Ατομική Διπλωματική Εργασία

Search Server Characterization

Ανδρέας Πρόξενος

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2019

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Search Server Characterization

Ανδρέας Πρόξενος

Επιβλέπων Καθηγητής

Σαζεϊδης Γιάννος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2019

Acknowledgments

I like to thank Mr. Yanos Sazeides who guided me and gave me feedback in the course of my dissertation. I thank Zacharias Hadjilambrou and Georgia Antoniou, who were always willing to help me go through problems that I encountered. Also, I like to thank Mr. Chrysostomos Nicopoulos for providing me with the infrastructure for conducting the experiments.

Abstract

The rapid increase of information has increased the need for efficient search. Search engines have huge traffic and need a lot of resources in order to provide service.

This kind of systems are highly configurable and in order to choose the best configuration for guarantying QoS, experiments have to be conducted.

In search engine experiments a sample of queries are served in order to characterize the latencies. As sample theory implies, there is a trade-off between precision and cost of experiment and it is based on the sample size. So we need to give an insight on what is the optimal size of experiment sample for having cost as low as possible and precision as high as possible. Through this dissertation, we conducted experiments on an example of a popular open source search engine, Apache Solr.

We analyze how the average and tail latency is affected by the intra-server spread of index load (number of shards) for different traffic levels. At the same time we analyze how the results and their precision change, while the size of the experiment (in terms of number of queries in a single experiment and number of runs of the same experiment) changes. We try to find out how many queries and how many runs per experiments are needed in order to have a desired precision.

Through the analysis we give a precision level that can be reached on every configuration. We indicate where is worth having large sample and where the experiment can be precise with smaller sample. We show how the precision of experiments is affected by the traffic level on every number of shard configuration. Furthermore, we quantify the imprecision range using the width of Confidence Interval.

Contents

Chapter 1

Introduction

1.1 Online Searching

Online search is the application of Information retrieval on online services. It comes in the form of enterprise search, web search, email search, etc. These services are provided by the most popular applications. In 2010 Google reported that it was serving more than a billion searches per day, which means over 10 000 queries per second [13].

1.2 Search Server experiments

In search engines, experiments are performed in order to determine whether a configuration offers a desire behavior. Search engine services need to guarantee a QoS in terms of latency, because when a search engine does not serve requests fast enough (fraction of second), there could be decrease in the services' revenue due to dissatisfaction of the users [3]. So, before deploying a search server or before applying a different configuration on an already deployed search server it is needed to make sure that it will not cause slowdown in response time, and it will be beneficial for the service. To make that sure, experiments have to be conducted to measure the latencies.

1.3 Motivation

In the experiments of search engines the size of experiment has two dimensions: the number of queries for a single experiment and the number of times an experiment has to be repeated.

As sampling theory implies [9], the sample size gives a trade-off between accuracy of result and cost of experiment. In this work, we analyze how large the experiment have to be, in both dimensions, in order to have a desired precision. Precision is measured using the width of confidence interval with 95% statistical confidence. In other words, the question that we want to answer is how many queries to send to the search engine for a single experiment and how many runs to do of the same experiment. Also, we want to know what configuration benefits the response times of the requests, at what conditions and how confident we can be that the specific configuration is indeed beneficial.

1.4 Contribution

1.4.1 Setup and configuring software

To perform the search server experiments we used Apache Solr, which is based on Apache Lucene, an academically accepted online search benchmark [2]. The search server, the client and the configuration server are the components of the setup, and each one is installed on a different machine. The whole setup implements an online search service, which is used for the experiments that will be analyzed. This setup
, combined with the knowledge that is gained by building it, is a useful framework for more experiments on the topic of online searching and a good benchmark for experiments on system configurations.

1.4.2 Parameters that affect latency and experiment precision

We conclude that higher spread of index load is beneficial for latency on low traffic levels but detrimental on high traffic levels. We give a precision level that can be reached on every configuration. We indicate where is worth having large sample and where the experiment can be precise with smaller sample. We show how the precision of experiments is affected by the traffic level. More specifically, we show that configurations with high spread of index load are affected more than configurations with low spread of index load on precision of result. Even with high sample size the precision of high spread of index load configurations cannot converge to the level of precision of lower spread of index load configurations.

For every parameter we give an error range (a confidence interval with 95% statistical confidence) in order to guarantee QoS. We show precision vs size of experiment to give insight to where to draw the line between cost of experiment and precision of experiment.

Furthermore, outliers are observed in the experiments and that suggests multiple runs per experiment.

1.5 Organization of Following Chapters

The organization is made as follows: In chapter 2 there is a more detailed explanation about the importance of search engine experiments and what the popular metrics are for search engine experiments. In chapter 3 there is an explanation of the statistical formulas and methods that are used for the experiment statistical analysis. In chapter 4, the infrastructure of the experiments is described. Chapter 5, describes the experimental methodology. Chapter 6 has the experiment results including graphs and short comments about graphs. Chapter 7 discuss further the results and concludes the purpose and the contribution of the dissertation.

Chapter 2

Importance of Experiments on Search Engines

2.1 Importance of experiments for guaranteeing search engine QoS.

A major challenge for the large-scale online searching services is the management of tail latency of search requests. [3] The search servers' software and hardware is frequently upgraded with more features in every version that might benefit the tail latencies. In order for the search servers managers to decide whether a new upgrade and in general a new configuration will be beneficial for the latencies, confidence and understanding is needed. The only way to be confident and to understand what will be the outcome, is to conduct experiments out of production.

In a typical experiment, a portion of a query log (10,000-100,000 queries) is sent as request to the search engines. [1] [2] suggest sending queries using interarrival rate. When all the queries are served, the average latency and the tail latency (90th,95th,99th percentiles) are measured. For every different configuration the same procedure is done and the configuration with overall lowest latencies is chosen as the optimal. The same experiment can be conducted on the same configuration multiple times.

But, by conducting experiments out of the production means that the server managers are facing the risk of not being precise enough and resulting in conclusions that are not valid. For example a QoS can be guaranteed for tail latency in the environment of the experiment but due to imprecision, the production tail latency may exceeds the QoS guarantee.

2.2 Popular Experiment metrics

The metrics for service latency analysis that are widely used for the mean latency are the average and for the tail latency the 90th, 95th and 99th percentile. In this work, the average will be used as metric for mean latency, and 99th percentile is used as metric for tail latency.

Mean latency is not enough to represent the QoS of the server because for example, even if the mean latency is under the QoS latency threshold, it is important to examine that the grand majority of the requests are under the QoS latency threshold too. That is the reason that 99th percentile is used. It can be guaranteed that 99 percent of the requests are serviced under a certain value of latency. [2] [3]

2.3 Experiment Parameters

The configurations that will be examined in this work are the number of index shards and the interarrival rate of the requests. Index sharding is a technique used by search servers and databases in order to spread load to the resources. Interarrival rate of requests denotes the traffic of the server. Another parameter is the sample size. It comes in two dimensions. The length of the experiment, which means how many queries will be served in a single experiment run and the width of the experiment, which means the number of runs of the same experiment.

2.4 Related work

2.4.1 “Characterization and analysis of a Web Search benchmark” [2]

An example of research on search server experiments to optimize latency. The framework is the Nutch benchmark, which internally is very close to the Solr. They are both based on Lucene.

They analyze how beneficial is intra-server partitioning, which is the corresponding with sharding on Solr and having the shards on the same machine, on different system configurations and other parameters including traffic level. In this research the experiments are not evaluated by statistical methods in order to be determined whether they are precise.

2.4.2 “Treadmill: Attributing the Source of Tail Latency through Precise Load Testing and Statistical Inference” [1]

In this work they are using statistical methods to evaluate the results’ precision but the workload is not a search engine. Nevertheless, it has various ways of measuring precision and it can be a future work to use them for our framework.

Chapter 3

Statistics

3.1 Confidence Interval and Statistical confidence

Confidence interval is an interval around the value of a parameter that is measured by a sample of a population. In this interval it might be included the true value of the population parameter, so it is a good approximation. But it does not always include the value of the true population parameter and in order to measure the precision of the interval there is the statistical level parameter. Usual confidence levels are 90%, 95% and 99%. For example, 95% confidence level means that for the 100 confidence interval we observe, 95 of them will include the true value of the population parameter.

The width of the confidence interval is affected by the confidence level that we set, the sample size and the variance between the values in the sample.
The width is larger when the confidence level is high, the variance is high and the sample is small. [9]

3.2 Formulas

Formula for calculating confidence intervals of median (50th percentile) and 99th percentile. This formula is non parametric which means that it can be applied in no-normal distributed data. [6]

r’ = n*q – (N_{1-a/2} * sqrt{n*q*(1-q)})
s’ = 1+ n*q + (N_{1-a/2} * sqrt{n*q*(1-q)})

Where r’ and s’ are the r-th and the s-th observation in the ascending sorted sample. The confidence interval is the interval between the values of those two observations.

- n is the sample size
- q is the quantile that the interval will be produced
- N_{1-a/2} is the appropriate value from the standard Normal distribution for the 100(1-a/2) percentile.

3.3 Bootstrapping

Bootstrapping is a computer based technique that aims on estimating the true value of a parameter of a population by a sample. For example to estimate the true average of the population. It is a resampling technique, and it independently creates sample with replacement from a given sample data, and it performs inference using all the data set including the resampled data. For this work, it is used to calculate confidence interval on the average. It is non parametric so it can be applied to non-normal data. There are libraries for Bootstrapping in the popular scripting languages for statistics e.g. R and Matlab. [7]

3.4 Distribution normality test.

Distribution tests help to identify if a data set follows normal distribution. More specifically, a hypothesis is made that the data follow normal distribution. Base on a confidence interval, usually 95%, the divergence of data is computed. The parameter that shows the divergence is the P-value. If it is smaller than 1-the percentage of confidence, in this example 1-.95=0.05, then the hypothesis can be rejected, so the data probably do not follow normal distribution. Else we accept the hypothesis. The smaller the P-value the less certain it is that the hypothesis is correct. [12]

Chapter 4

The infrastructure

3.1 Description

The search server that is going to be characterized is the Apache Solr, which is an open source enterprise search platform. It is written in Java and it is based on Apache Lucene (a popular open-source information retrieval software library).

