Search Engine Big Data Management and Computing

RagavanN[1] Athinarayanan S[2]
[1]M.Tech Student [2]Assistant Professor
[1,2]Department of Information Technology
[1,2]PSN College of Engineering & Technology, Melathediyoor, Tirunelveli, Tamil Nadu, India

Abstract— Web Search Engine System includes wide range of in-memory data management of both data storage systems as well as Dynamic data processing techniques. Growing main memory capacity has fueled the development of in-memory big data management and processing. By eliminating disk I/O bottleneck, it is now possible to support interactive data analytics. Data gathered from crawling resources are spilt into Text and Numeric data and link need to be provided based on Offset Indexing. Frequent access data need to be loaded into Main memory using cache technologies such as web-cache in Apache or IIS. Bulk data are stored in In-memory databases such as REDIS. Big Numeric data can be stored in Flat File Binary Format such that they can be loaded into RAM in a single read operation. As run time disc access in flat file binary files are faster than conventional files, big data are stored in terms of Binary format which reduces the storage space up to 35% than other file formats. Some issues such as fault-tolerance and consistency are also more challenging to handle in in-memory environment. A Comprehensive presentation of important technology and factors to achieve efficient in-memory data management and processing in Web Search Engine are also considered.

Key words: Search Engine, Big Data, Binary File, In-Memory Database, Main Memory

I. INTRODUCTION

A. Search Engine

A web search engine is a software system that is designed to search for information on the World Wide Web. The search results are generally presented in a line of results often referred to as search engine results pages (SERPs). The information may be a mix of web pages, images, and other types of files. The search engine then analyzes the contents of each page to determine how it should be indexed (for example, words can be extracted from the titles, page content, headings, or special fields called meta tags). The explosion of Big Data has prompted much research to develop systems to support ultra-low latency service.

B. Search Engine Big Data

In designing a file system for our needs, it is needed to make certain assumptions that offer both challenges and opportunities. Multi-GB files are the common case and should be managed efficiently. In the proposed system binary files contain headers, blocks of metadata used by a computer program to interpret the data in the file. The header often contains a signature or magic number which can identify the format. The header file which contains metadata is stored in memory, so that master operations are fast. Furthermore, it is easy and efficient for the master to periodically scan through its entire state in the background. This periodic scanning is used to implement dynamic loading, free-up the files from memory. The master file size is small and it often contains the detail file offset details. Detail files on the other hand grow up to any limit. It can reach up to Terra size. Since detail files are lookup by means of hop method, it is not fully loaded into the memory. This requires offset data (gathered from master file) that are dynamically retrieved in a single read. Thus in-memory data storage systems, and in-memory data processing systems, including in-memory batch processing and real-time stream processing flat files are essential parts of web search engine. One of the best known search engine file system is Google's files system[13] commonly called as GFS.GFS is enhanced for Google's core data storage and usage needs (primarily the search engine), which can generate enormous amounts of data that needs to be retained.

C. In- Memory Database

"Memory is the new disk, disk is the new tape" is becoming today [1]. In the last decade, multi-core processors and the availability of large amounts of main memory at plummeting cost are creating new breakthroughs, making it viable to build in-memory systems where a significant part, if not the entirety, of the database fits in memory. Modern high-end servers usually have multiple sockets, each of which can have tens or hundreds of gigabytes of DRAM, and tens of cores, and in total, a server may have several terabytes of DRAM and hundreds of cores. Moreover, in a distributed environment, it is possible to aggregate the memories from a large number of server nodes to the extent that the aggregated memory is able to keep all the data for a variety of large-scale applications (e.g., Facebook [2]).

In general, research in-memory data management and processing focus on the following several aspects for efficiency. The main key factors considering the efficiency are Indexes, Data layouts and Data overflow.

1) Indexes:

Although in-memory data access is extremely fast compared to disk access, an efficient index is still required for supporting point queries in order to avoid memory-intensive scan. Hash-based indexes are commonly used in key-value stores, e.g., Memcached [3], Redis [4], RAMCloud[5], and can be further optimized for better cache utilization by reducing pointer chasing [6].

