data in the middle of the Spark job, change the number of data transfers between memory and disk, and change the parallelism of each task in the Spark job [19], [20]. Different parameter values have different impacts on the execution performance of Spark

jobs. According to two principles: the main parameters affect the running performance of the task job and the resource scheduling in the process of the task job [37], [38], [39]. Therefore, it is very important to calculate and select some necessary parameters of Spark through cluster resources. The number of cores of nodes slave01~slave03 is 16. In order to ensure the normal operation of other services, spark.executor.cores is set to 5, just enough to fully use the cores of nodes slave01~slave03. According to the calculation, the total number of spark.executors.instances in the cluster is 21. spark.executor can be calculated by combining the calculation formula (spark.executor.memory + spark.executor.memoryOverhead) = yarn.nodemanager.resource.memory-mb \*(spark.executor.cores/yarn.nodemanagher.resource.cpu-vcores) which is 2G. To make full use of cluster resources, this experiment chooses to dynamically allocate the number of executors. spark.default.parallelism defaults to 2 to 3 times the total number of cores, and the maximum multiple is taken here. The detailed parameters are as follows.

```

spark.executor.cores 5
spark.executor.memory 2g
spark.executor.memoryOverhead 1g
spark.driver.memory 2g
spark.default.parallelism 315
spark.default.parallelism
spark.dynamicAllocation.enabled true
spark.shuffle.service.enabled true
spark.dynamicAllocation.initialExecutors 1
spark.dynamicAllocation.minExecutors 1
spark.dynamicAllocation.maxExecutors 21
spark.dynamicAllocation.executorIdleTimeout 60s
spark.dynamicAllocation.schedulerBacklogTimeout 15s
    
```

This article reasonably adjusts the above parameters to improve the efficiency and performance of Spark jobs, and the specific results are shown in FIGURE 3.

**III. PARALLEL PYTHON ENHANCED AHO-CORASICK AUTOMATON WITH HIGHLY AVAILABLE HADOOP-BASED SPARK**

This algorithm is based on the high-availability Hadoop distributed Spark cluster framework. On the basis of this framework, it includes two parts: the improved algorithm of AC automaton and the parallel computing method. This distributed Spark cluster can cope with high fault-tolerant storage and fast real-time calculation of massive data. In order to further improve the efficiency of fast matching, this paper introduces the idea of improved AC automata and parallel computing method.

**A. IMPROVED ALGORITHM OF ACT AUTOMATON**

The double-array trie can effectively reduce the storage space without affecting the query rate [21], [22]. Its core is the base array and the check array. Each element of the base

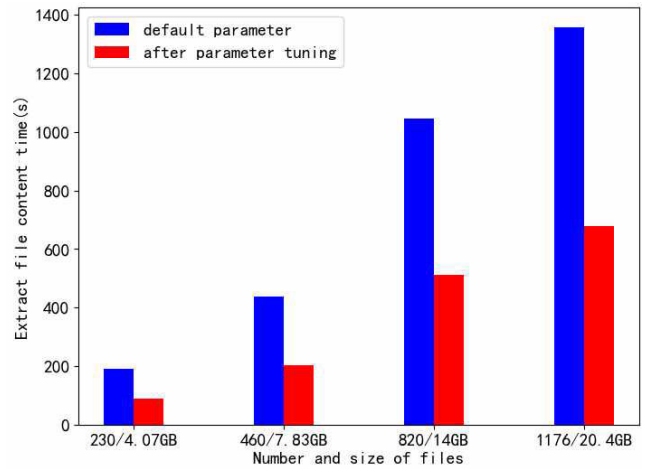


FIGURE 3. Comparison of Spark tuning parameters.

	1	2	3	4	5	6	7	8	9	10				
base	1	0	6	-17	-1	-13	2	0	-10	0				
check	9	0	1	7	7	7	3	0	3	0				
			Sb	Sbab	Sbac	Sbad	Sba		Sbc					
tail	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	h	e	i	o	r	#	\$		s	#	\$	g	e	
	15	16	17	18	19	20	21	22	23	24	25	26	27	28
	#	\$	y	#										

FIGURE 4. Improved double-array trie.

array represents a trie node, and each base array The element represents the predecessor state of a certain state, and the check array is used to check whether the state transition is correct. The trie can be represented by base and check arrays, and the following constraints are satisfied between them:

$$\text{base}[s] + c = t \tag{1}$$

$$\text{check}[t] = s \tag{2}$$

among them, s is the subscript of the current state, that is, the parent node, the default base [0] is 0, t is the subscript of the transition state, that is, the child node, and c is the code value of the input character.

