Transactions on Networks and Communications - Vol. 9, No. 2
Publication Date: April, 25, 2021
DOI: 10.14738/tnc.92.9887.
Fridge, E. L., & Bagui, S. S. (2021). Estimating Query Timings in Elasticsearch. Transactions on Networks and Communicaitons, 9(2). 15-36.

Estimating Query Timings in Elasticsearch

Evorell L. Fridge and Sikha S. Bagui
Department of Computer Science, University of West Florida
Pensacola, FL 32514 USA

ABSTRACT

In a shared Elasticsearch environment it can be useful to know how long a particular query will take to execute. This information can be used to enforce rate limiting or distribute requests equitably among multiple clusters. Elasticsearch uses multiple Lucene instances on multiple hosts as an underlying search engine implementation, but this abstraction makes it difficult to predict execution with previously known predictors such as the number of postings. This research investigates the ability of different pre-retrieval statistics, available through Elasticsearch, to accurately predict the execution time of queries on a typical Elasticsearch cluster. The number of terms in a query and the Total Term Frequency (TTF) from Elasticsearch's API are found to significantly predict execution time. Regression models are then built and compared to find the most accurate method for predicting query time.

Keywords: Elasticsearch, Elasticsearch Query, Query Cost Model, Document Frequency, Total Term Frequency, Term Vectors.

INTRODUCTION

Elasticsearch[1] is a highly scalable, full-text search engine intended to be used in a clustered environment. It was designed to be a distributed service that can scale to tens or hundreds of nodes if the situation calls for it. However, additional computing capacity comes with increased costs in computing resources and maintenance overhead. In order to match the capabilities of a cluster to the workload demanded of it, Elasticsearch administrators may perform load testing operations on their cluster to estimate its performance capabilities. Load testing allows administrators to examine how a cluster can perform in a worst-case situation (often called stress testing), or at an average expected load level. This allows these administrators to estimate whether or not their search cluster will meet the requirements of service level agreements (SLAs) that might be in effect with their users.

Load testing can only approximate the performance of a cluster in a given situation. It cannot fully simulate realistic user behavior, and although it gives administrators a rough idea of the performance of a cluster, it is not able to guarantee adherence to SLAs. Specific customers may exhibit widely fluctuating usage patterns or monopolize the search cluster, thereby affecting the performance available to the rest of the customers in a Search as a Service (SaaS) scenario. Search engine administrators may need to place an upper bound on search query load from a particular customer to enforce SLAs through the use of rate limiting techniques.

[1] https://www.elastic.co/elasticsearch/

Accurately estimating the execution time of a query is a crucial step in any rate limiting application. If each customer is given a budget of execution time, a given query must be evaluated to predict the amount of impact it will have on the cluster. Although Elasticsearch is fully capable of handling many queries per second, this sort of evaluation can be used to ensure a fair distribution of load or at least enforce a maximum amount of time that a query can wait before it is allowed to run on a cluster.

Query estimation can also be useful in exceptionally large search applications. Because Elasticsearch is a distributed system, the same query is distributed to all nodes in the system and executed in parallel. However, for very high user loads, it may be desirable to replicate the same index data among multiple Elasticsearch clusters. This would require a load balancing system to be placed in front of these clusters. If this system was able to estimate the impact of a query, query load could be intelligently distributed among all Elasticsearch clusters in order to level out the server load.

The distributed nature of Elasticsearch makes it difficult to gather statistics on typically known predictors of search engine performance such as the number of postings of all query terms. The novelty of this research lies in the usage of Elasticsearch's term vectors API to randomly sample a subset of documents in the index and approximate these predictors. Our research gathers these pre-retrieval statistics from an Elasticsearch index and investigates which of them are significant predictors of execution time of a query. This work also examines different regression models to identify which one can estimate query execution time with the lowest amount of error.

The rest of this paper is organized as follows. Section two presents the background for this work, including a discussion of the statistics available for query time prediction within Elasticsearch. Section three presents similar works from researchers who have investigated query time estimation, both in database and search engine applications. Sections four through six describe the process used to investigate predictors for Elasticsearch query timings and present the data collected. Section seven describes the use of predictors to build query cost models and the evaluation of the accuracy of those models. Finally, sections eight and nine contain the discussion of the results and conclusion.

BACKGROUND

Elasticsearch is a distributed search engine product that is based on the open-source Apache Lucene[2] search engine [1]. An Elasticsearch cluster uses multiple servers called nodes, and each node runs multiple instances of Lucene. Each Lucene instance is known as a shard, and Elasticsearch benefits from parallel execution of queries across multiple machines and shards simultaneously.

Although Elasticsearch is capable of performing many different kinds of queries and aggregations, Elasticsearch's match query is recommended for full-text search. It is "the standard query for performing a full-text search, including options for fuzzy matching" [2]. The default behavior of Elasticsearch's match query is to use the disjunctive "OR" when evaluating queries with multiple terms. Text in a query is analyzed (including edits for "fuzzy

[2] https://lucene.apache.org/

transpositions" [2] of characters), and then a query against the index is performed separately. The search results are obtained by applying the boolean OR operator to the matching documents for each term, and the resulting document list is presented as the search results [2].

The list of matching documents from an Elasticsearch query is ranked using the document-at-a-time approach [1]. In the document-at-a-time approach, "all matching documents are enumerated, one after another, and a score is computed for each of them. At the end all documents are sorted according to their score and the top k results… are returned to the user" [3]. Elasticsearch computes a page rank using the term frequency and inverse document frequency when searching on a particular field in a document [1]. The term frequency is the number of occurrences of a term in a field for a particular document, and the inverse document frequency is a measurement of how rare the term is relative to the number of documents in an index. The more documents contain a term, the less relevant it is. The length of a field is also taken into account, since a very short text fragment with that term such as a title is probably more important than a long field containing the contents of a book or article.

Execution time of a search query depends on the number of terms in a query as well as the number of documents in an index. Büttcher, Clarke and Cormack [3] state that the time complexity of the document-at-a-time search algorithm is $\theta(N_q \cdot n + N_q \cdot \log(N_q))$, where n is the number of query terms, and N_q is the "total number of postings of all query terms." Each occurrence of a term is stored in an inverted index, which allows the documents that contain the term to be easily retrieved. These document lists are known as postings, and they are usually stored in a compressed format to optimize disk and memory storage. Lucene's posting files are stored on disk in files called segments.