2) Data layouts:

Columnar layout of relational table facilitates scan-like queries/analytics as it can achieve good cache locality [7], [8], and can achieve better data compression. In addition, there are also proposals on handling the memory fragmentation problem, such as the slab-based allocator in Memcached [3]. Cache-conscious design such as columnar structure and data de-fragmentation are the main focuses.

3) Data overflow:

With the advancement of hardware, hybrid systems which incorporate non-volatile memories (NVMs) (e.g., SCM, PCM, SSD, Flash memory) [9], [10], [11]become a natural solution for achieving the speed. Alternatively as in the

traditional database systems, effective eviction mechanism could be adopted to replace the in-memory data when the main memory is not sufficient.

II. SYSTEM IMPLEMENTATION

The proposed system consists of three services namely Crawl Batch Service, In-Memory Service and Keyword Search Service and are represented in Fig 2.1

A. Crawl Batch Service

Crawl data are parsed using Content Parsing algorithm and keywords (user search criteria) are generated and auto ids are assigned to each keyword. These keywords are stored in RDBMS. The Crawled Web page URLs are maintained in No SQL distributed database. The relationship between Web Page and its parsed keywords are stored as big data flat binary file format since the run time disc access in flat file binary files are faster than conventional files, also Binary format which reduces the storage space up to 35% than other file formats. Fig.2.2 shows the Batch Service Architecture. In our system, MONGO DB is used for storing Web Page URLs.

B. In-Memory Middleware Service

Newly generated keywords in RDBMS are and loaded into In-Memory database with key-value hash structure in a distributed architecture fashion periodically. In our proposed system MYSQ is used as RDBMS which holds the Keyword details. As billons of single keywords and millions of complex keywords are needed to be stored, we have maintained 26 keywords table as per alphabetical order. For example, if the newly arrived single keyword is "PSN", and complex-keyword is "PSN College of Engineering", then these keywords will go to the P- keyword table. The middleware service is implemented in PHP using In-Memory Database client "REDIS". The REDIS server accepts request from REDIS client by listening at a specified port. The REDIS client will look up into MYSQL keywords table and loaded into the In-Memory database (REDIS) in a specified frequent interval of time. Fig.2.3 shows the architecture of Middleware Service.

C. Keyword Search Service

Search service is responsible for getting request from user and Lookup binary files and do search operation on that and respond the result. Search service should be running continuously, so that it can be added into the system crontab utility. In case of any segmentation issues occur, it will restart automatically. The layout of the search service is shown in the following fig .3. Search service should load all the master bin files created by the batch service at its first step. Then it should listen at a specified port number to accept request from the user. Once the request arrives at the port, it will parse the request parameters. By taking the one of the keyword parameter, it forms a request to middle ware service to have the corresponding key id. Then by using that keyword id, checking will be done against the loaded master bin data. If the keyword is available in memory, then its corresponding page offset is fetched from memory. Then by using the page offset, the required detail binary file will need to be accessed. Hopping into the detail binary file to go the required position and from where the keyword's page indexes are collected. The page ids for the collected page indexed will also be generated and then the resultant page ids are sent as response to the user. If no matched keyword id exists when looking up the loaded master binary file, empty response will be sent through. Fig 2.4 shows the Keyword Search Service.

The first step in Batch service is parsing the category wise crawled page detail from the crawled source. The page master level entries are then saved into database and the remaining numeric type page details such as page_id, keyword_ids are loaded into the map data structure in C. Then three types of binary files are created. They are