On this basis, further improvement is introduced to compress the non-common prefix of the trie by the tail array. For example, double-array improvement is performed on the pattern character P = {"bachelor#", "bcs#", "badge#", "baby#"}, as shown in FIGURE 4, the negative value of base[s] means non-public Prefix, the absolute value indicates the starting position of the character suffix corresponding to the node in the tail array.

As shown in TABLE 3, compared with the memory occupied by the AC algorithm trie, the memory occupied by the improved double-array trie string is greatly reduced, which does not exceed 10% of the AC algorithm trie. At the same time, compared with the double-array trie algorithm, the memory usage of this algorithm also has an advantage, and the maximum decrease is about 2 percentage points.

TABLE 3. Comparison of string memory ratios of different algorithms.

string number	AC algorithm (Byte)	double -array trie (Byte)	Improved double-array trie (Byte)	double-array ratio of AC	improved double-array ratio of AC
10	15,552	632	492	4.06%	3.16%
30	32,400	1216	964	3.75%	2.97%
60	53,352	1992	1412	3.73%	2.64%
80	70,416	2624	1536	3.72%	2.18%
140	102,384	3808	1824	3.71%	1.78%

TABLE 4. Construction of enhanced AC automata.

Combining AC algorithm and improving the matching process of double-Array trie
1) traverse the input string (c is the character, sn is the state, n starts from 0, s0=0);
2) first jump according to goto:s(n+1)=base[sn] + c + 1, if check[s(n+1)]=base[sn], success, return s(n+1), failure, if s0=0? true, return 0, otherwise, continue to jump according to the fail pointer;
3) jump according to the fail pointer,
s(n+1)=base[fail[sn]] + c + 1,
if check[s(n+1)]=base[sn], success, return s(n+1); failure, if s0=0? true, return 0, otherwise, continue to jump according to fail;
4) according to the above return value, verify the output table, if output[s(n+1)] != null, represents that the match is successful, and all matched pattern strings are returned;
5) continue processing with the next character.

The AC algorithm combined with the process of improving the double-array trie is shown in TABLE 4.

B. PARALLEL COMPUTING METHOD

In the face of massive data in DNS logs, it is extremely time-consuming to simply use AC automaton matching to establish a CNAME domain name set. The original AC automaton is serial processing, and in the environment between multi-core CPUs and clusters, it cannot make full use of server resources for fast search or matching. For this reason, on the Spark distributed cluster, the enhanced AC automata is combined with the parallel Python [23] calculation to process large data in parallel, which can make full use of the multi-core CPU of the cluster to improve the computing performance. Assuming that the domain name data in the DNS log is a set T, it can be quickly matched with the target domain name data set P by using parallel multi-threading technology on the Spark distributed cluster, combined with the enhanced

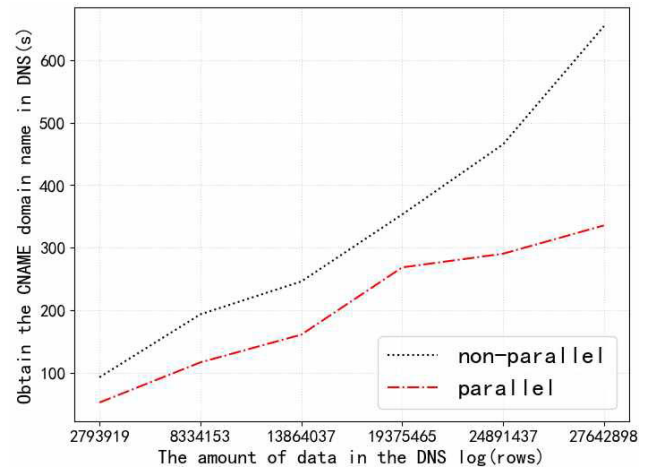


FIGURE 5. Comparison before and after parallel processing of DNS log data.

automaton algorithm. The specific steps are summarized as follows: 1) Segment the DNS log domain name T into s subsets, namely

$T = \{t_1, t_2, t_3, \dots, t_s\}$  ( $i = 1, 2, \dots, s$ ), and the subset  $t_i = \{tstr_{i1}, tstr_{i2}, \dots, tstr_{ij}, \dots, tstr_{in}\}$  ( $i = 1, 2, \dots, s, j = 1, 2, \dots, n, tstr_{ij}$  means the  $i$ th subset, the  $j$ th domain name), the target domain name set  $P = \{pstr_1, pstr_2, \dots, pstr_k\}$  ( $k = 1, 2, \dots, m, k$  representing the  $k$ th domain name); 2) Create a cluster All work sections, and start the parallelized Python cluster; 3) Parallelize s subsets and assign them to the working nodes of the cluster; 4) Submit job tasks for calculation. It can be seen from FIGURE 5 that using parallel processing methods in Spark computing can improve the efficiency of big data processing and shorten the running time.