Another factor that can affect query execution time is data caching. At a low level, operating systems cache Lucene segments in RAM to reduce disk access costs. Search engines can also cache query results in RAM as long as the underlying data in a shard has not changed. Frequent or repeated queries can be handled quicker since the segments have already been loaded into RAM and are accessible without having to retrieve them again.

Elasticsearch provides a term vectors application programming interface (API) [4] that allows programmers to retrieve the underlying term frequency and document frequency values for a given collection of terms. These are the same values calculated by Elasticsearch at index time and used for performing a search query and sorting the results. This API provides the ability to analyze an existing document in an index, but it is also possible to submit an "artificial document" [4] to the API and obtain a list of term statistics for each word. Because the number of terms and postings is related to the amount of time a search takes [3], it may be possible to predict a search query's execution time by examining how these values vary in relation to the execution time of a query.

Two statistics available through the term vectors API that are directly related to an index's postings list are the total term frequency (TTF) and document frequency (DF). For each term in a query, the DF is the number of documents in the index that contain that term. The TTF is the sum of the number of times that term occurs in all documents in the index. When a query contains multiple terms, the values for each term can be summed to compute a value similar to the "total number of postings of all query terms," as mentioned in [3]. Because a postings list

contains an entry for all the occurrences of a term in each document, in this work a posting will be considered an individual entry for a document and therefore the DF should be a closer match.

Figure 1 presents an example of the results provided by Elasticsearch's term vectors API when searching with an artificial document containing the query "the wagon." Each term in the query is listed in the "terms" object along with the DF and TTF values. Note that the TTF value for each term is always equal to or greater than the DF since a term can occur one or more times per document.


```
{
    "_index": "client_index_2",
    "_type": "_doc",
    "_version": 0,
    "found": true,
    "took": 2,
    "term_vectors": {
        "text": {
            "terms": {
                "the": {
                    "doc_freq": 40796,
                    "ttf": 174277310,
                    "term_freq": 1
                },
                "wagon": {
                    "doc_freq": 7194,
                    "ttf": 45037,
                    "term_freq": 1
                }
            }
        }
    }
}
```


FIGURE 1. A result object from Elasticsearch's term vectors API.

Using the term vectors API on an Elasticsearch index with multiple shards results in a "relative measure" of term and document frequencies because "the information is only retrieved for the shard the requested document resides in" [4]. In indexes that require multiple shards to store a very large number of documents, the documents are evenly distributed among all the shards at index time. When using the artificial document approach, a shard is chosen at random to compute the term vector calculations. By choosing a shard at random to obtain these values, this effectively serves as a way to randomly sample a subset of the documents in the index.

Obtaining these values at the shard level benefits the execution time of the term vector query. By not having to coordinate the results among all shards, the values are able to be obtained quickly from a single Lucene instance and returned to the user. This approach is also scalable because the queries are distributed randomly among all shards, thereby balancing the load of the term vector queries.

RELATED WORKS

The ability to predict the performance of a query has been of interest in many database-related applications, not just full text search. An early example of using regression analysis to predict database performance is Perros' [5] research estimating the timing of funds transfer processes in banking. Perros modeled the entire transaction system, including network timings, CPU load, and disk access times. He used nonlinear regression techniques because the rate of change varied as more load was placed on the system.

More recent research has focused on estimating the performance of SQL-based database systems. Tozer *et al.* [6] used a linear regression process to estimate SQL query execution time in the context of other queries currently running on the system. Their system collected information about system performance during different workload scenarios and built a regression model that best estimated the database performance. However, their prediction models were built offline prior to serving clients. Sheikh, *et al.* [7] developed an online algorithm that used linear gaussian models to predict SQL query execution times. This approach allowed their prediction model to change as the workload on a server varied.

Full-text search engines pose different query prediction challenges than SQL-based databases. SQL queries can vary in complexity, and their structured nature means that an estimated execution plan can be generated for a query on a particular database engine. It is not uncommon for full-text search engines to use the same type of disjunctive query for every search term submitted. Unlike SQL where the query usually specifies the order in which data should be returned, one of the key challenges of full-text search is returning data sorted in the order of relevance to the user's query. Several researchers have attempted to improve search engine effectiveness and performance by predicting and in some cases shortening query times on full-text search engines.

Hauff's [8] work deals extensively with evaluating query effectiveness and accuracy in full-text search engines. Hauff makes the distinction between pre-retrieval and post-retrieval techniques. Pre-retrieval techniques use statistics available before a query is executed to estimate the query's cost, while post-retrieval techniques store and analyze search results from previous queries. Hauff examined the effectiveness of several pre-retrieval techniques using linear regression. She also proposed a new method for scoring results based on post-retrieval analysis of search terms. However, her research focused only on optimizing accuracy and relevance of search results with little regard given to search performance optimization.

Tonellotto, Macdonald, and Ounis [9] noted that "all work on query performance prediction thus far has focused upon the prediction of effectiveness, ignoring the efficiency that a query may achieve." To that end, they attempted to shorten execution times of queries by using dynamic pruning techniques to eliminate the need to evaluate all the documents in a full-text search index. Macdonald, Tonellotto, and Ounis [10] continued this research the following year, analyzing several possible variables to identify the best predictors of search engine performance. They concluded that "the most accurate efficiency predictor is the total number of postings" for queries where all documents are examined. Their results also showed that response time distributions vary based on the number of search terms in the query.

Peng and Plale [11] built an API that regulated the number of queries submitted to a Solr[3] search engine cluster in order to enforce fair sharing among users. This required accurate estimation of query costs, so they also used linear regression to build a model to estimate the CPU time used by a query. Like Macdonald, Tonellotto, and Ounis [10], Peng and Plale used the number of search terms in a query and the sum of the postings from each term in the search query as input variables for their regression model. Peng and Plale used search queries taken from AOL search engine server logs to query a large body of documents indexed from the

[3] https://lucene.apache.org/solr/

HathiTrust research library. Their experiment was conducted on a large research cluster and resulted in a reported 0.99 Pearson correlation between estimated values from their linear model and the actual search performance of the Solr cluster.

Although estimating query costs was not the primary focus of Peng and Plale's [11] research, their work made use of a linear regression model to predict CPU time. Peng and Plale [11] used the number of search terms and the sum of the postings to train their linear model, though they did not mention how they obtained the sum of the postings from the open source Solr search engine. They also used high performance supercomputer nodes to host their Solr cluster, which allowed them to store the entire contents of an index in memory and evaluate CPU time as the primary metric of query cost. However, a typical real-world search engine installation uses both RAM caching and SSD storage along with nodes that have more modest computing specifications. More research is needed in order to determine the best way to use this linear model approach to estimation within an Elasticsearch cluster installation.