- Index File
- Detail File
- Index-Data File

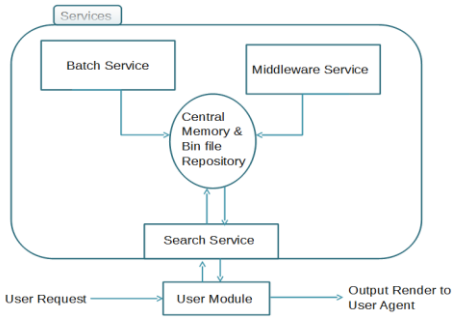

Fig. 2.1: Web Search System Architecture

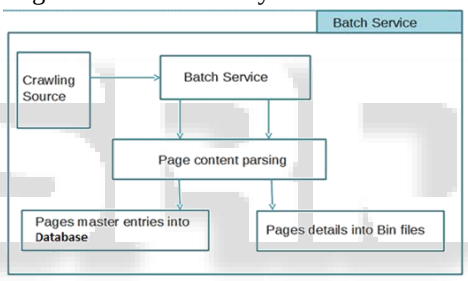

Fig. 2.2: Batch Service

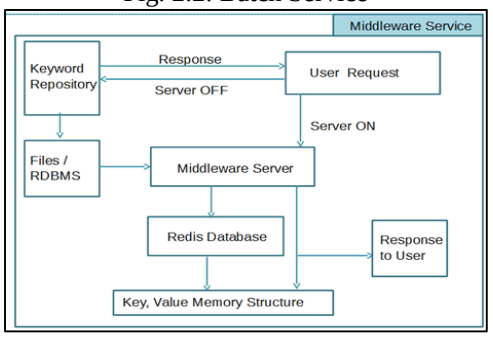

Fig. 2.3: Middleware In-Memory Service

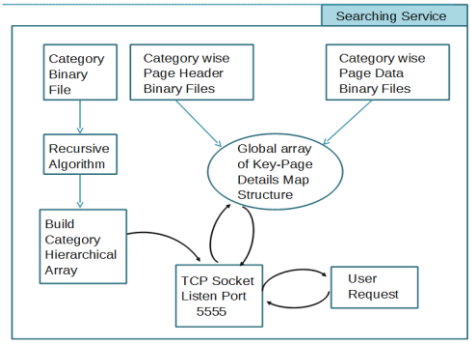

Fig 2.4 Search Service

Index file contains the header information such as total pages, total keywords and keywords offset entries.

Detail files contains the page level details such as how many images, videos belongs to this page and also the page status and created date entries. The above two files sizes are in nominal range. The third file Index-Data file is the big one. It contains key wise page alias entries. For example if a category contains about 1million keywords and each keyword belong to 1million pages, then totally 1 million x 1 million = 100 billion entries are available. If the page alias are saved as integer data type i.e. 32 bit or 4 bytes, then the size of the Index-Data file is 100 billion x 4 bytes = 400 billion bytes which is around 372 GB

III. TECHNIQUES USED

Four types of techniques are adopted in searching binary files. They are

1) Flat file Offset index Technique.
2) Ranking Data Technique.

A. Flat File Offset Index Technique

To read the dataset multiple times, in case of big data files, hop read method is preferred. In this method, the entire big data bin files are not fully loaded into the memory and instead, the data to be read offsets are preloaded into the memory. By using the flat file offset, dynamically hopping will be implemented and results are fetched from the bin file in milli seconds. This feature is very useful while reading big data binary files.

B. Ranking Data Technique

In order to produce randomized results to user side, two techniques are used. In the first method, the needed page offsets are loaded into a map. As map by default sort the data by its key, smaller new page ids / key ids are brought forward automatically. By changing the category child from first one to any one will bring the chosen category page details into user side.

IV. RESULTS AND DISCUSSIONS

The first and foremost step to carry out the web search is to start up the Search Service after completion of Batch Service. Once the search service is getting ready and makes it to listen a specified port, Middleware service needs to be started. This service generates page binary files from RDBMS. The batch service code compilation using g++ compiler creation of binary files is shown in fig 4.1.

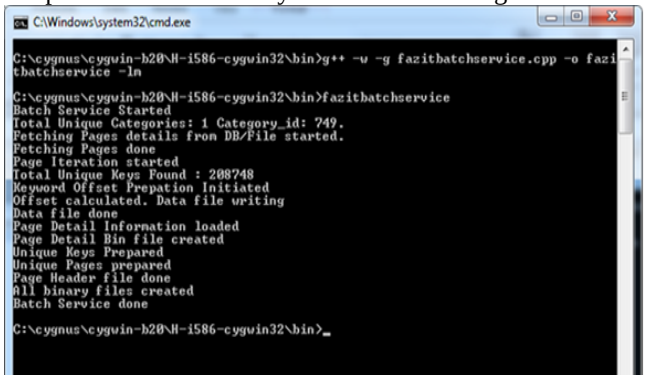

Fig. 4.1: Batch Service Result

In- Memory Redis Server started at port no: 6379 (shown in fig 4.2). Load Keyword details into In-memory Redis database from PHP.