C. ALGORITHM FLOWCHART IMPLEMENTATION

As shown in FIGURE 6, the figure is a detailed flow chart of the processing of massive DNS logs based on the Spark-based parallel enhanced AC automaton algorithm.

IV. SIMULATION RESULTS AND ANALYSIS

The data source of this simulation experiment is the DNS log data provided by the mobile telecom operator; the file format is a compressed package of .gz; the processed DNS data set is about 0.1TB; the domain name data volume in the target domain name database is selected twice. The experimental environment is the high-availability Hadoop-based Spark cluster platform built by the operator's laboratory. Increased multiple is defined as the time ratio before and after in TABLE 6 when the number of CNAME increases is the same.

It can be seen from TABLE 5 that among many matching algorithms, the calculation time of the PPEACS algorithm is the least. Compared with the single mode matching algorithm, the calculation speed of a single CNAME domain name matching is dozens of times; compared with the multi-pattern matching algorithm AC algorithm and the

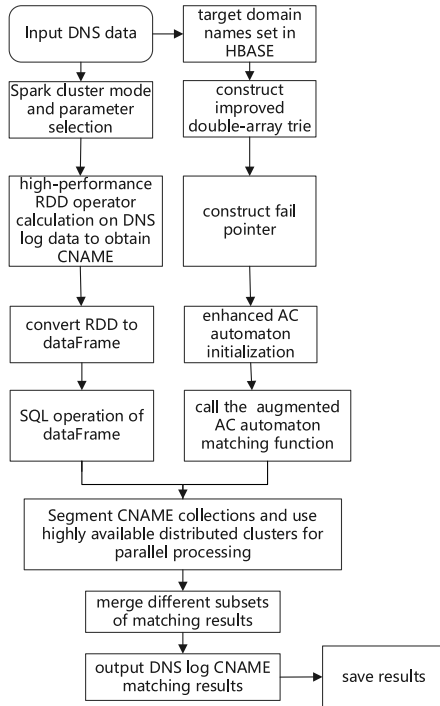


FIGURE 6. Flowchart of PPEACS algorithm.

TABLE 5. Comparison of matching performance of different algorithms.

algorithm	CNAME domain names	target domain names	mode	running time/s
BF	37,680	4,005	single	3506.1
KMP	37,680	4,005	single	3265.3
BM	37,680	4,005	single	2659.6
AC	37,680	4,005	single	318.2
AC-BM	37,680	4,005	single	89.5
PPEACS	37,680	4,005	single	39.4

improved AC-BM Compared with the algorithm, it is about 8 times that of the AC algorithm, and 2~3 times that of the improved AC-BM algorithm. Obviously, this algorithm has obvious advantages and higher efficiency. It can be seen from TABLE 6 that the performance of the algorithm is still optimal when the number of CNAMEs of DNS logs and the number of target domain names increase on a large scale, compared with other algorithms, the time spent by PPEACS has the lowest increased multiple, which shows that the algorithm is extremely advantageous and very applicable in the fast matching of big data. Compared with the original AC algorithm, which is also multi-mode, it needs to construct a large number of next arrays to store node information. This algorithm only needs to use double array numerical operations to represent node storage and conversion information. Compared with the AC algorithm, the storage space has been greatly improved.

TABLE 6. Comparison of performance of different algorithms in multi-mode.

CNAME domain names	target domain names	algorithm	running time/s	increased multiple
396,960	5,000	AC	3113.5	0
396,960	5,000	AC-BM	1038.9	0
396,960	5,000	PPEACS	415.6	0
2,793,919	5,000	AC	21575.4	6.9
2,793,919	5,000	AC-BM	7286.9	7.0
2,793,919	5,000	PPEACS	1456.6	3.5

TABLE 7. Comparison of whether PPEACS optimizes settings.

CNAME domain names	target domain names	Spark optimization settings	running time/s
37,680	5,000	no	125.4
37,680	5,000	yes	54.6
396,960	5,000	no	968.2
396,960	5,000	yes	415.6
2,793,919	5,000	no	6957.5
2,793,919	5,000	yes	1516.6

TABLE 8. Maximum memory usage of string matching in number of different CNAME domain names.