The most important features of Apache Solr for this analysis is full text search, distributed search, indexing, index sharding and caching. This features will be explained in the following subsections. [4]

Popular search engines are also the ElasticSearch, Endeca, FredHopper, Mercado, Google Mini, Microsoft Search Server, Autonomy and Microsoft Search Server Express. The reason that we selected this specific server is because it is free, it is an academic accepted benchmark, similar experiments are conducted on Lucene (which is based on), and through our own specification comparison with its competitors, it is the most configurable and extensive.

3.2 Architecture

To simulate the Search server production environment, three software components are used.

Those components are the server application (Apache Solr), the configuration server (Zookeeper) and the client. Each software component is installed on a different machine in Nicocluster, a cluster in the University of Cyprus. More specifically, client is deployed on server named Nicocluster8, Zookeeper is deployed on Nicocluster9 and server on Nicocluster10. The general information flow is the following: Client creates requests for full text searching and sends them to server.

Server searches the index, creates responses and sends them back to client. It keeps its state by communicating with Zookeeper.

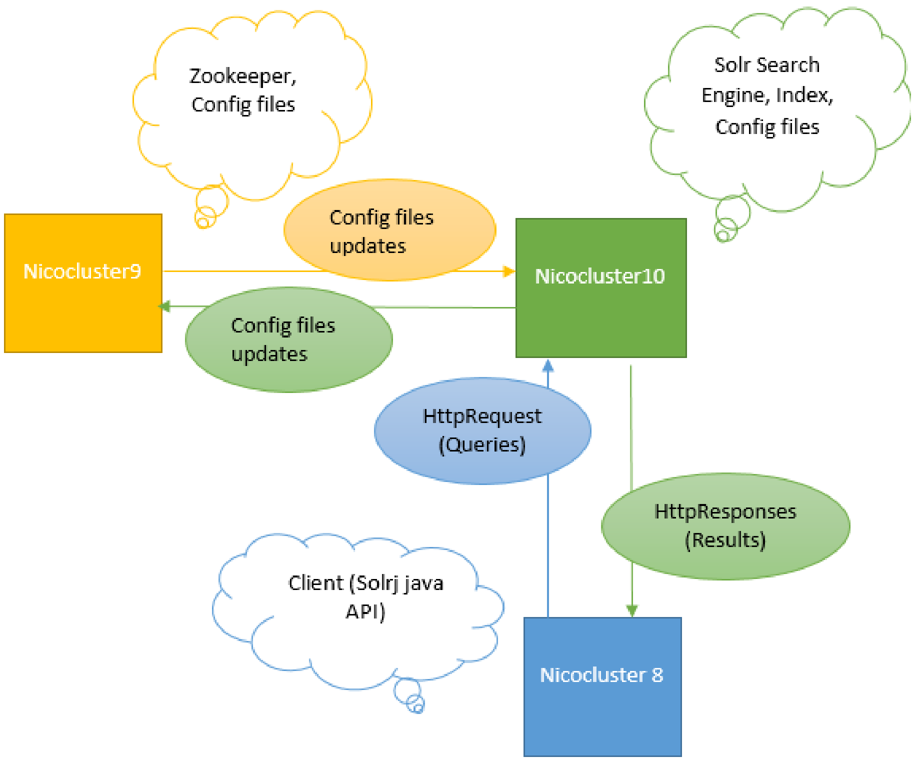

Figure 1. Communication between nodes

4.3 The search server

4.3.1 Indexing

Search engines, in order to be more efficient on searching a term over a number of documents, instead of searching the actual documents for every search request they go through all documents only one time and they create a data structure that is called inverted index. This data structure is similar to a term index that it is found at the end of some books.

Same happens with Apache Lucene index.

The structure goes as follows: An array holds all the terms of the index. It is sorted alphabetically in order to be able to perform search of a term using binary search. In a different array which is parallel to the term array, there are pointers to the posting list of a term. Posting list is called a list of pairs of the values documentID and termFrequency, which are the ID of a document that contains at least one time the corresponding term and how many time it contains the term. So for every document that the term is included, there is a pair on its posting list. It can be sorted by a custom scoring schema, for example pagerank, but the default scoring is by documentID. That makes easy the intersection of two posting lists while searching a query with more than one term. [2] Expect from enabling fast searching, index is also efficient way of storage. It compresses the pairs in posting list using a simple compression algorithm, called variable integer format, which is using the first bit of each byte to determine whether there are more bytes left. [5]

ID	Text
1	We waited here for five months.
2	Months later we found out why.

Table 1. Examples of documents

Term:	DocFreq:	<DocId,TermFreq>
we	2	<1,1>,<2,1>
waited	1	<1,1>
here	1	<1,1>
for	1	<1,1>
five	1	<1,1>
months	2	<1,1>,<2,1>
later	1	<2,1>
found	1	<2,1>
out	1	<2,1>
why	1	<2,1>

Table 2. Representation of Table 1 in index

4.3.2 Index Sharding

A Lucene index with its vital configuration files, which can serve request independently is called Solr Core. We need to create a Solr Core to perform operations like indexing and searching.
To enable better utilization of the resources, the documents are not stored in the same Core. We split the document dataset in multiple cores in order to search through them in parallel. This method is called Sharding. Each core is called a shard of the index and it contains a disjoint subset of the documents in the index. The logical join of the all shards of an index is called Solr Collection. Each shard can run on different machine or on the same machine. In this work all shards are on the same machine. [5] [10]

4.3.3 Searching

When a Solr node receives a search request, the request is routed behind the scenes to a shard that is part of the collection being searched. The chosen shard acts as an aggregator: it creates internal requests to randomly chosen replicas of every shard in the collection, coordinates the responses, and constructs the final response for the client. By receiving a request, the collection internally reacts as follows: the request will be routed by an idle shard to all the shards of the same collection. The request is executed in each shard independently and then the shard that routed the request will receive back the responses. It will aggregate the responses and send them back to the client. Now let’s discuss the index search procedure in each Solr Shard [10]. First it is necessary to state that in the experiments only conjunctive (AND) queries were used to the explanation, even though it is similar to OR queries, will be explained accordingly. The procedure of executing a conjunctive query is splitted in three major steps: the binary search, the ranking and the sorting. First, binary search is conducted on the term array in index in order to find the posting list for every term that is searched. Then, the posting lists of all terms are intersected and a score is assigned to each document based on tf.idf weighing scheme, where the most relevant documents are the ones that have the rarest term more frequently [8]. Last is the sorting of the posting list based on the ranking and the selection of the top n documents. N is specified in the request. As a summary, the total search time is the summation of binary search in term table, the posting list intersection, the ranking of each document and the sorting based on the ranking. [2]

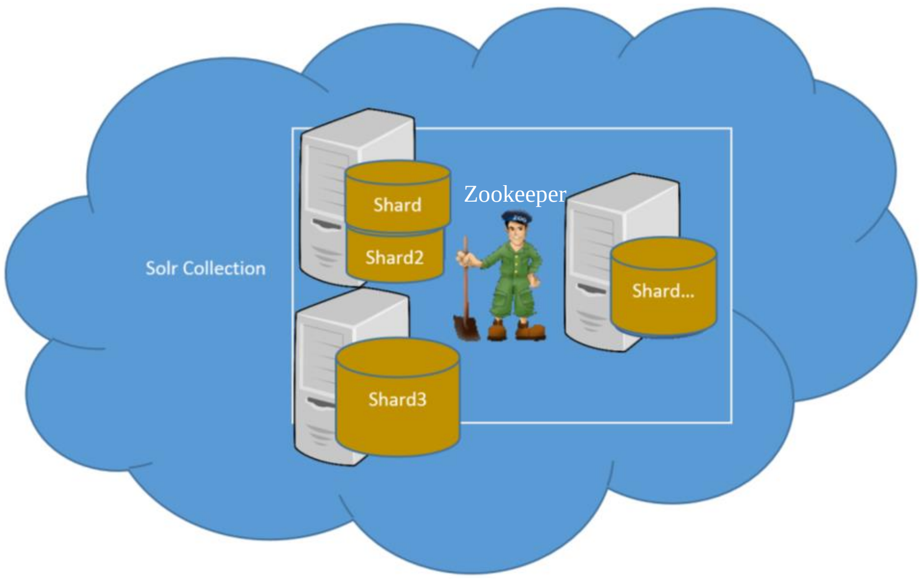

Figure 2. Solr Cloud Architecture

4.4 The centralized configuration server

Zookeeper is responsible for maintaining the configuration files of the server. It is centralized which means that instead of having the configuration files spread on multiple machines or multiple directory it holds them to a centralized directory and every server can find its corresponding configuration via Zookeeper. It has the option of having an ensemble of Zookeepers so if on crushes the configuration file will be available by the other members of the ensemble. In Solr, Zookeeper provides centralized configuration to the group of cores – shards that belong to the same collection. So same configuration is shared among cores belonging to same collection. A collection keeps its state through Zookeeper (e.g. what cores are active or inactive in order to forward requests only to active cores). [11]

4.5 The client

Client simulates the users who will send requests to the server. A client can start multiple threads and each thread sends a query and receives its response. The threads are created and run based on inter-arrival time poison distribution. Client is based on SolrJ Java API, which is an API that makes it easy for applications written in Java to talk to Solr. [11]

4.6 The access and interaction to the system

The three nodes are communicating by HTTP requests. Each node is aware of the IP address and the Port Number of the rest of the nodes. The connections to the system were all through SSH protocol (for command line) and SCP protocol for transferring files.