Elasticsearch is highly configurable, though many users are able to install it with very little initial setup. The internal workings of the underlying Lucene indexes are abstracted from both the users and the administrators behind an API. Other researchers investigating full-text search performance have chosen to either implement their own search engine or use one that allows for very low-level inspection and configuration. Additional research is needed to determine suitable predictors of query execution timings that will work within Elasticsearch itself.

This work attempts to build on the work done by Peng and Plale [11] by replicating the linear model training portion of their research using Elasticsearch on a typical three-node cluster built with commonly available hardware. It attempts to identify a reliable process for estimating query duration using the Elasticsearch search engine. The distributed nature of Elasticsearch makes gathering information from each node in the index challenging; therefore, it is preferable to use the existing API and data provided by Elasticsearch itself. The term vectors API is a prime candidate for this task, since for each term in a query it reports a relative score for the number of documents that contain a term as well as the total number of times a term occurs. Both of these values are related to the concept of a postings list mentioned by Büttcher, Clarke, and Cormack [3], and they will be evaluated along with other variables like word count and query character length to determine the best predictors of query duration.

EXPERIMENTAL SETUP

In order to realistically evaluate search engine performance, it is important to use realistic search queries that come from real users. For this experiment, search queries were taken from an AOL search engine server log data set that contained user queries gathered from March 1 to May 31, 2006 [12]. The data were stored in several tab-separated log files. A subset of search queries from the first log file was used to train prediction models (group A), and another subset from the second log file was used to evaluate the accuracy of those models (group B). Public domain texts from the HathiTrust digital library [13] were chosen to be indexed into Elasticsearch. Each document contained the digitized text of a book, and there were 539,777 documents totaling 248 GB of data. The Elasticsearch index was allocated 12 shards to store these documents, which resulted in a little over 20GB of data stored per shard.

A three-node Elasticsearch cluster was used to perform these experiments. Each node was hosted on an Intel-based server with a 4 GHz quad-core hyperthreaded processor, and 32 GB RAM. A 1 TB NVMe SSD storage device was dedicated to storing index data, and each node had a separate SSD as a boot drive. Elasticsearch 7.8.1 was installed on each node, and 16 GB of RAM were allocated to Elasticsearch's JVM heap while the rest of the memory was left for OS file caching. Two additional computers were used to run the API and simulate user requests, and all machines were connected via a one gigabit Ethernet switch.

An API was developed using Node.js and Express to implement and evaluate the prediction models. The API had endpoints that would listen for a training request and record the execution time as well as other metrics that might be useful for constructing prediction models. The training data from the final AOL data sets were submitted to the API using the Apache JMeter [14] testing suite.

PREPARING DATA FOR ANALYSIS

Many of the search queries in the AOL data set were repeated multiple times. This is a common occurrence in user-generated data. Typically, the second or third occurrence of the same query resolves much faster because the data needed to calculate the results have already been cached in memory. This could have an adverse effect on prediction evaluations, because it may increase the variability of search result timings and result in some timings appearing artificially low. To mitigate this effect, all search terms in the AOL data set were randomized to ensure that the queries were evenly distributed.

Each search term used to evaluate query execution time should make use of the search engine's resources and not return an empty document set instantly, which could potentially throw off prediction calculations. The AOL search data contained queries that were collected during the year 2006, while the HathiTrust data contained books that have entered the public domain and were in most cases over 100 years old. Therefore, many of the words in the AOL query set had not yet entered the public lexicon at the time these books were written. The HathiTrust data also contained books in languages other than English, while the AOL search queries were almost entirely English-based. This disparity resulted in many search terms not having any results found when used to query the HathiTrust index in Elasticsearch. For example, approximately 27% of 50,000 search terms randomly chosen from group A did not return any results when used to search the HathiTrust index.

In order to prepare the AOL search queries to be used effectively with the HathiTrust library, each query from AOL groups A and B was used to query the HathiTrust index to determine if any of the words in the query were found in any of the books in the index. A final set of queries was selected from each group by querying the Elasticsearch index with each term, and only retaining the search term if documents were found. This filtering process resulted in AOL query sets that were a better fit for the HathiTrust data.

Although caching is a useful tool and typically a part of any search engine implementation, caching can also produce unpredictable and unrepeatable results when evaluating search engine performance. In addition to the random sorting of search terms previously discussed, a full run of the training data was performed whenever the Elasticsearch cluster was rebooted.

This ensured that the cluster was not running "cold" and had a full memory cache of shard segments loaded.

EVALUATING QUERY EXECUTION TIME PREDICTORS

In order to determine the best predictors of query execution time, multiple statistics were gathered during a training run of 50,000 search terms from group A. In addition to the search query, a term vectors query was also run for each term in the query to obtain the total term frequency (TTF) and document frequency (DF) data. The values that were gathered included: number of words in the query (word count), number of characters in the query (character count), the sum of all DF values for each word in the query, the maximum DF from any of the words, the sum of all TTF values for each word in the query, and the maximum TTF from any of the words.

After gathering the data, the results were filtered using the process described in section five to remove data points for queries that did not have any matching documents. The resulting 36,507 queries that had matching documents were used for this analysis. Macdonald, Tonelotto, and Ounis [10] described a query's word count as a significant predictor of query execution time, and the ordinal nature of the variable made it easy to separate into groups. The data were grouped by word count into six separate groups, with queries having six or more terms placed in the same category. A Pearson correlation analysis was run on each group of data to determine the level of correlation between each variable and the time taken to execute a query, and the results are presented in Table 1.

