I J C T A, 9(15), 2016, pp. 7511-7516

A Proposal for Enhancement of Elasticsearch by mitigating n-Gram Indexing

Urvi Thacker*, Manjusha Pandey and Siddharth S. Rautaray*****

ABSTRACT

Searching is one of the most important activity in the world of Internet. Whenever one looks for any information in the World-Wide Web (WWW), the very first activityperformed is searching. As the amount of data in World-Wide Web (WWW) is increasing at a very fast rate, it is becoming very difficult to derive useful information from it. It allows every ordinary user to publish data that can be retrieved by other users. To fetch data according to requirement requires time as the domain of searching is very vast. Elasticsearch is highly available and distributed search engine. It is index based search engine and works on the concept of Inverted Index, which allows very fast data insertion and retrieval i.e. in Near-Real-Time. Elasticsearch allows two types of Indexing – Ngram and non-Ngram. This paper gives a comparison of non-Ngram and Ngram indexing and prefers better method as is evident by the experiments and results presented in the paper.

Keywords: *Elasticsearch, Indexing, Restful*

1. INTRODUCTION

In this world of data, to get any information we need to be able to search and analyze huge data. As the data size is growing, it is becoming tougher and time consuming task to analyze it. For quite a long time we have been basically over-powered and equipped by the amount of data moving through and delivered by our systems. Existing innovation has concentrated on the most proficient method to store and structure distribution centres brimming with data. That's fine and good until we really need to settle on choices progressively educated by this data. Elasticsearch is a tool of big data search. It can scale to hundreds of servers and petabytes of data (structured/unstructured). It can do some other intelligent assignments, but at its core it is made for moving through text, returning text similar to a given query and/or statistical analyses of a collection of text [1].

Elasticsearch additionally permits to consolidate geolocation with full-text search, structured search, and analytics [2]. It can be combined to other big data tools like Hadoop, to provide real time searches for log data [3].

More particularly, elasticsearch is a standalone database server, written in Java that takes data in and stores it in a sophisticated format optimized for language based searches. Working with it is simple and advantageous as its fundamental convention is executed with HTTP/JSON (Java Script Object Notation)[4]. Elasticsearch is also easily scalable, supports clustering and leader election out of the box. The Elasticsearch server is easy to install, and the default configuration provided is adequate for a standalone use i.e. without any change. A node is a running instance of Elasticsearch. Two or more nodes combinedly form an Elasticsearch cluster. To set up an Elasticsearch cluster, the value that needs to be edited in the configuration

* Kalinga Institute of Industrial Technology, Bhubaneswar India, *Email: thackerawake@gmail.com*

** Kalinga Institute of Industrial Technology, Bhubaneswar India, *Email: manjushafcs@kiit.ac.in*

*** Kalinga Institute of Industrial Technilogy, Bhubaneswar India, *Email: siddharthfcs@kiit.ac.in*

file - elasticsearch.yml is the cluster name. However, a default value has been provided. Elasticsearch will then automatically discover nodes on the network and bind them to form a cluster [4].

Elasticsearch works on concept of Inverted Index, which allows very fast data insertion and retrieval i.e. in Near-Real-Time. It indexes all the unique words that occur in any document corresponding to the list of documents in which it appears. Whenever a search is fired, the word is looked upon in the index and according to Elasticsearch's default algorithm - Lucenes Practical Scoring Function results are fetched and then given to user.

Elasticsearch provides two types of indexing nGram and non-nGram. nGram solves the problem of Partial Matching. Partial matching means user should be able to type in partial query string like Logic and the search result should return all the objects which contain *Logic* as one of their property value.

Partial Matching can also be solved by Wildcard Queries. In non-nGram indexing when a document is indexed, several inverted index gets created. One for each field. Each inverted index contains the field name, corresponding value of the field in given document and a pointer to the document. But if we don't use standard analyser of non-nGram and use ngram then the field will have multiple values of that field in inverted index. The analyzer will break the field value in sequences if size n (nmin to nmax) and store it in inverted index.