Chapter 5

Experimental Methodology

5.1 Hardware Specification

The three nodes used for the experiments are 3 servers from Nicocluster, a server cluster of University of Cyprus. The specs of each server are the following: (by ISCPU Linux command)

Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
Byte Order	Little Endian
CPU(s)	16
On-line CPU(s) list	0-15
Thread(s) per core	2
Core(s) per socket	4
Socket(s)	2

L3 cache	12288K
NUMA node0 CPU(s)	0,2,4,6,8,10,12,14
NUMA node1 CPU(s)	1,3,5,7,9,11,13,15
Virtualization	VT-x
L1d cache	32K
L1i cache	32K
Memory	32GB
Turbo Boost and SMT	Enabled

NUMA node(s)	2
Vendor ID	GenuineIntel
CPU family	6
Model	44
Model name	Intel(R) Xeon(R) CPU E5620 @ 2.40GHz
Stepping	2
CPU MHz	2394.000
BogoMIPS	4798.15

5.2 Index Dataset Details

Search server has indexed the English Wikipedia dump files (enwiki-latest-pages-articles.xml version 21-5-2018), whose size is 65GB.
Size of index depends on number of Shards: (because of the overhead of multiple shards)

1 Shard	16 788 MB
2 Shards	17 218 MB
4 Shards	17 483 MB
8 Shards	18 634 MB

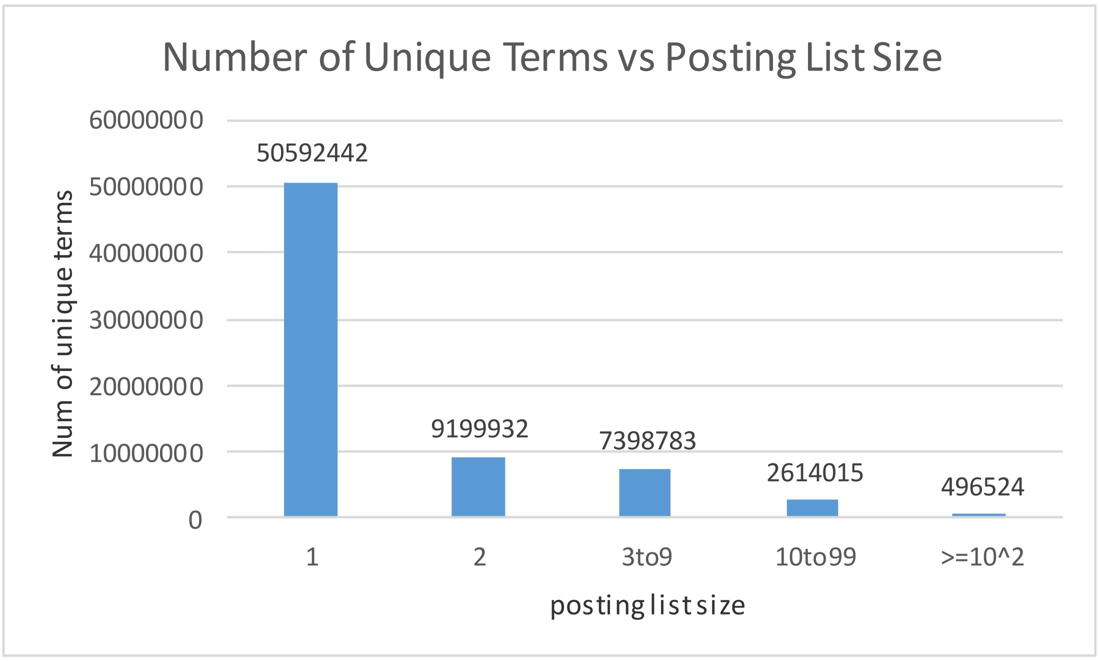

Figure 3. Number of Unique Terms vs Posting List Size

The above histogram shows the number of unique terms and their posting list size. For example, the graph shows that more than 50 million terms are found in only one document. 289 terms are found on 1000000 to 6221433 documents. Most popular term is the term "from" and it is found in 6221433 documents.

The total number of documents in the index is 12864220.

The total number of unique terms in the index is 70301697.

5.3 Query Stream Details

The query stream that is used for the experiments is taken from AOL query log that was released in 2006. This query log contains 36 million queries that have been submitted by the users of AOL search engine. To create the query stream that it will be used for experiments we selected 1 million queries that are unique (exists only once in the query stream) and match at least 1 document from the index, because the queries in AOL query log are not intended for Wikipedia documents and a lot of queries did not match any document. [14]

We are aware of the controversy surrounding the AOL query log because of privacy issues. We want to state that we do not use the AOL log for people identification. We use the real-life representative queries only for analysis purposes.

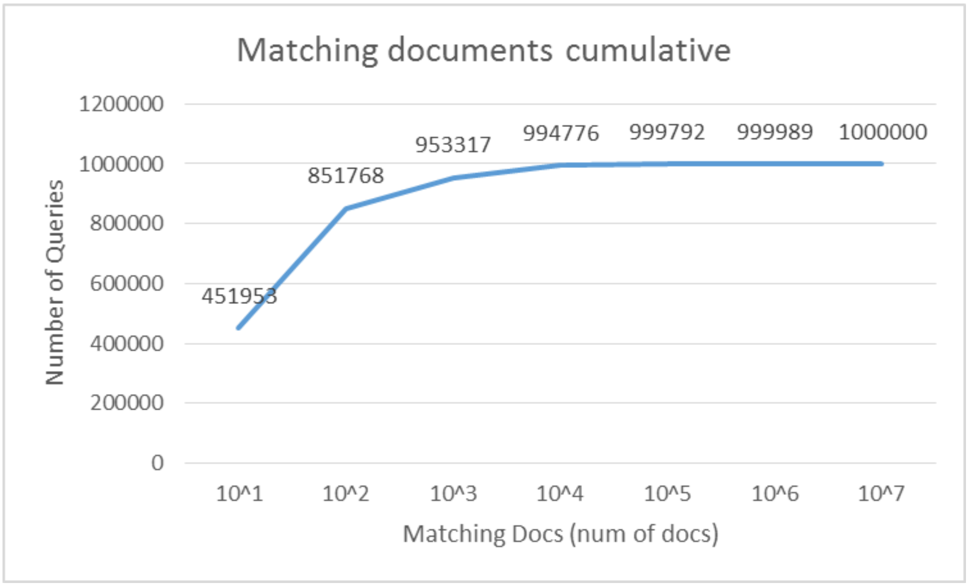

Figure 4. Matching Documents Cumulative

The graph above is the cumulative number of matching documents for a number of queries.

Y axis shows the number of queries that match less document than the corresponding number on the x axis.

For example, 451953 queries match from 1 to 9 documents, 851768 queries match from 1 to 999 documents and continues until 1000000 queries that in this case match less than 10 000 000 queries.

5.4 Experiment structure

The parameters of the analysis are the following:

a) Interarrival rate of the requests
b) Number of shards on the server
c) Sample length

d) Variability between runs of the same experiment

The metrics used for analysis are:

a) Average latency
b) tail latency - 99th percentile
c) 95% confidence interval of each metric

Experiments with 50 q/s,100 q/s and 150 q/s interarrival rate are performed for each of the shard configuration in order to analyze the behavior of each configuration from low traffic to high traffic.

Each experiment is performed 10 times in order to analyze the variability between the runs of the same experiment. Coefficient of variation is used to measure the variability.

The values of the metrics are shown every 10000 queries in order to analyze them according to the sample size.

Note: All confidence intervals have 95% statistical confidence

3.9 Technologies used for experimental analysis

UNIX shell script has been used to automate experiment procedures, java for processing raw data and transforming them to excel files or input for R scripts. Excel is used for the graphs, R for implementing bootstrapping and Minitab for the distribution analysis.

Chapter 6

Experiments and Results

6.1 Distribution analysis

6.1.1 Histogram and CDF analysis

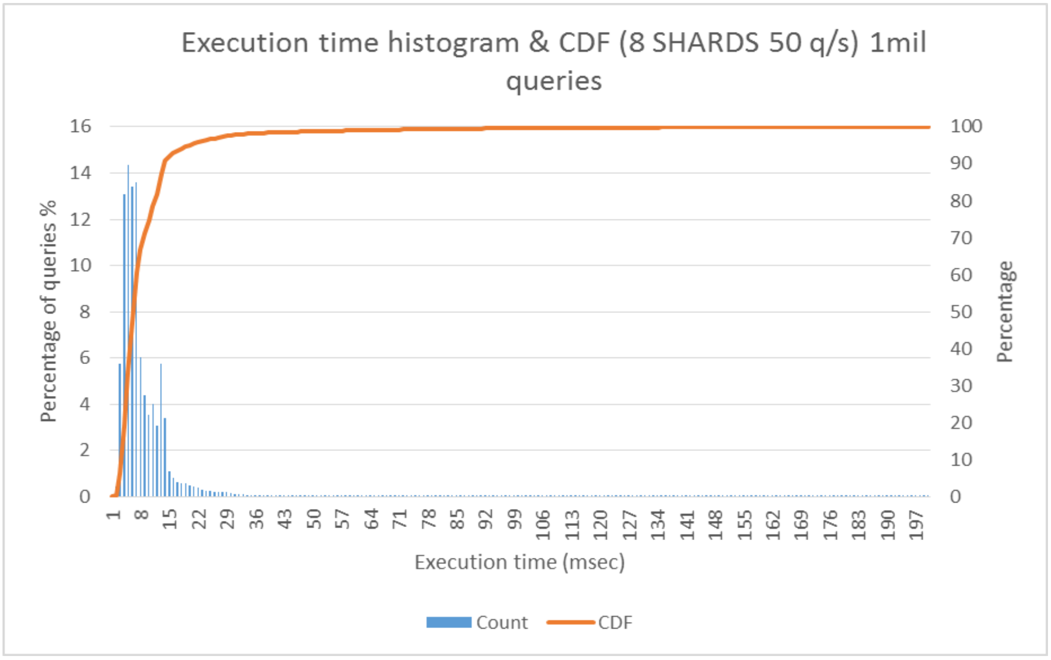

Figure 5. Execution time histogram and CDF of 8 shard configuration at 50 q/s interarrival rate for 1 million queries