CNAME domain names	AC/KB	AC-BM/KB	PPEACS /KB
1,000	138,876	102,368	9,724
5,000	290,756	110,576	10,336
10,000	478,088	120,812	11,164

From TABLE 7, we can know that optimization settings such as RDD optimization and parameter optimization during Spark calculation and analysis will be 2 to 3 times more efficient than running without optimization settings about Computing performance for massive DNS log data on Spark distributed clusters. It can be seen from TABLE 8 that, with the increase of the number of CNAMEs compared with the AC algorithm and the AC-BM algorithm, the PPEACS occupies the lowest memory and the advantage of this algorithm in terms of memory usage becomes more and more obvious.

## V. CONCLUSION

The main purpose of this study is to solve the problem of low real-time processing and matching efficiency of alias records in massive DNS log data. To ensure the real-time and stable operation of the cluster, a cluster high availability method is used. Also, to improve the efficiency of big data processing, a distributed cluster with Spark as the computing engine is used to effectively process big data through reasonable

resource scheduling. Aiming at the low matching efficiency of existing algorithms for massive data, the Spark-based parallel enhanced AC automaton method proposed in this paper first uses the high-availability Hadoop Spark distributed cluster design to ensure high fault-tolerant storage and real-time processing of massive DNS data by the cluster. Secondly, it matches CNAME data in large-scale DNS logs. Compared with the original AC algorithm, the execution efficiency is increased by about 8 times, which is about 3 times that of the AC-BM algorithm. Compared with other matching algorithms, it has higher performance. Finally, on the basis of the AC algorithm, the trie tree is improved and enhanced to reduce the occupied space of the memory. The method has been applied in the actual production business of the mobile company, and it has improved the efficiency for the company to quickly match and obtain the target CNAME from the massive DNS logs.

Similarly, it can also be applied to keyword filtering, intrusion detection, virus detection, word segmentation and other scenarios, which is of great significance.