Table 1. Potential Predictors of Execution Time

# Words		# Chars	DF (Sum)	DF (Max)	TTF (Sum)	TTF (Max)
1	Pearson's r	-.211**	.584**	.584**	.349**	.349**
	Sig. (2-tailed)	.000	.000	.000	.000	.000
	N	5372	5372	5372	5372	5372
2	Pearson's r	-.031**	.300**	.287**	.094**	.093**
	Sig. (2-tailed)	.001	.000	.000	.000	.000
	N	12048	12048	12048	12048	12048
3	Pearson's r	-.23*	.222**	.154**	.090**	.087**
	Sig. (2-tailed)	.029	.000	.000	.000	.000
	N	8688	8688	8688	8688	8688
4	Pearson's r	-.24	.185**	.069**	.057**	.052**
	Sig. (2-tailed)	.089	.000	.000	.000	.000
	N	5027	5027	5027	5027	5027
5	Pearson's r	-.17	.123**	.005	-.010	-.022
	Sig. (2-tailed)	.388	.000	.810	.612	.265
	N	2591	2591	2591	2591	2591
6+	Pearson's r	.195*	.217**	.038*	.097**	.052**
	Sig. (2-tailed)	.000	.000	.045	.000	.006
	N	2781	2781	2781	2781	2781

* Indicates a .05 significance level. ** Indicates a .01 significance level.

The closer a correlation value is to 1, the stronger the relationship between a predicting variable and the independent variable. Negative correlations are also possible, meaning that the dependent variable will decrease as the independent variable increases. This was the case with the character count of the search query, which had a negative correlation for word counts one through five and a positive correlation of 0.195 for six or more words. The correlation was significant for all word count values except four and five.

The document frequency (DF) sum was the predictor that had the highest correlation with execution time. This was especially true of queries with one word, where the Pearson correlation value was 0.584. This means that 58.4% of the variability in time taken can be explained by the DF sum. However, the strength of the correlation began to diminish as the

word count increased. The DF sum was also the only predictor with a statistically significant correlation among all six word counts at the .01 significance level. The DF sum and DF max values were identical for a word count of one because there was only one term being evaluated. The DF max's correlation value became smaller as word count increased, and its correlations were significant for all word count values except five.

The total term frequency (TTF) sum and TTF max had an identical correlation value of 0.349 for a word count of one, but their correlation values dropped to below 0.100 for all remaining word counts. The TTF sum and TTF max values had significant correlations for all word count values except five. Based on the results of this Pearson correlation analysis, word count and document frequency sum were chosen as the predictors worthy of further investigation.

SPSS was used to perform a regression analysis on each word count group. The resulting R^2 values (Table 2) showed that the ability to predict the variability in the search results dropped precipitously as the number of words in the search term increased although the standard error of the estimate remained relatively constant.

Table 2. Regression Analysis Results for Document Frequency Sum Computed for Each Word Count Value

Number of words	R^2	Std. Error of Estimate
1	.341	24.049
2	.090	24.779
3	.049	24.416
4	.034	24.072
5	.015	25.083
6+	.047	26.328

The linear regression equations were plotted along with the observed data points to illustrate the solution obtained by the linear regression analysis (Figures 2-7). For the 1-term group (Figure 2), there was a more dramatic slope to the resulting linear equation. Higher word counts resulted in a tighter data grouping and a flatter (but still slightly positive) equation slope, as can be seen in Figures 3-6. The group with six or more words, (Figure 7) suffers from greater outliers than the other groups because it contains the few queries with relatively high numbers of words. A handful of these outliers had unusually high execution times or unusually high document frequency sums. The result of this is that the majority of the query times are clustered together without an obvious linear slope.

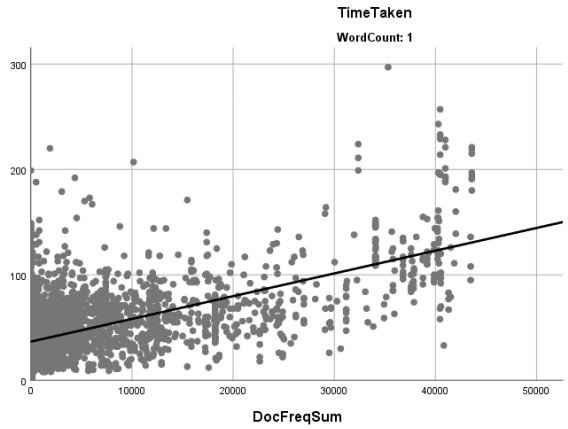


FIGURE 2. Scatter plot of document frequency sum versus time taken for one term.

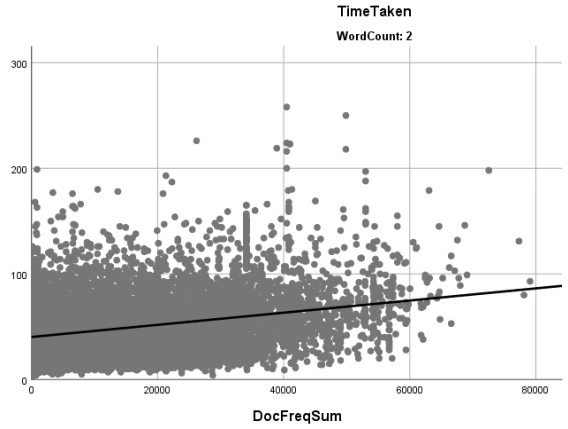


FIGURE 3. Scatter plot of document frequency sum versus time taken for two terms.

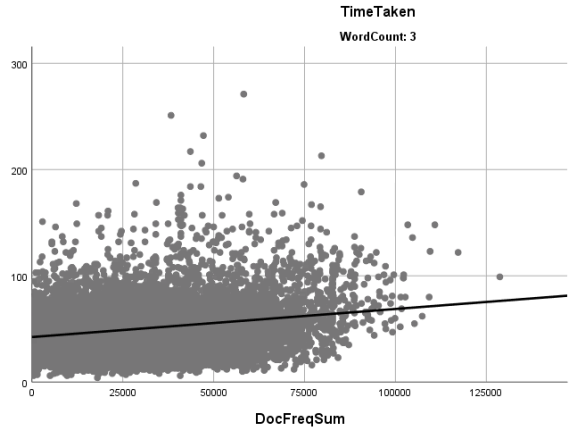


FIGURE 4. Scatter plot of document frequency sum versus time taken for three terms.

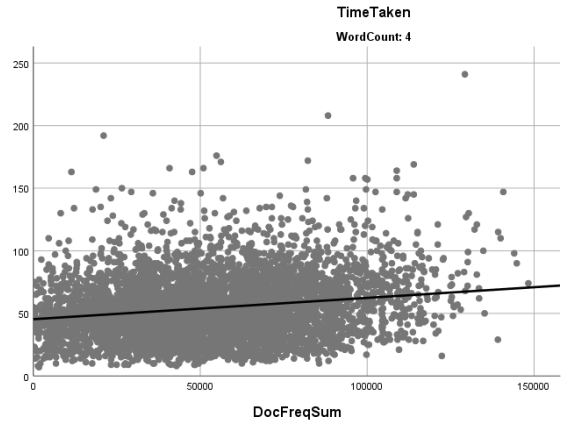


FIGURE 5. Scatter plot of document frequency sum versus time taken for four terms.