This graph shows the distribution of 1 million queries that was serviced by the 8 shards configuration at 50 q/s interarrival rate. As we can observe from the graph, the query execution time follows a right skewed distribution. Query service time distribution follows similar distribution regardless of the sample size, the sample content or the interarrival rate. The majority of queries are serviced in less than 20 milliseconds but the higher quantiles of the distributed are much slower. Same observation about search request distributions are made by previous works. https://cra.org/cra-w/wp-content/uploads/sites/5/2018/04/VUTH-14-combined-slides.compressed.pdf

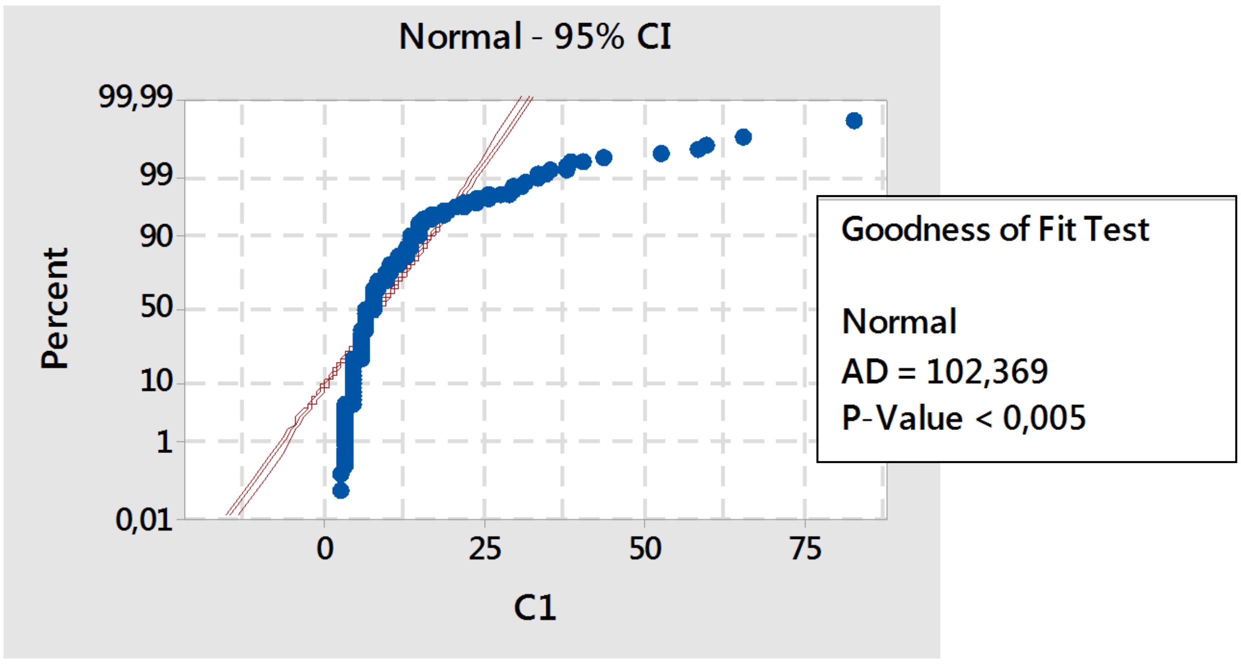

Figure 6. Normality test result

6.1.2 Distribution normality test analysis

This graph shows the results of the Goodness of Fit test of the query service time distribution. The hypothesis error margin is 0, 05 and the P-Value is less than 0,005 so we can be confident that the distribution is not normal. We can also observe the non-normality of the distribution by the blue line in the graph, which does not fit in the red lines.

6.2 Utilization Analysis

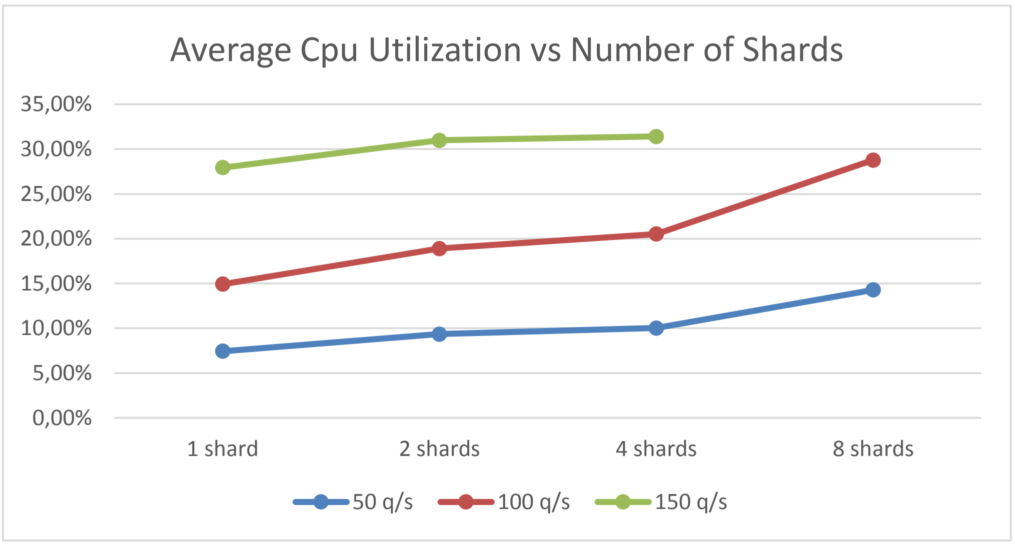

Figure 7. Average CPU Utilization vs Number of Shards for all interarrival rates

This graph shows the average CPU utilization during the experiments. The y axis shows the percentage of utilization. X axis shows the shard configuration. The color denotes the interarrival rate. It can be observed that the higher interarrival rate and the higher number of shards increases the utilization. Note: 8 shard configuration failed to execute at 150 queries per second due to memory limitations of the system.

6.2 Average latency statistical analysis

6.2.1 - 50 Queries per second interarrival rate

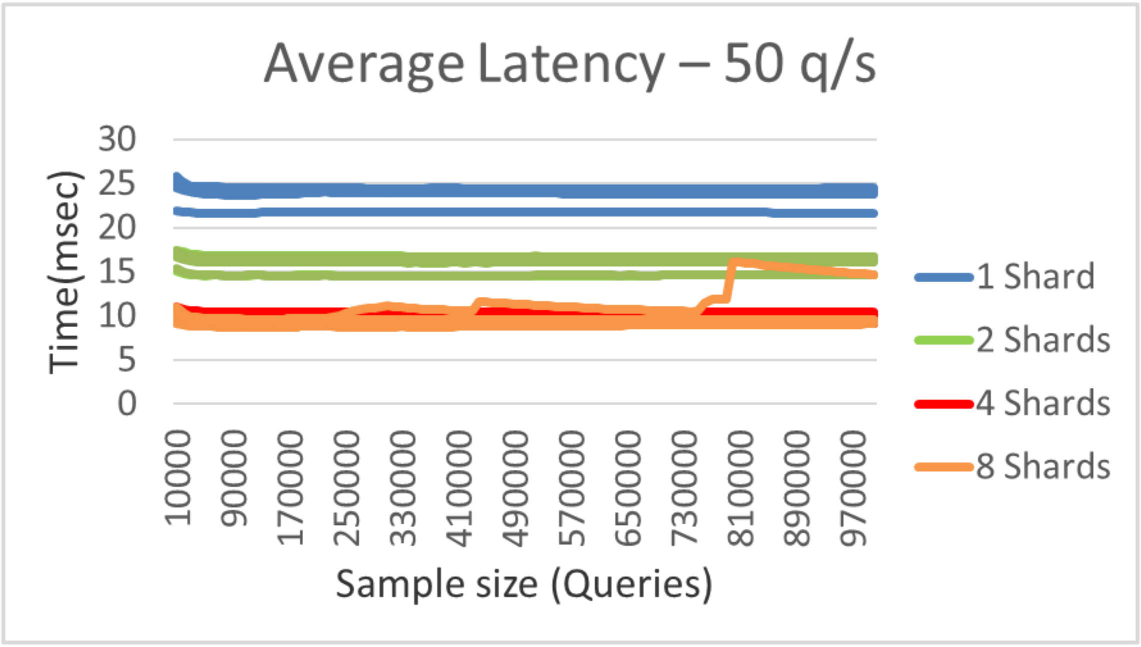

Figure 8. Average Latency at 50 queries per second interarrival rate

Graph shows the average latency of 10 runs of each shard configuration at 50 q/s interarrival rate. The y axis shows the latency in milliseconds. The x axis shows the size of the sample in queries. So, for example, the sample of the initial 10000 queries is a subset of the sample of the initial 50000 queries. It can be observed that the average converges very soon (less than 50 000 queries). The slowest configuration in this graph is the 1 Shard configuration and the fastest is the 8 Shard configuration (if we exclude the outlier). The averages of the runs of the same configuration are close to each other. There is no obvious reason why there are some outliers in the experiments (e.g. the outliers are not measured in the same day and they are not in a specific order). This software is very complex and has a lot of underlying optimization. In addition with the complexity of the hardware and the OS, it makes it common to observe some outliers.