## REFERENCES

- [1] M. Liao, M. Chen, J. Zhou, X. Xiang, F. Li, and Y. Jiao, "DNS log analysis system based on big data fusion algorithm," *Telecommun. Sci.*, vol. 35, no. 5, pp. 129–139, 2019.
- [2] N. Verma, D. Malhotra, and J. Singh, "Big data analytics for retail industry using MapReduce-Apriori framework," *J. Manag. Anal.*, vol. 7, no. 3, pp. 424–442, Jul. 2020.
- [3] A. Mostafaeipour, A. J. Rafsanjani, M. Ahmadi, and J. A. Dhanraj, "Investigating the performance of Hadoop and Spark platforms on machine learning algorithms," *J. Supercomput.*, vol. 77, no. 2, pp. 1273–1300, Feb. 2021.
- [4] E. Nazari, M. H. Shahriari, and H. Tabesh, "BigData analysis in healthcare: Apache Hadoop, Apache Spark and Apache flink," *Frontiers Health Informat.*, vol. 8, no. 1, p. 14, Jul. 2019.
- [5] S. R. M. Zeebaree, H. M. Shukur, L. M. Haji, R. R. Zebari, K. Jacksi, and S. M. Abas, "Characteristics and analysis of Hadoop distributed systems," *Technol. Rep. Kansai Univ.*, vol. 62, no. 4, pp. 1555–1564, 2020.
- [6] H. Kadhodaie, A. M. E. Moghadam, and M. Dehghan, "Big data classification using heterogeneous ensemble classifiers in Apache Spark based on MapReduce paradigm," *Exp. Syst. Appl.*, vol. 183, Nov. 2021, Art. no. 115369.
- [7] X. Wu and Z. Wen, "Improvement of BF pattern matching algorithm," *Comput. Meas. Control*, vol. 26, no. 5, pp. 173–176, 2018.
- [8] O. C. Abikoye, A. Abubakar, A. H. Dokoro, O. N. Akande, and A. A. Kayode, "A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm," *EURASIP J. Inf. Secur.*, vol. 2020, no. 1, pp. 1–14, Dec. 2020.
- [9] Y. Duan, H. Long, and Y. Q. Qu, "Application of improved BM algorithm in string approximate matching," *Proc. Comput. Sci.*, vol. 166, pp. 576–581, 2020.
- [10] D. Regeciova, D. Kolar, and M. Milkovic, "Pattern matching in YARA: Improved Aho-Corasick algorithm," *IEEE Access*, vol. 9, pp. 62857–62866, 2021.
- [11] M. Karimov, K. Tashev, and S. Rustamova, "Application of the Aho-Corasick algorithm to create a network intrusion detection system," in *Proc. Int. Conf. Inf. Sci. Commun. Technol. (ICISCT)*, Nov. 2020, pp. 1–5.
- [12] A. Khumaidi, Y. A. Ronisah, and H. P. Putro, "Comparison of Knuth Morris Pratt and Boyer Moore algorithms for a web-based dictionary of computer terms," *J. Informatika*, vol. 14, no. 1, p. 7, Jan. 2020.
- [13] L. Shuai and S. Li, "Performance optimization of snort based on DDPK and Hyperscan," *Proc. Comput. Sci.*, vol. 183, pp. 837–843, Jan. 2021.
- [14] P. Shen, X. Ding, and W. Ren, "Research on kerberos technology based on Hadoop cluster security," in *Proc. 2nd Int. Conf. Adv. Energy, Environ. Chem. Sci. (AEECS)*, 2018, pp. 238–243.
- [15] P. Kumar et al., "Analysis and comparative exploration of elastic search, MongoDB and Hadoop big data processing," in *Soft Computing: Theories and Applications: Proceedings of SoCTA 2016*, vol. 2. Singapore: Springer, 2018, pp. 605–615.
- [16] J. Tang, M. Xu, S. Fu, and K. Huang, "A scheduling optimization technique based on reuse in spark to defend against APT attack," *Tsinghua Sci. Technol.*, vol. 23, no. 5, pp. 550–560, 2018.
- [17] M. M. Khan, M. A. U. Alam, A. K. Nath, and W. Yu, "Exploration of memory hybridization for RDD caching in Spark," in *Proc. ACM SIGPLAN Int. Symp. Memory Manag.*, Jun. 2019, pp. 41–52.
- [18] K.-M. Lee, I. Kim, and K.-C. Lee, "DQN-based join order optimization by learning experiences of running queries on Spark SQL," in *Proc. Int. Conf. Data Mining Workshops (ICDMW)*, Nov. 2020, pp. 740–742.
- [19] S. Dolev, P. Florissi, E. Gudes, S. Sharma, and I. Singer, "A survey on geographically distributed big-data processing using MapReduce," *IEEE Trans. Big Data*, vol. 5, no. 1, pp. 60–80, Mar. 2019.
- [20] J. Lee, B. Kim, and J.-M. Chung, "Time estimation and resource minimization scheme for Apache Spark and Hadoop big data systems with failures," *IEEE Access*, vol. 7, pp. 9658–9666, 2019.
- [21] Y. Chen, S. Wushouer, and Q. Yu, "An improved multi-pattern matching algorithm based on Aho-Corasick algorithm," *Mod. Electron. Technol.*, vol. 42, no. 4, pp. 89–93, 2019.
- [22] L. Jia, C. Zhang, M. Li, Y. Chen, Y. Liu, and J. Ding, "An efficient two-level-partitioning-based double array and its parallelization," *Appl. Sci.*, vol. 10, no. 15, p. 5266, Jul. 2020.
- [23] M. Xinyuan, W. Tao, and M. Kaigang, "Application of Python parallel computing in online identification of Thevenin equivalent parameters," in *Proc. IEEE 3rd Student Conf. Electr. Mach. Syst. (SCEMS)*, Dec. 2020, pp. 20–23.
- [24] Z. Jia and Z. Han, "Research and analysis of user behavior fingerprint on security situational awareness based on DNS log," in *Proc. 6th Int. Conf. Behav., Econ. Socio-Cultural Comput. (BESC)*, Oct. 2019, pp. 1–4.
- [25] S. Cui, M. R. Asghar, and G. Russello, "Multi-CDN: Towards privacy in content delivery networks," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 5, pp. 984–999, Sep. 2020.
- [26] Y. Tian and X. Yu, "Trustworthiness study of HDFS data storage based on trustworthiness metrics and KMS encryption," in *Proc. IEEE Int. Conf. Power Electron., Comput. Appl. (ICPECA)*, Jan. 2021, pp. 962–966.
- [27] D. Chen, R. Zhang, and R. G. Qiu, "Noninvasive MapReduce performance tuning using multiple tuning methods on Hadoop," *IEEE Syst. J.*, vol. 15, no. 2, pp. 2906–2917, Jun. 2021.
- [28] Y. Yao, H. Gao, J. Wang, B. Sheng, and N. Mi, "New scheduling algorithms for improving performance and resource utilization in Hadoop YARN clusters," *IEEE Trans. Cloud Comput.*, vol. 9, no. 3, pp. 1158–1171, Jul. 2021.
- [29] S. Tang, B. He, C. Yu, Y. Li, and K. Li, "A survey on Spark ecosystem: Big data processing infrastructure, machine learning, and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 71–91, Jan. 2022.
- [30] M. M. Rathore, H. Son, A. Ahmad, A. Paul, and G. Jeon, "Real-time big data stream processing using GPU with Spark over Hadoop ecosystem," *Int. J. Parallel Program.*, vol. 46, no. 3, pp. 630–646, Jun. 2017.
- [31] M. Jiang, B. Gallagher, A. Chu, G. Abdulla, and T. Bender, "Exploiting Spark for HPC simulation data: Taming the ephemeral data explosion," in *Proc. Int. Conf. High Perform. Comput. Asia-Pacific Region*, Jan. 2020, pp. 150–160.
- [32] M. M. Rathore, A. Paul, A. Ahmad, M. Anisetti, and G. Jeon, "Hadoop-based intelligent care system (HICS): Analytical approach for big data in IoT," *ACM Trans. Internet Technol.*, vol. 18, no. 1, pp. 1–24, Feb. 2018.
- [33] H. Zhang and G. Rong-Li, "Distributed HBase cluster storage engine and database performance optimization," in *Proc. IEEE 23rd Int. Conf. High Perform. Comput. Commun., 7th Int. Conf. Data Sci. Syst., 19th Int. Conf. Smart City, 7th Int. Conf. Dependability Sensor, Cloud Big Data Syst. Appl. (HPCC/DSS/SmartCity/DependSys)*, Dec. 2021, pp. 2274–2277.
- [34] L. Zhang, J. Sun, S. Su, Q. Liu, and J. Liu, "Uncertainty modeling of object-oriented biomedical information in HBase," *IEEE Access*, vol. 8, pp. 51219–51229, 2020.
- [35] P. M. Dhulavvagol and S. G. Totad, "Performance enhancement of distributed system using HDFS federation and sharding," *Proc. Comput. Sci.*, vol. 218, pp. 2830–2841, 2023.
- [36] Z. Cai and Y. Ling, "Research on the resource shortage scheme based on high availability cluster," in *Proc. Int. Conf. Comput. Netw., Electron. Autom. (ICCNEA)*, Sep. 2019, pp. 71–74.