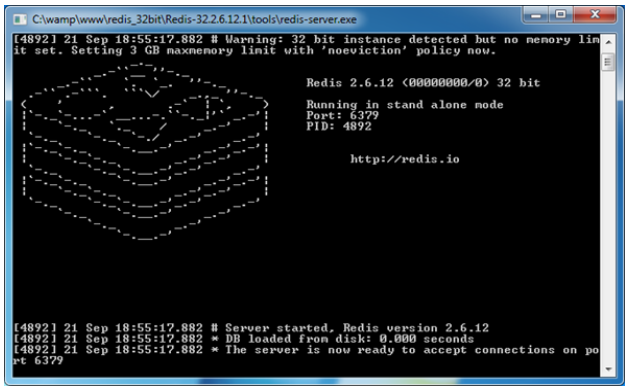

Fig. 4.2: REDISSES Server Console

The Page_Index, Page_Detail and Page_Index_Data binary files generated from Batch service are making it into In-memory map structure. Load the hierarchical category details from category bin file into memory pointer array. After loading page binary files and category binary file, search service will look up at port no: 5555 for fetching request from user side. Requests are then spilt out and needed binary file is looked up and iteration is performed in the binary file and page output details are provided as respond and send the result through the same port. The search service result is shown in fig 4.3.Searching service fetch the request using tcp and generated response and send back the result. Response is parsed and Category page counts are rendered in the right frame and page details are rendered in the left frame. The execution and output of Search Service is shown in fig 4.4.

Fig. 4.3: Search Service Load

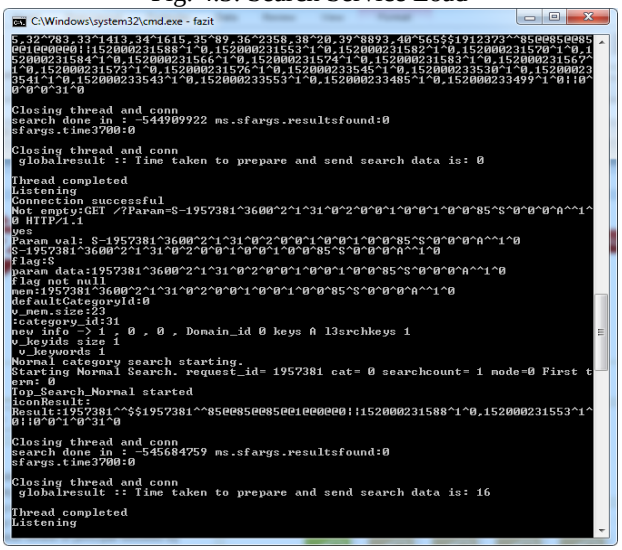

Fig. 4.4: Search Service Result

V. PERFORMANCE EVALUATION

A. Middleware Server Timing

Conventional DB lookup timing for 1 million records is around 20 milliseconds (average) In Redis In-Memory database, the memory lookup time: 3 to 7 millisecond. For mass amount of data, DB look is increasing geometrically (without indexing), whereas for Redis In-memory database it is constant independent of number of data. Thus DB logging bottleneck is eliminated in memory database systems. Speed efficiency increased to = 285 %. Fig 5.1. shows the DB lookup performance of conventional vs in-memory method.

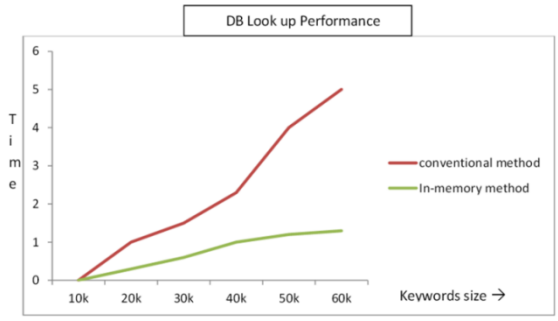

Fig. 5.1: DB lookup performance of conventional vs in-memory method

B. Search Server Timing:

Conventional searching data in Files or DB for average of 1 million records is 300 milli seconds. Searching data using preloaded offset method took around 100 milli-seconds. ** Speed efficiency = 300 %. It is shown in Fig 5.2.