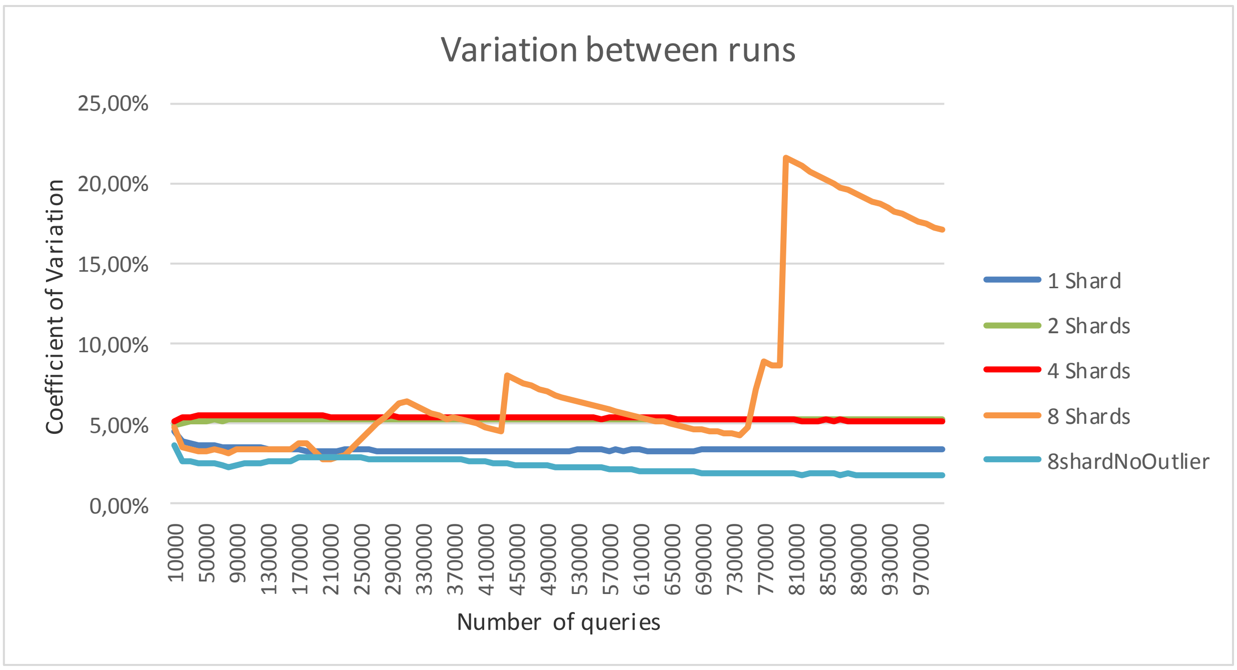

Figure 9. Coefficient of Variation of Average Latency between runs of same shard configuration

This graph shows the variation between the runs of the same shard configuration. Y axis shows the Coefficient of variation in percentage and the x axis the sample size. It also shows the coefficient of variation for the 8 shard configuration, when the outlier is excluded. The variation is steady for all the configurations in all sample sizes, if the 8 shard outlier is excluded. It can be observed that the outlier in 8 shard configuration on the average as shown in figure 8, change significantly the coefficient of variation.

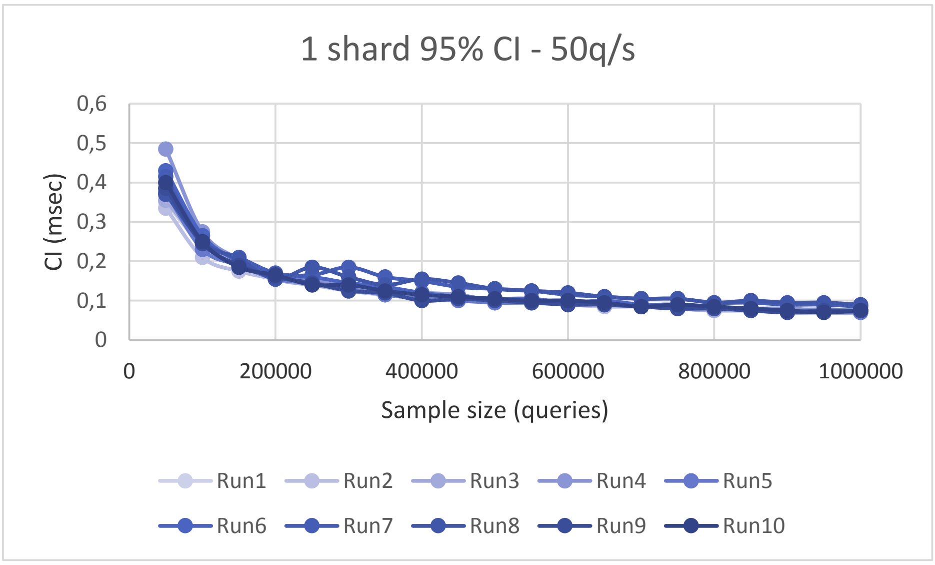

Figure 10. 1 Shard configuration CI at 50 queries per second interarrival rate

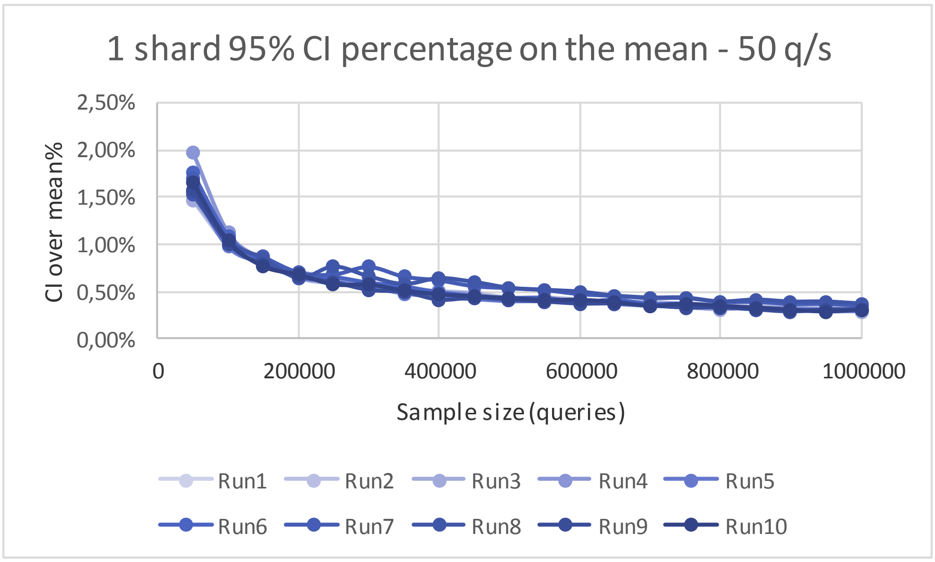

Figure 11. 1 Shard configuration CI over mean at 50 queries per second interarrival rate

This graph shows the Confidence Interval of the average latency of 1 shard configuration. The difference between the graph in figure 10 and the graph in figure 11 is that the graph in figure 10 shows the actual value of the confidence interval of the average, and the second graph shows the percentage of the confidence interval value over the average latency. The y axis shows the confidence interval in milliseconds (1st graph) and in percentage (2nd graph). The confidence intervals are one way, which means that the value shown is +- on the respective average latency value that is shown on the first graph. The x axis shows the sample size in queries. It can be observed that the significant drop of confidence interval (2% to 1%) happens on the first 200 000 queries and then it drops slowly. From 200 000 sample size to 1000000 sample size the confidence interval drops from 0, 7% to 0, 4%.

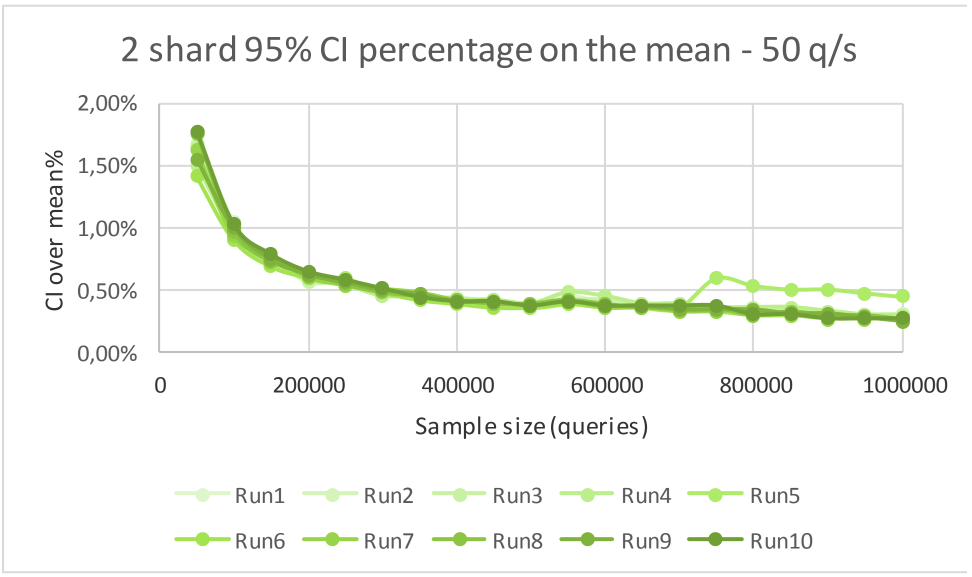

Figure 12. 2 Shard configuration CI over mean at 50 queries per second interarrival rate

This graph follows the same concept with the previous graphs but for 2 shards configuration. It can be observed that the confidence interval drops lower that 1% in less than 200 000 queries.

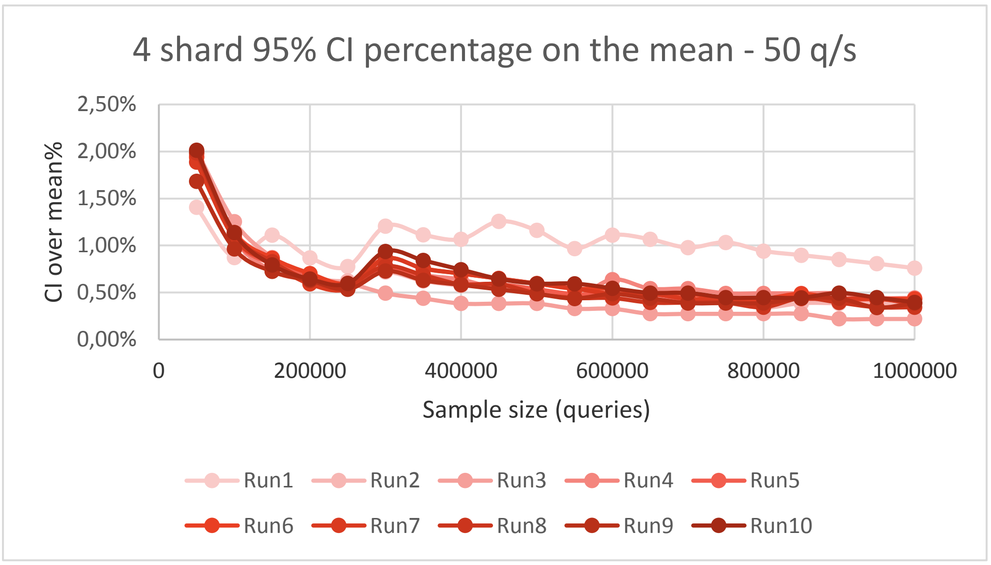

Figure 13. 4 Shard configuration CI over mean at 50 queries per second interarrival rate

It can be observed that the significant drop of confidence interval (2% to 1%) happens on the first 100 000 queries. At 250 000 queries the confidence interval is increasing.