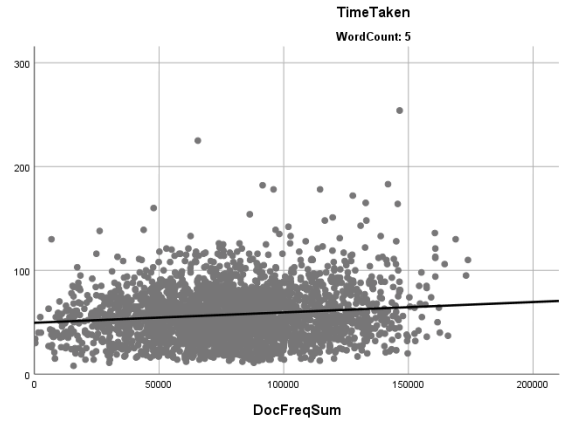


FIGURE 6. Scatter plot of document frequency sum versus time taken for five terms.

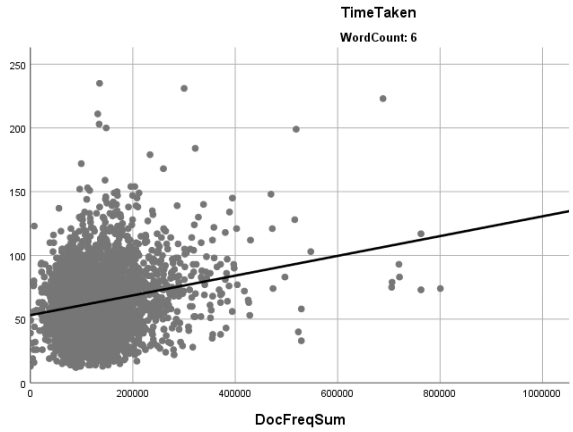


FIGURE 7. Scatter plot of document frequency sum versus time taken for six or more terms.

BUILDING QUERY COST MODELS

Peng and Plale [11] used linear regression to predict the execution time of a query on a Solr cluster. This research attempted to replicate this approach on an Elasticsearch cluster by evaluating prediction variables and regression methods to determine which provided estimates with the lowest Mean Absolute Error (MAE). MAE is calculated by averaging the absolute value of the difference between the estimated and the actual execution times for each

query. The root mean square error (RMSE) is also provided, which is the standard deviation of the prediction errors.

Each regression model used the sum of the document frequencies for each word in the query and the "took" value from an Elasticsearch query's metadata as input for training. The "took" value was the actual execution time of the query on the cluster in milliseconds. Three regression methods were used for this evaluation: simple linear regression, Thiel-Sen regression and polynomial regression. Simple linear regression provides a linear equation that minimizes the root mean squared error value for each query. However, this process can be negatively influenced by a few outliers with extremely different values. The Thiel-Sen regression method [15] is more resistant to extreme outliers by simply using the median of all slopes that pass through all data points. Finally, a polynomial regression method was also evaluated to test for the possibility that there was a non-linear relationship between execution time and the sum of the document frequencies.

Although higher order equations can be obtained with polynomial regression, the scatter plot of search timings (Figures 2-7) indicated that a second-degree equation would be a sufficient fit for the data. For most of the word count categories, the data appeared to follow a relatively linear trend and a dramatic curve in the data values was not observed.

To obtain training data for the model, a list of search terms that matched documents in the index was obtained by using the filtering process described in section five. A set of 10,000 of these search terms was submitted to the Elasticsearch cluster sequentially with no other queries running. The execution time, sum of document frequencies, and number of terms were recorded for each query. The data from these training sessions were then grouped by the number of terms and then loaded into the API to create six different regression equations for each model, one for each word count category. The regression equations for each word count category are graphed in Figures 8-13.

The regression equations for a word count of one (Figure 8) show a relatively tight grouping, with very little difference among the predicted values. All the equations in Figure 8 have a steeper slope than the rest of the categories and therefore the graph has a higher range on the Y-axis to display the full range of execution time.

The equations for a word count of two (Figure 9) are more spread out. Thiel-Sen's equation estimated values lower than those generated by simple linear regression. This is the case for all word counts of two and above, as displayed in Figures 9-13. The graph for the polynomial equation intertwines with that of the simple linear equation.

The equations for word counts of three through five, illustrated in Figures 10-12, contain a similar pattern of a polynomial curve that begins much higher than the other models and eventually merges with the other lines. However, this pattern is broken in Figure 13's graph of prediction equations for word counts of six or greater. Figure 13 contains a set of nearly parallel lines all tightly grouped together once again, similar to what is seen in Figure 8.

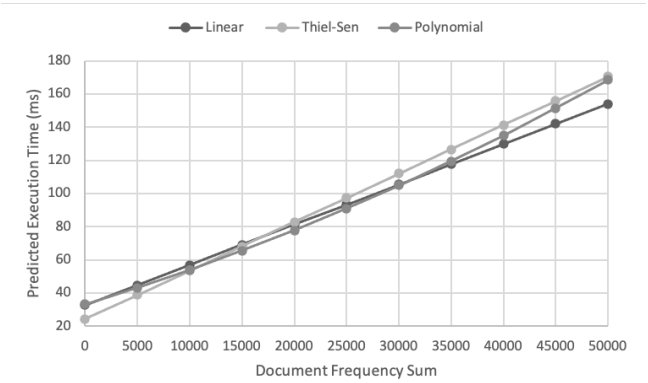


FIGURE 8. Execution time prediction models for a word count of 1.

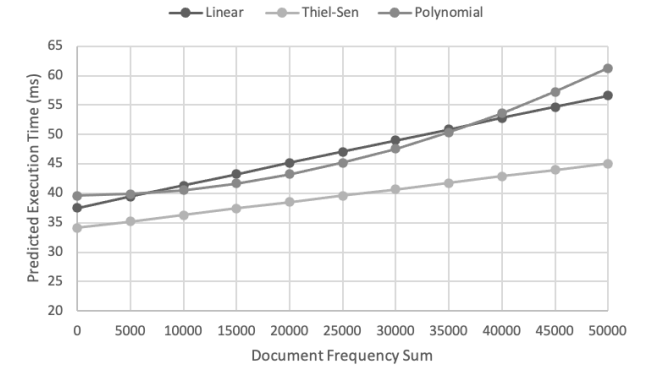


FIGURE 9. Execution time prediction models for a word count of 2.