C. Service Latency Time:

As Middleware and Search services are running continuously, initiating and invoking the service is too fast even in higher bandwidth. Initiating and invoking ordinary files is 50 milliseconds average which is nothing in running services. Service Latency efficiency will be increased to 50%

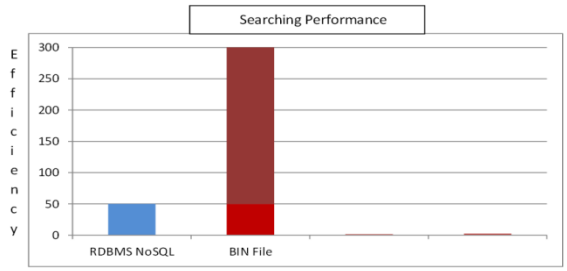

Fig. 5.2: Searching performance of Database vs BINARY file method

VI. CONCLUSION

We made our web search engine that handles wide range of in-memory data management as well as big data dynamic processing techniques. In-Memory database such as REDIS can be deployed in distributed environment also. Big Numeric data can be stored in Flat File Binary Format. So we can easily determine thefuture expansion of the proposed system size based on the number of keywords and web pages.

Another important feature of the big data files is that can be loaded into RAM in a single read operation. This increases the loading performance. Once the page master level entries are loaded into the memory structure, its relevant detail entries are fetched in hop method from the detail file. Its relevant offset positions maintained the master files are pre-loaded into the memory data structure. So the run time disc accesses in flat file binary files are faster than conventional files. Also big data are stored in terms of Binary format which reduces the storage space up to 35% than other file formats

In performance evaluation, we have confirmed that our system had worked successfully. From the performance evaluation, we evaluated the processing time when the number of keyword pages were varied according to the capacity of data. There, we saw that the processing time is in linear mode when the volume size of processed data was increased in all cases.

ACKNOWLEDGEMENTS

The researchers duly acknowledge the support provided by the Management and Principal of PSN College of Engineering and Technology-Tirunelveli by means of providing all the research facilities.

REFERENCES

[1] S. Robbins, “RAM is the new disk,” InfoQ News, Jun. 2008.
[2] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazi_eres, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, “The case for RAMClouds: Scalable high-performance storage entirely in dram,” ACM SIGOPS Operating Syst. Rev., vol. 43, pp. 92–105, 2010.
[3] B. Fitzpatrick and A. Vorobey. (2003). Memcached: A distributed memory object caching system [Online]. Available: http:// memcached.org/
[4] S. Sanfilippo and P. Noordhuis. (2009). Redis [Online]. Available: http://redis.io
[5] S. M. Rumble, A. Kejriwal, and J. Ousterhout, “Log-structured memory for dram-based storage,” in Proc. 12th USENIX Conf. File Storage Technol., 2014, pp. 1–16.
[6] B. Fan, D. G. Andersen, and M. Kaminsky, “MemC3: Compact and concurrent MemCache with dumber caching and smarter hashing,” in Proc. 10th USENIX Conf. Netw. Syst. Des. Implementation, 2013, pp. 371–384.
[7] H. Plattner, “A common database approach for OLTP and olap USING an in-memory column database,” in Proc. ACM SIGMOD Int. Conf. Manag. Data, 2009, pp. 1–2.
[8] M. Kaufmann and D. Kossmann, “Storing and processing temporal data in a main memory column store,” Proc. VLDB Endowment, vol. 6, pp. 1444–1449, 2013.
[9] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, “Fawn: A fast array of wimpy nodes,” in Proc. ACM SIGOPS 22 ndSymp. Operating Syst. Principles, 2009, pp. 1–14.
[10] H. Lim, B. Fan, D. G. Andersen, and M. Kaminsky, “Silt: A memory- efficient, high-performance key-value

store," in Proc. 23rd ACM Symp. Operating Syst. Principles, 2011, pp. 1–13.
[11] B. Debnath, S. Sengupta, and J. Li, "Skimpystash: Ram space skimpy key-value store on flash-based storage," in Proc. ACM SIGMOD Int. Conf. Manag. Data, 2011, pp. 25–36.
[12] MySQL AB. (1995). Mysql: The world's most popular open source database [Online]. Available: http://www.mysql.com/
[13] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in Proc. 19th ACM Symp. Operating Syst. Principles, 2003,pp. 29–43.