The reason is that Java Garbage Collector triggers a full Garbage collection, which is a heavy operation, and consumes the system recourses. Some full garbage collection is triggered at the start of the server so the experiment is not interrupted. It is more likely for a Full garbage collection to be triggered in a configuration with multiple shards because there are more threads that serve requests and more threads means more memory space needed.

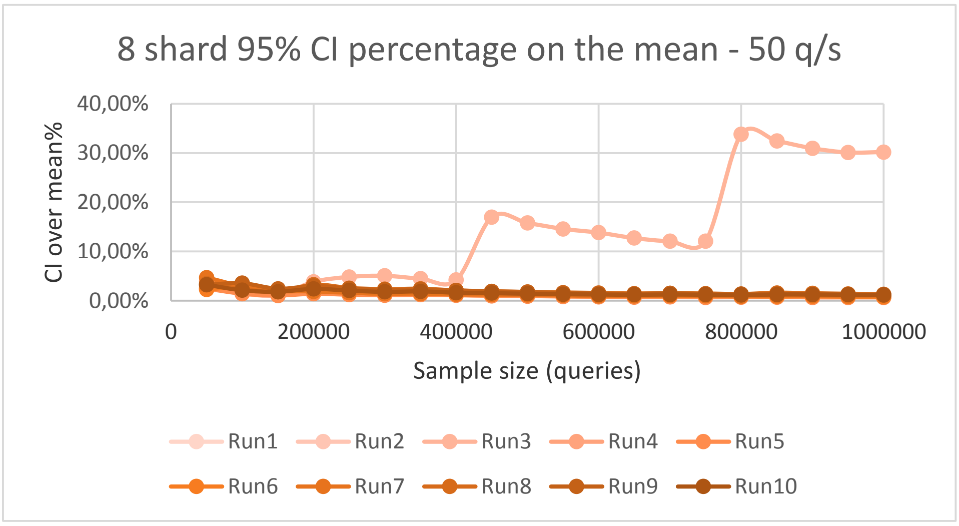

Figure 14. 8 Shard configuration CI over mean at 50 queries per second interarrival rate

This graph shows the CI over mean of 8 shard configuration. Outlier has significantly higher CI than the rest and has the same shape as the actual value of the average of outlier in figure 8.

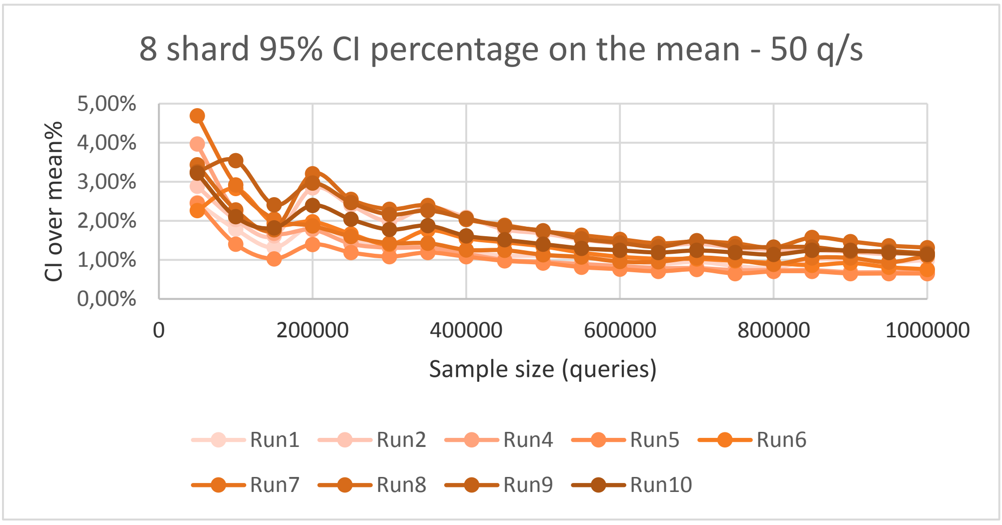

Figure 15. 8 Shard configuration CI over mean at 50 queries per second interarrival rate – no outlier

This graph shows the CI over mean for 8 shard configuration but without the outlier Even if the outlier is excluded the 8 shard configuration does not converge under 1%. There is an instability in CI in the first 200 000 queries and after 400 00 it converges to under 2%.

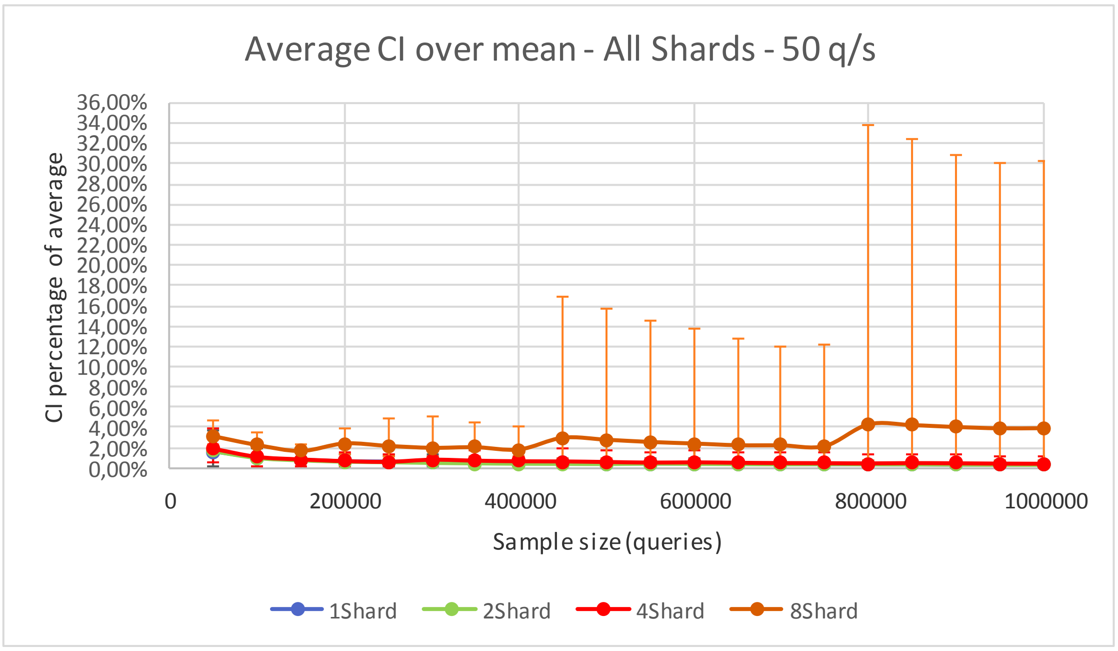

Figure 16. All configurations average CI over mean at 50 queries per second interarrival rate

This graph shows the average CI of every shard configuration, which means the average of the 10 runs' CIs for every different shard configuration. The error bars shows the minimum and the maximum value of the CIs that are represented by the average.

1, 2 and 4 shard configuration converges at the same percentage, which is under 1% at the first 100 000 queries.

6.2.2 – 100 queries per second interarrival rate

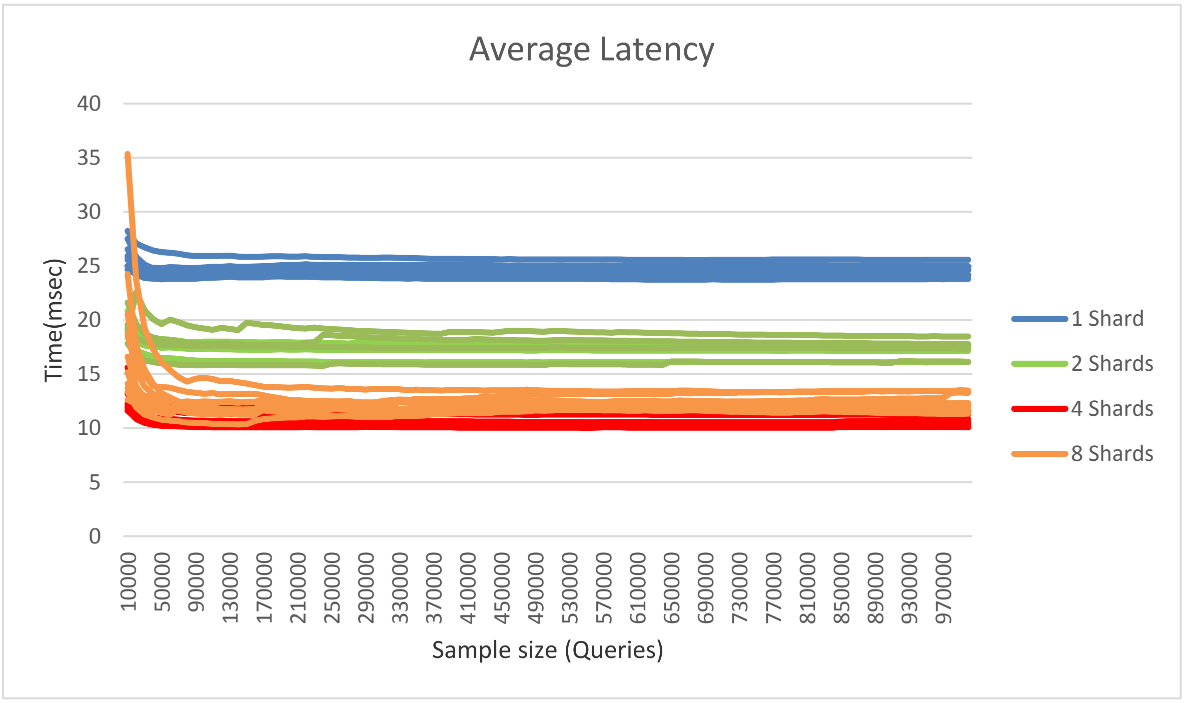

Figure 17. Average Latency at 100 queries per second interarrival rate

Graph shows the average latency of 10 runs of each shard configuration at 100 q/s interarrival rate. 1 shard configuration has the highest latencies. 4 shard has the lowest latencies. 8 shard configuration average starts in a very high value of average and it drops fast in the first 10 000 – 90 000 queries.