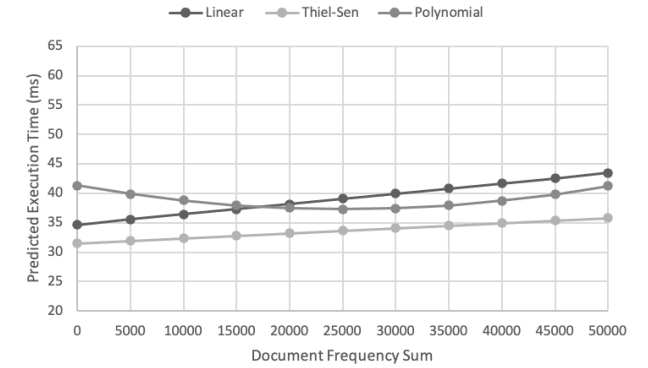


FIGURE 10. Execution time prediction models for a word count of 3.

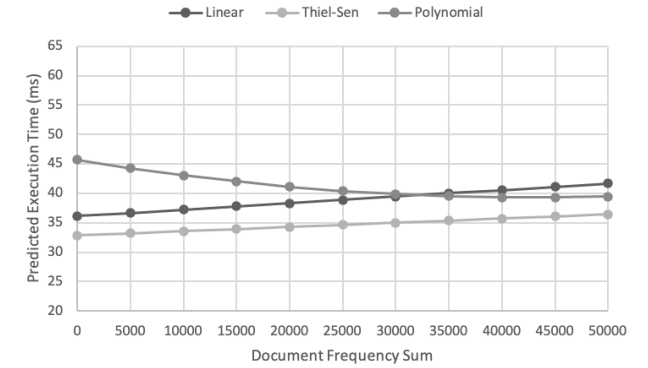


FIGURE 11. Execution time prediction models for a word count of 4.

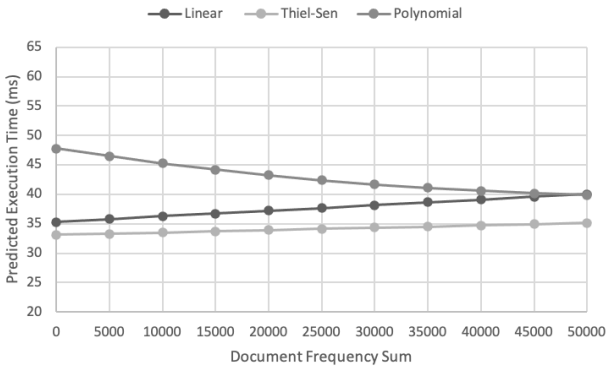


FIGURE 12. Execution time prediction models for a word count of 5.

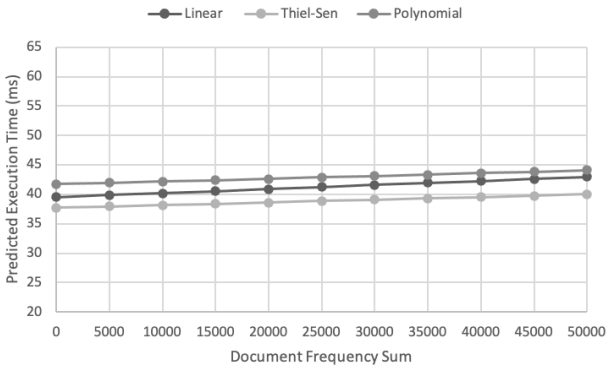


FIGURE 13. Execution time prediction models for a word count of 6 and greater.

Each of the prediction models were evaluated by submitting a second set of search terms to each model and comparing how closely the predicted values compared with the actual execution times. Another set of 10,000 search terms was created using the same filtering process on search terms from group B so that only terms with matching documents remained. These queries were then submitted to the API to evaluate the accuracy of a prediction model. For each query, both the predicted and actual execution times were recorded along with the number of terms in each query and the document frequency sums. This process was repeated for each of the three models, and the data were recorded for analysis. Two evaluation runs were performed for each prediction model. Values for each prediction model were computed and are shown in Table 3.

The best mean absolute error (MAE) time was chosen from each model's experimental runs in order to identify the one with the lowest error. The simple linear regression model achieved a best MAE of 18.716 ms and the corresponding root mean square error (RSME) value was 24.354 ms. The Thiel-Sen regression model yielded MAE values slightly higher than simple linear regression, with a best MAE value of 19.573 ms and corresponding RMSE of 26.029 ms. The second-degree polynomial regression had the lowest MAE value among the three different regression models, although its value was only 0.154 ms better than simple linear regression. The polynomial regression model had a best MAE value of 18.562 ms and a RMSE value of 24.282 ms. The standard deviation for all MAE values was 0.507 ms.

In addition to computing the mean absolute error between each model's predictions and actual observed values, the observed execution time values for each query were compared to one

another as well. The average difference in execution time for the same query on different experimental runs was computed for each regression model and is described as the mean difference between runs (MDBR). Each model used the same query set and queried the same elastic index, and all MDBR values computed were within a millisecond of each other. Values of 12.899 ms, 12.095 ms, and 12.505 ms were returned from each pair of experimental runs.

Table 3. Total Error Values For Each Prediction Model

Model	Run	Mean Execution Time (ms)	Mean Absolute Error (ms)	Root Mean Square Error (ms)	Mean Difference Between Runs (ms)
Simple Linear Regression	1	48.365	18.731	24.724	12.899
	2	47.129	18.716	24.354	---
Thiel-Sen Regression	1	47.104	19.727	26.337	12.095
	2	47.135	19.573	26.029	---
Polynomial Regression (2nd degree)	1	46.978	18.706	24.250	12.505
	2	47.111	18.562	24.282	---

Since each prediction method made use of separate equations for each of the word count categories, each of the values from Table 3 were calculated for each word count category. This allows each equation to be inspected to determine if certain word count values have higher mean absolute error (MAE) values than others. Tables 4-6 contain MAE and RMSE values for each regression model, broken down by the word count category. They also contain the mean difference between runs (MDBR) computed only for queries in that specific word count category. Additionally, the MAE values for each prediction model are graphed in Figures 14-16 for better visualization.