2. PROPOSAL FOR ENHANCEMENT OF ELASTICSEARCH

In this paper, we propose use of non-nGram Indexing over nGram Indexing. From the analysis performed it has been observed that using nGram the CPU usage and memory usage is much greater that the non-nGram indexing and time of execution of search queries is almost similar. From an experiment performed earlier, it has been observed that if the index size increases the time of execution of search query increases. So, if nGram is used, a simple query would take more time to perform search than in case of non-nGram.

The below Figure 1 explains the main advantages of non-nGram indexing over nGram Indexing.

3. DATASET

The data set used has been taken from VMWare internal site. The data represents the structure of different entities which helps in network virtualization. For e.g. logical switch, logical port, logical router, logical router port, transport zone, logical firewall and many more.

A logical switch contains its details like date of creation, created by, Logical port id (id of the logical port it is connected to), resource type, address binding etc. Similarly, a logical ports contains its date of

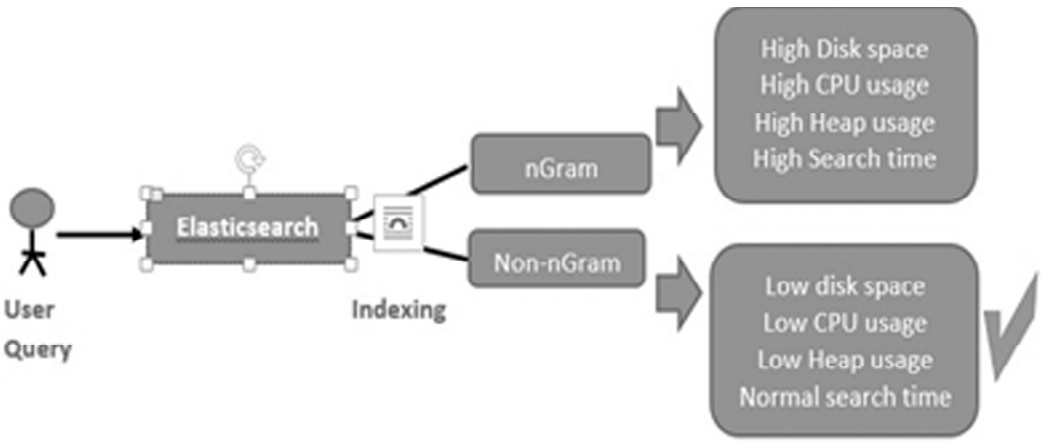


Figure 1: Advantages of non-nGram overnGram Indexing

creation, logical switch id (logical switch to which it is connected to), attachment type, switching profile ids etc. Similarly, logical ports, logical router ports, transport zones has been used in the dataset.

Sample Dataset:
Logical Router Port
{“result_count”: 1,
“results”: [{
“resource_type”: “LogicalRouterLinkPort”,
“revision”: 0,
“id”: “770ffb50-44f7-4d8c-a359-de0af1088932”,
“service_bindings”: [{
“service_id”: “dc5804f7-fdd9-4ef6-b5f8-80b1494ca183”, “resource_type”: “DhcpRelayService”
}]
“logical_router_id”: “b676dec7-5d78-4492-9b16-cb7cdcf65328”,
“_last_modified_user”: “admin”,
“_last_modified_time”: 1415746635953,
“_create_time”: 1415746635953,
“_create_user”: “admin”}]}
{
_index: “nsx23” _type: “tz”
_id: “AVFnxxr42Gux1asDnqTU” _score: 1
_source: { _revision: 1
display_name: “TZ117034”
_schema: “/v1/schema/TransportZone” transport_type: “OVERLAY” _last_modified_user: “admin” description: “”
tag: {
tags: “tenant” scope: “Tenant” }-
resource_type: “TransportZone” _last_modified_time: “1440169964305” transport_zone_profile_ids: { profile_id: 338369
resource_type: “BfdHealthMonitoringProfile” }-
uuid: 338369 host_switch_name: “a1” }-
}
Transportzone

4. COMPARITIVE ANALYSIS

The analysis was done considering the partial match requirement. Considering the two approaches - wildcard query and nGrams indexing, analysis on 200K objects was done. Below mentioned parameters were taken into consideration.