The averages of each configuration becomes steady at an early point, at less than 100 000 queries. The averages between the same configurations do not converge to the same value.

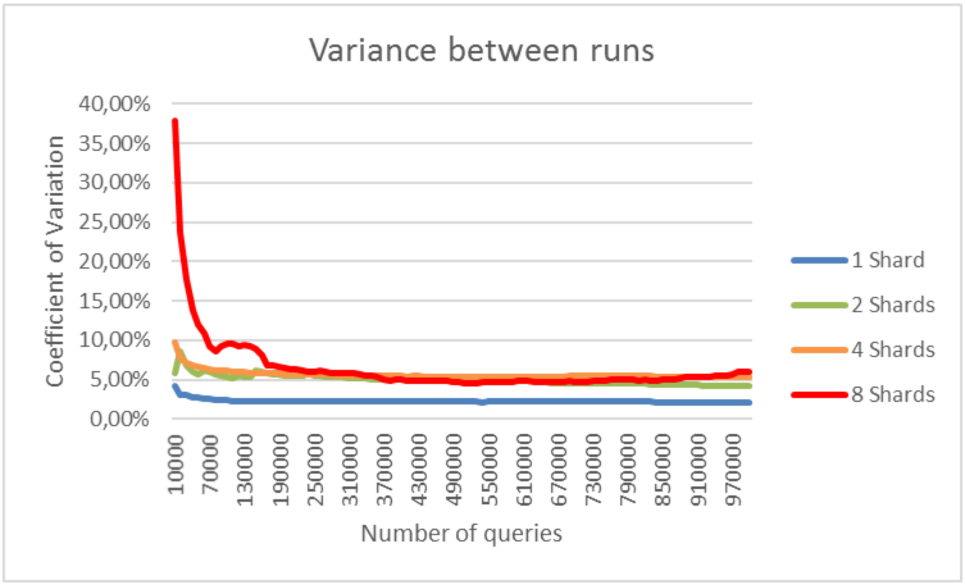

Figure 18. Coefficient of Variation of Average Latency between runs of same shard configuration at 50 queries per second interarrival rate

This graph shows the variation between the runs of the same shard configuration at 100 queries per second interarrival rate.

After 190 000 queries the coefficient of variation converges on 5% for 2, 4 and 8 shard configurations. Variation between runs of 1 shard configuration converges under 5%.

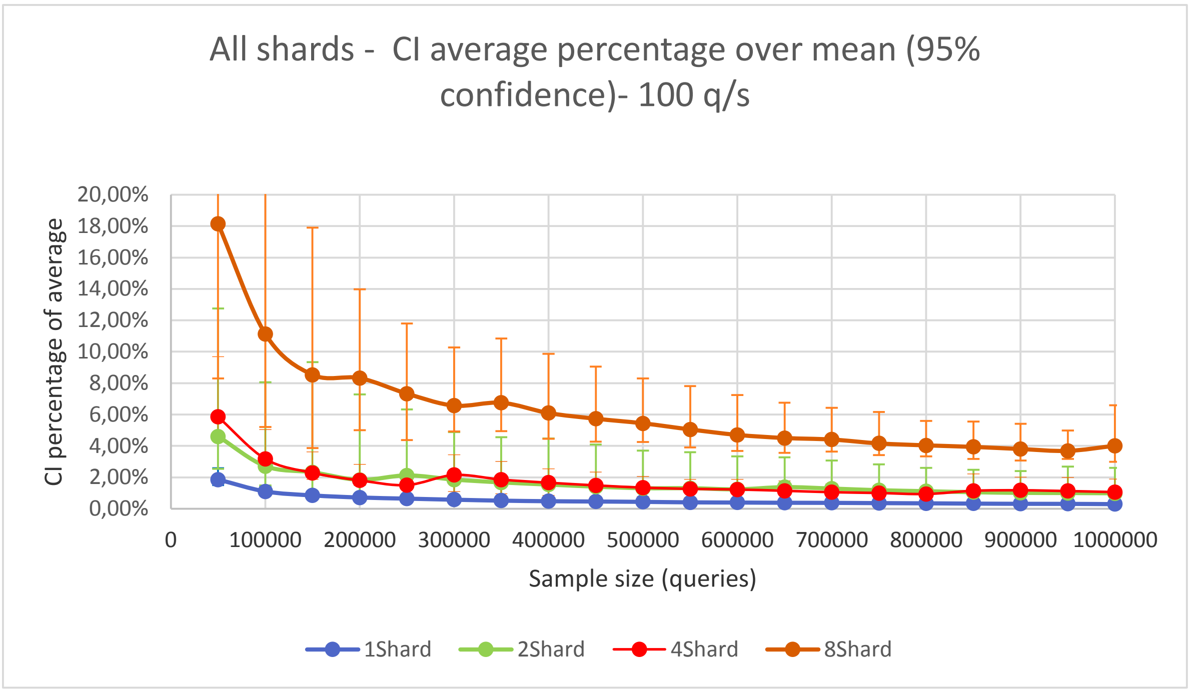

Figure 19. All configurations average CI over mean at 100 queries per second interarrival rate

Significant difference in convergence of CI. 1 shard stable. 2 and 4 shard less affected by the interarrival but 8shard affected significantly.

6.2.3 - 150 Queries per second interarrival rate

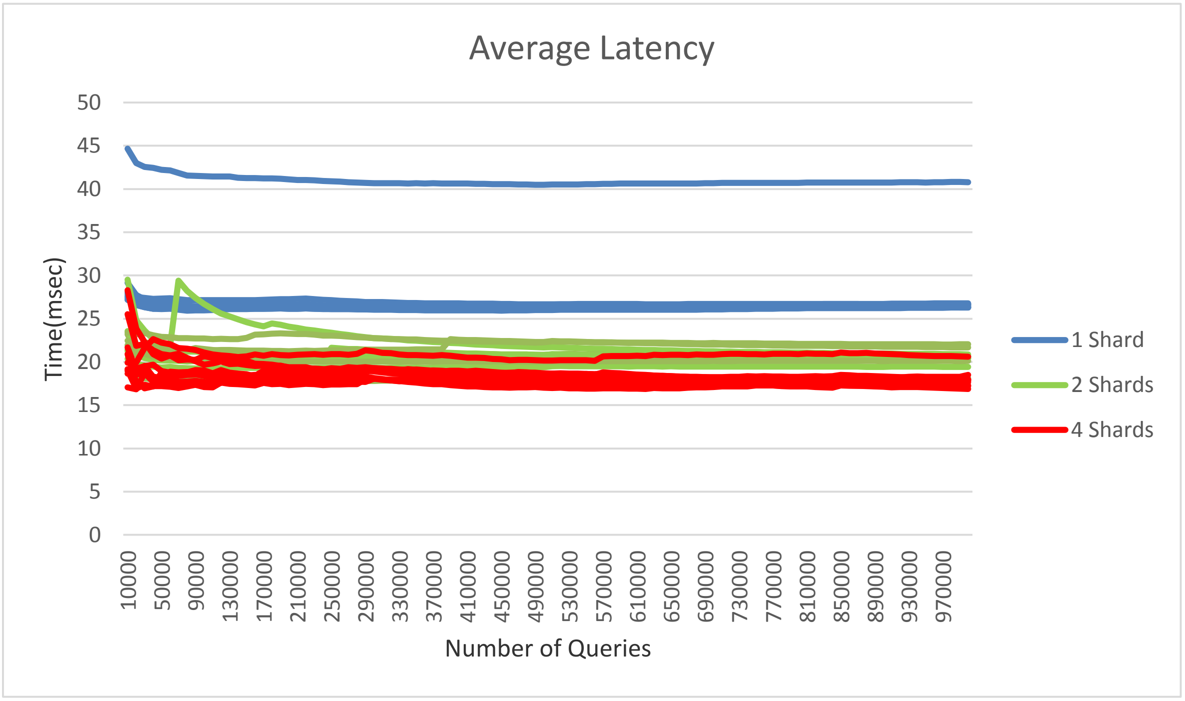

Figure 20. Average Latency at 150 queries per second interarrival rate

Graph shows the average latency of 10 runs of each shard configuration at 150 q/s interarrival rate. 8 shard failed to execute due to the limitations of the system. Variability between runs. Each run seems to converge early but differ from other runs. Less difference between configurations' average latency. Overlap of 4 and 2 number of shards configurations.