- [37] D. Chang, Z. Qiao, L. Li, and Q. Zheng, "Parameter optimization of Spark in heterogeneous environment based on hyperband," in *Proc. 2nd Int. Conf. Big Data Economy Inf. Manag. (BDEIM)*, Dec. 2021, pp. 204–208.
- [38] D. M. Adinew, Z. Shijie, and Y. Liao, "Spark performance optimization analysis in memory management with deploy mode in standalone cluster computing," in *Proc. IEEE 36th Int. Conf. Data Eng. (ICDE)*, Apr. 2020, pp. 2049–2053.
- [39] H. Chen, P. Chang, Z. Hu, H. Fu, and L. Yan, "A Spark-based ant lion algorithm for parameters optimization of random forest in credit classification," in *Proc. IEEE 3rd Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, Mar. 2019, pp. 992–996.



**ZHENGANG LENG** received the bachelor's degree in electronic information engineering from the School of Electrical Information Engineering, Panzhuhua University, China, in 2019. He is currently pursuing the master's degree with the School of Information Science and Engineering, Xinjiang University, China. His research interests include data mining and distributed storage.



**GUOJIAN XU** received the bachelor's degree in communication engineering from the School of Information Science and Engineering, Chongqing Jiaotong University, China, in 2015. He is currently pursuing the master's degree with the School of Information Science and Engineering, Xinjiang University, China. His research interests include big data analysis and processing.



**MINGYANG SONG** received the bachelor's degree from the School of Electronic and Electrical Engineering, Nanyang Technological University, China, in 2019. He is currently pursuing the master's degree with the School of Information Science and Engineering, Xinjiang University, China. His research interests include data mining and analysis.



**ZHENHONG JIA** received the B.S. degree from Beijing Normal University, Beijing, China, in 1987, and the M.S. and Ph.D. degrees from Shanghai Jiao Tong University, Shanghai, China, in 1987 and 1995, respectively. He is currently a Professor with the Key Laboratory of Signal Detection and Processing, Xinjiang University, China. His research interests include image processing and photoelectric information detection and sensors.

...