- CPU utilization of elasticsearch process during index creation
- Memory Utilization of elasticsearch process during index creation
- Index size
- Search Query Execution Time

Configuration: The analysis was carried out on an Ubuntu VM with 8 CPUs and 8GB memory (ip - 10.110.8.5) with elasticsearch (version used: 1.7.2 based on lucene version 4.10.4).

Tool used for profiling: VisualVM for CPU and memory profiling, pidstat for Disk I/O

JVM arguments

Elasticsearch was started with the following JVM arguments

-Xms256m
-Xmx1g -Djava.awt.headless=true
-XX:+UseParNewGC
-XX:+UseConcMarkSweepGC
-XX:CMSInitiatingOccupancyFraction = 75
-XX:+UseCMSInitiatingOccupancyOnly
-XX:+HeapDumpOnOutOfMemoryError
-XX:+DisableExplicitGC
-Dfile.encoding=UTF-8 –Delasticsearch
-Des.foreground=yes -Des.path.home=/home/ess/src/main/java/Document/elasticsearch-1.7.2

5. ANALYSIS AND OBSERVATION

5.1. Index Size

In Fig. 2, X-axis represent the number of objects (in thousands) and Y-axis represents the heap size in Mb. It is observed that the index size grows linearly with the number of objects if ngram is used.

From Figure 3, it can be observed that the CPU usage is very high when indexing is performed using ngram in comparision to non-ngram. Also, Heap size grows faster in case of ngram indexing. The disk size was observed to be 1.11 Gb (ngram indexing) i.e almost 10 times higher than 80.53 Mb (non-ngram indexing).

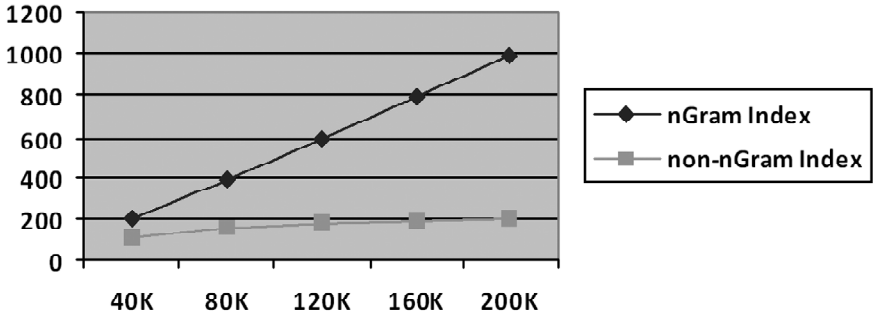
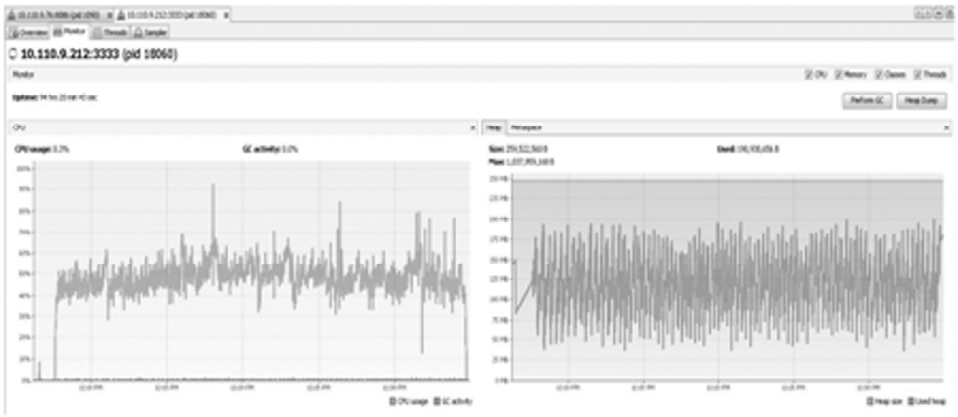
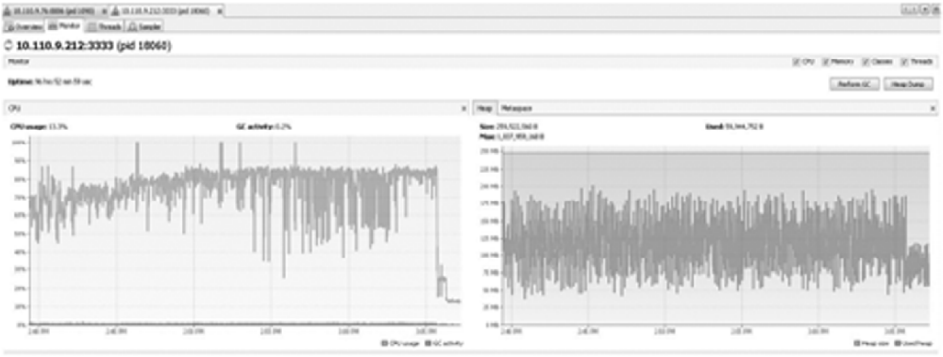