Table 4 contains the mean absolute error (MAE) for each word count category for the simple linear regression model. The MAE values remain constant around 18.500 ms for word counts one through five. For the six and greater word count, the best MAE value increases to 20.499 ms, although this is also accompanied by an increase in the MDBR.

Table 5 contains the MAE values for the Thiel-Sen regression model. The MAE values are slightly higher than the simple linear regression model for every word count category, especially for word counts of one and six. The polynomial regression model follows a pattern similar to simple linear regression, with a MAE value hovering around 18.500 ms.

The mean difference between runs (MDBR) grew slightly worse as word count increased. For the simple linear regression model, the MDBR slowly grew from 11.380 ms to 14.120 ms as the word count increased from one to five words, before finishing at 16.394 ms for the six words or greater category. Figures 14-16 illustrate that this behavior was almost identical for every prediction method, with an increase from approximately 11ms to 16ms observed over the word count range.

Table 4. Error Values For Simple Linear Regression By Word Count

Word Count	Run	Mean Execution Time (ms)	Mean Absolute Error (ms)	Root Mean Square Error (ms)	Mean Difference Between Runs (ms)
1	1	47.343	18.577	24.595	11.380
	2	47.159	18.243	24.010	---
2	1	48.460	18.589	24.815	11.936
	2	47.436	18.796	24.737	---
3	1	46.621	18.592	24.354	12.995
	2	45.111	18.357	23.793	---
4	1	47.697	18.200	23.448	14.190
	2	46.208	18.894	23.591	---
5	1	50.291	19.336	25.438	14.120
	2	47.600	18.257	24.076	---
6+	1	54.812	20.499	27.175	16.394
	2	53.119	20.523	26.568	---

Table 5. Error Values For Thiel-Sen Regression By Word Count

Word Count	Run	Mean Execution Time (ms)	Mean Absolute Error (ms)	Root Mean Square Error (ms)	Mean Difference Between Runs (ms)
1	1	47.725	21.033	28.085	10.481
	2	46.863	20.383	27.073	---
2	1	47.067	19.455	26.093	11.016
	2	47.451	19.415	26.249	---
3	1	45.217	19.232	26.050	12.638
	2	45.011	18.973	25.267	---
4	1	46.516	19.113	24.777	13.334
	2	46.491	19.305	24.805	---
5	1	47.399	19.143	25.820	12.797
	2	48.522	20.077	27.126	---
6+	1	52.719	21.570	27.930	15.483
	2	52.762	20.569	26.397	---

Table 6. Error Values For Polynomial Regression By Word Count

Word Count	Run	Mean Execution Time (ms)	Mean Absolute Error (ms)	Root Mean Square Error (ms)	Mean Difference Between Runs (ms)
1	1	47.372	18.843	24.617	10.670
	2	47.094	18.593	24.204	---
2	1	47.224	18.746	24.510	11.919
	2	47.308	18.369	24.510	---
3	1	45.169	18.172	23.768	12.607
	2	45.290	17.918	23.580	---
4	1	46.162	18.481	23.256	13.618
	2	46.565	19.048	24.038	---
5	1	47.215	18.690	23.819	13.543
	2	47.111	18.333	23.786	---
6+	1	52.039	20.345	25.957	15.427
	2	52.960	20.702	26.390	---

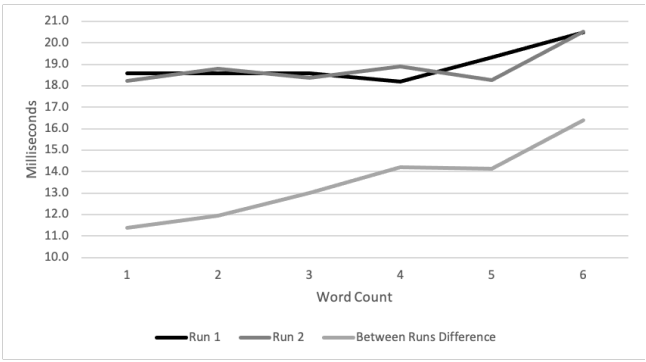


FIGURE 14. Mean absolute error values for simple linear regression.

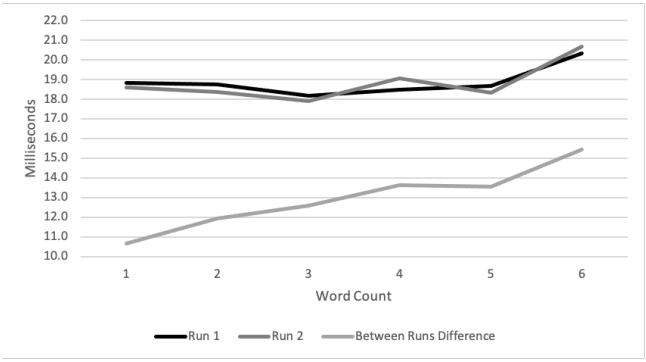


FIGURE 15. Mean absolute error values for Thiel-Sen regression.

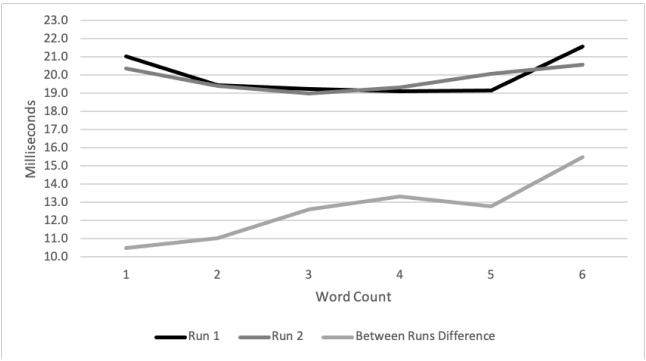


FIGURE 16. Mean absolute error values for polynomial regression.

DISCUSSION

Several possible predictors were evaluated to determine which could best be used to predict execution time of an Elasticsearch query. The results of a Pearson correlation analysis revealed that the sum of document frequencies for each word in a query was a significant predictor. This became even more apparent when the search timing data were segmented by word count. The sum of document frequencies was the only predictor that remained significant for each of the word count categories. The importance of word count and the sum of document frequencies is consistent with the findings of Macdonald, Tonelotto, and Ounis [10], who also said that they found them to be predictors of query execution time. This finding is also consistent with the formula from [1] for the time complexity of the document-at-a-time search model.