Figure 2: Comparison of Index size with nGram and non-nGram Indexing



(a) non-nGram indexing



(b) nGgram Indexing

Figure 3: CPU Performance and Heap Size

Table 1
Execution time of queries with nGram and non-nGram Indexing

Use Case	Query Type	Query	Execution Time in ms *1st Run*	*2nd Run*	*3rd Run*
Fetch all documents containing "Object222"	Wild Card	{ "query":{ "query_string": { "query":"Object222*" } }}	20	5	3
	Ngram index	{ "query":{ "query_string": { "query":"Object222" } }}	11	5	4
Fetch all documents containing "Description OR of OR Object20000"	Wild Card	{ "query":{ "query_string": { "query":"Description of Object20000*" } }}	16	12	9
	Ngram index	{ "query":{ "query_string": { "query":"Description of Object20000" } }}	194	18	13
Fetch all documents containing "Description AND of AND Object20000"	Wild Card	{ "query":{ "query_string": { "query":"Description AND of AND Object20000*" } }}	4	3	2
	Ngram index	{ "query":{ "query_string": { "query":"Description AND of AND Object20000" } }}	9	5	7

Table I gives the execution time of queries performing partial search in nGram and non-Gram indexing. As observed, the time variation is very less in two approaches but considering the above factors non-nGram seems to be a good deal.

6. CONCLUSION

From above results, it has been observed that Index size grows linearly with the number of objects if ngrams are used which is not observed if non-nGram is used.

Also, search query performance is almost equivalent in both the scenarios, though it was observed that in certain cases the search time for a particular string for the very first time was comparatively high when ngrams were used. As we can support similar queries using ngrams vs. non-ngrams index, we propose using non-nGram as the index size on disk is fare enough in comparison to nGram. As the search domain increases the time of search automatically increases. From earlier experiments performed, it has also been observed that if the query get complex the time of search increases reasonable even if non-nGram is used.

If the same query will be executed in nGram then time of execution is expected to increase many fold. So we propose non-Gram indexing with wild card queries to support partial matching.

REFERENCES

[1] Oleksii Kononenko, Olga Baysal, Reid Holmes, and Michael W Godfrey. Mining modern repositories with elasticsearch. In Proceedings of the 11th Working Conference on Mining Software Repositories, pages 328{331. ACM, 2014.

[2] Yicheng Zheng, Feng Deng, Qingmeng Zhu, and Yong Deng. Cloud storage and search for mass spatio-temporal data through proxmoxve and elasticsearch cluster. In Cloud Computing and Intelligence Systems (CCIS), 2014 IEEE 3rd International.

[3] Jun Bai. Feasibility analysis of big log data real time search based on hbase and elasticsearch. In Natural Computation (ICNC), 2013 Ninth International Conference on, pages 1166{1170. IEEE, 2013.

[4] Clinton Gormley and Zachary Tong. Elasticsearch: The Definitive Guide. “O’Reilly Media, Inc.”, 2015.