The correlation value for the sum of document frequencies was at its highest for queries with a single word and dropped off to a lower value as the word count increased. This drop-off behavior was also observed in the R^2 values obtained when performing a regression analysis. This may be due to the increased pre-processing activities that are necessary to analyze each term before the postings are queried. For example, Elasticsearch needs to examine each word in the query for fuzzy transpositions, and as the word count grows this fixed cost per word will slightly increase search times independent of the number of matching documents in the index.

Various regression techniques were used to attempt to predict execution times using the sum of document frequencies for a search query. Because word count was found to be a significant predictor of execution times but was also an ordinal variable, different regression equations were computed for six different word count values. The three different prediction methods that were evaluated had very little difference among their mean average error (MAE) values, with a 0.507 ms standard deviation observed among all MAE values found in table three. Figures 8-13 illustrate the similarity in the predicted values obtained by each regression model. Although the values generated by the models deviate at times, the overall closeness in their output values is likely the reason why their performance is so similar. Given the lack of substantial difference among the performance of the models, use of the simple linear regression model is recommended due to its widespread use as a value prediction method.

Although individual query predictions could be off by a few dozen milliseconds, the average estimate generated by each model was pretty good. The average MAE for all word counts using the simple linear regression model was 18.72ms, which is 5.82 ms higher than the between runs difference value. Figure 14 illustrates that the MAE remained consistently in the range of 18.5ms for word counts one through five even though the mean difference between runs (MDBR) steadily increased. This resulted in even more accurate predictions as the word count increased. The MDBR can be considered a lower bound beyond which an execution prediction model cannot hope to improve upon since this level of variability is inherent in a series of actual queries on the cluster itself. This means that the average error produced through the prediction process was only a few milliseconds higher than the average time variations encountered between two search engine queries with the same query terms.

The speed with which these estimates are generated is also very important and contributes a great deal towards the practical application of this work. If the estimation process takes longer than the query itself would take, the estimate is of little practical value. In these experiments, the average time taken to query the term vectors API was 2 ms, which is much faster than a typical Elasticsearch query. This is the most time-consuming part of the estimation process since it requires a network request to the cluster. Once the sum of the document frequencies is obtained, computing the predicted value using the linear equation is trivial. Developers who would like to use this process to estimate queries must take into account the latency introduced by this estimation process as part of their application design.

CONCLUSION

This research investigated the use of statistics available through Elasticsearch to accurately predict the execution time of queries on an Elasticsearch cluster. It confirmed that the number of postings and the word count are both significant predictors of the execution time of a query. Although the precise number of postings was not readily available due to the distributed nature

of Elasticsearch, the term vectors API provided a document sum value that served as a relative measure of how frequently a term appears in the index. The sum of these document frequency values proved to be a significant predictor of execution time in Elasticsearch.

Several regression models were evaluated to determine accuracy in predicting query execution time. The regression models yielded nearly identical results, and the simple linear regression model was shown to predict execution times with a mean average error of 5.82 ms higher than the mean difference observed between evaluation runs. This research shows that it is also possible to predict query execution times for Elasticsearch with a relatively small amount of error.

This research has produced a method for estimating Elasticsearch query performance, but further work is required to apply this method within practical applications such as rate limiting or load balancing scenarios. More research is also needed to determine how the mean average error of this prediction method is affected as both the Elasticsearch cluster and the index size grows larger. Equivalent term vector information also exists in Solr, a similar search engine also based on Lucene. It may also be useful to examine if this information can be used to predict query execution time on that search engine platform as well.

ACKNOWLEDGEMENTS

This work has been partially supported by the Askew Institute of the University of West Florida.

References

[1]. Gormley, C. and Tong, Z., *Elasticsearch: The definitive guide,* Sebastopol, CA, USA: O'Reilly, 2015.

[2]. Elasticsearch Reference. *Match Query.* [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-query.html, Accessed on: Sept. 5, 2020

[3]. Büttcher, S., Clarke, C. L. and Cormack, G. V., *Information retrieval: Implementing and evaluating search engines*. Cambridge, MA, USA: The MIT Press, 2016. p. 137-145.

[4]. Elasticsearch Reference. *Term Vectors API.* [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-termvectors.html, Accessed on: Sept. 5, 2020

[5]. Perros, H. G., *A model for predicting the response time of an on-line system for electronic fund transfer.* ACM SIGMETRICS Performance Evaluation Review. 12(1): DOI: 10.1145/1041818.1041820, Accessed on: Sept. 5, 2020,

[6]. Tozer, S., Brecht, T. and Aboulnaga, A., *Q-Cop: Avoiding bad query mixes to minimize client timeouts under heavy loads.* ICDE 2010. DOI: 10.1109/ICDE.2010.5447850.

[7]. Sheikh, M. B., Minhas, U. F., Khan, O. Z., Aboulnaga, A., Poupart P. and Taylor, D. J., *A bayesian approach to online performance modeling for database appliances using gaussian models.* ICAC 2011. DOI: 10.1145/1998582.1998603.

[8]. Hauff, C., *Predicting the effectiveness of queries and retrieval systems*. Ph.D. Dissertation, Department of Electrical Engineering, Mathematics and Computer Science, University of Twente, Enschede, NL, 2010.

[9]. Tonellotto, N., Macdonald, C. and Ounis, I., *Query efficiency prediction for dynamic pruning*. ACM LSDS-IR '11. DOI: 10.1145/2064730.2064734.

[10]. Macdonald, C., Tonellotto, N. and Ounis, I., *Learning to predict response times for online query scheduling*. ACM SIGIR '12. DOI: 10.1145/2348283.2348367.

[11]. Peng, Z. and Plale, B., *A Multi-tenant Fair Share Approach to Full-text Search Engine*. DataCloud 2016. DOI: 10.1109/datacloud.2016.010.

[12]. Search Data Collection. [Online]. Available: https://archive.org/details/AOL_search_data_leak_2006, Accessed on: Sept. 15, 2020.

[13]. HathiTrust Digital Library Datasets. [Online]. Available: https://www.hathitrust.org/datasets, Accessed on: Sept. 15, 2020.

[14]. Apache JMeter. [Online]. Available: https://www.jmeter.apache.org, Accessed on: Sept. 15, 2020.

[15]. Theil, H., A Rank-Invariant Method of Linear and Polynomial Regression Analysis. In Henri Theil's contributions to economics and econometrics. Volume 1. Econometric theory and methodology B. Raj and J. Koerts, Eds. Springer Netherlands, pp. 345-381.