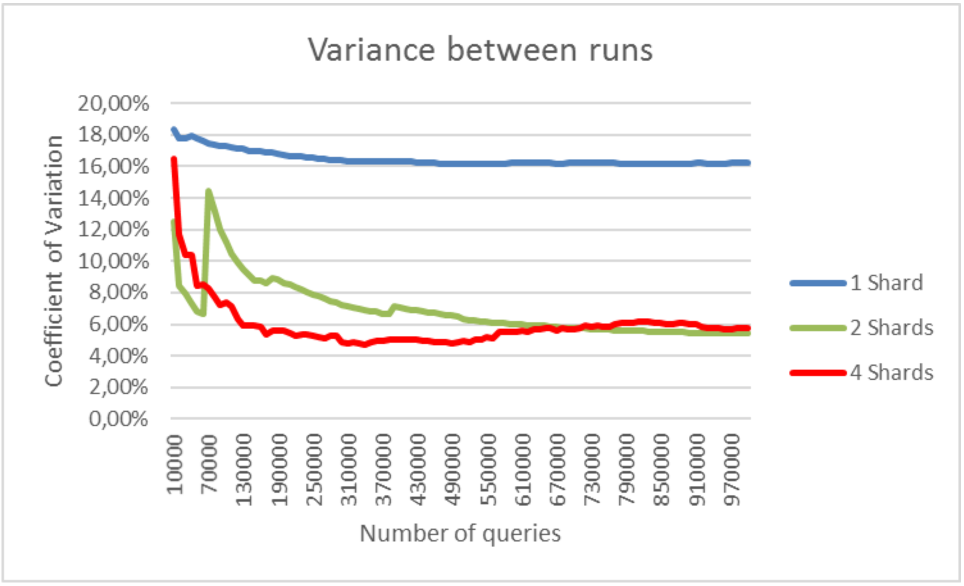

Figure 21. Coefficient of Variation of Average Latency between runs of same shard configuration at 150 queries per second interarrival rate

1 shard configuration has 18% variation due to the outlier run. The 2 and 4 shard configurations have less variance. The variance convergence after 650,000 queries to 6% in contrast with lower interarrival rates where the variance converges to less than 6% in less than 150,000 queries.

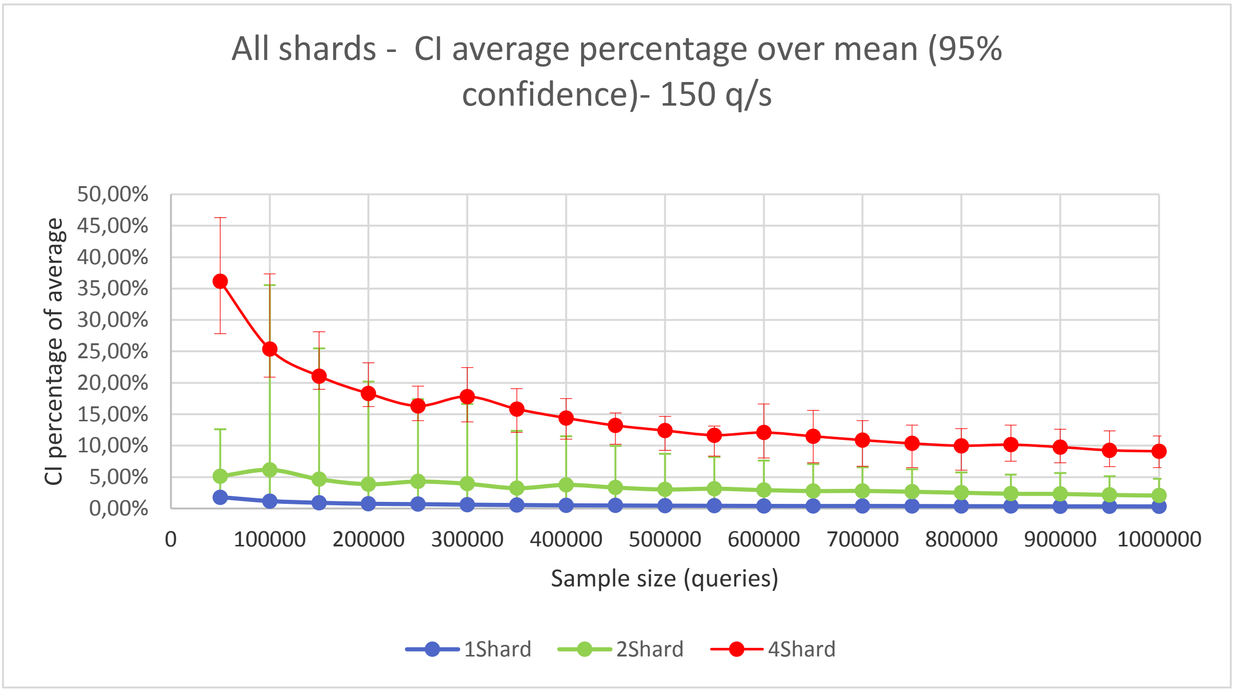

Figure 22. All configurations average CI over mean at 150 queries per second interarrival rate

4 shard configuration is the most affected by the increase of the interarrival rate. Does not converge under 10%. 2 shard configuration is less affected by the interarrival rate. Converges under 4 %. 1 shard configuration converges under 2%.

6.3 Tail latency statistical analysis

6.3.1 – 50 queries per second interarrival rate

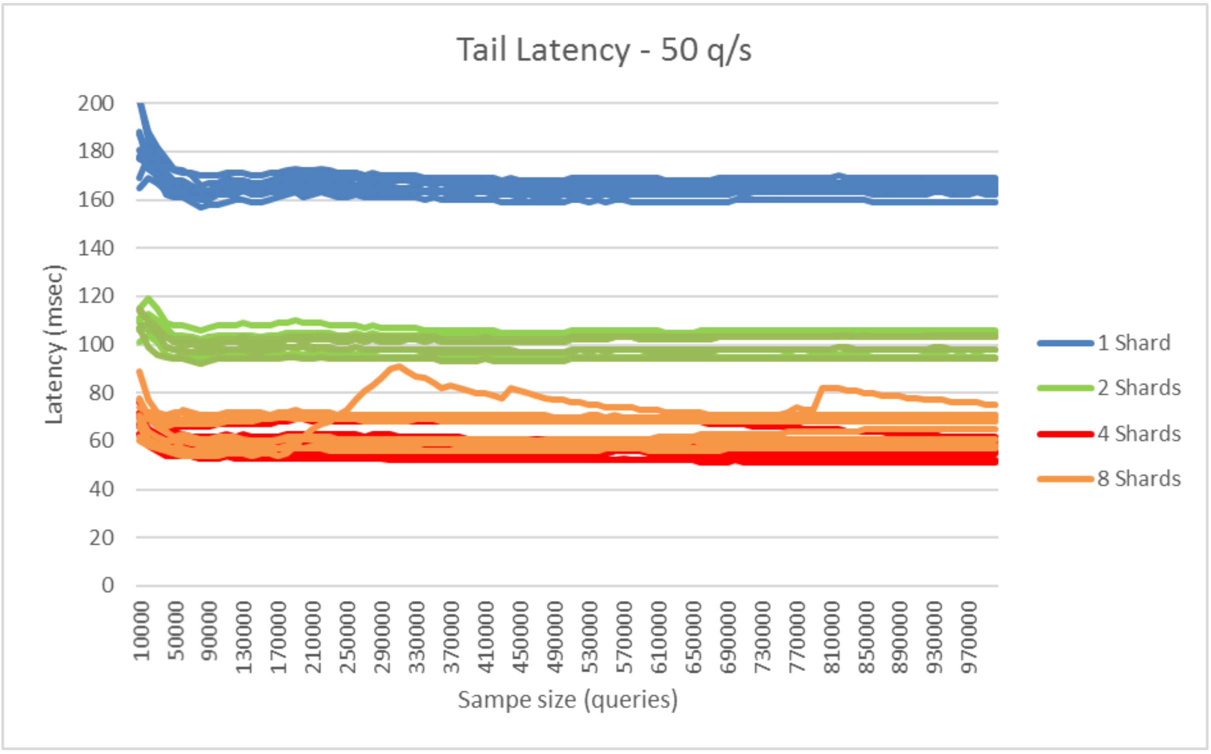

Figure 23. Average Latency at 50 queries per second interarrival rate

We can observe that outlier affecting less the tail on 8 shard configuration. This is because there are high latencies, significantly higher than the tail that changes the average but does not affect the tail. 1 shard is the slowest configuration, 4 shard the fastest. Significant drop in tail latency from 1 shard configuration to 2 shard configuration and from 2 shard to 4 shard.

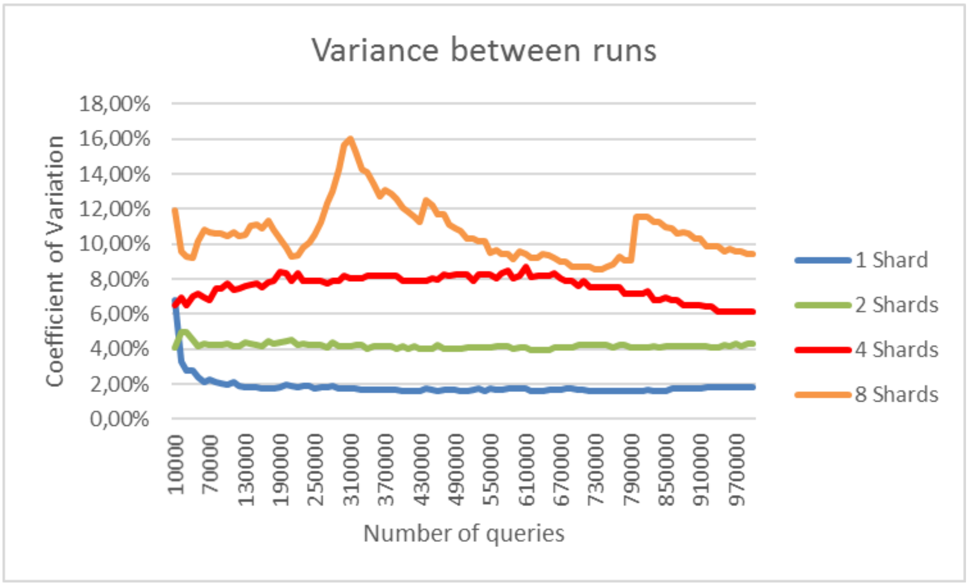

Figure 24. Coefficient of Variation of Tail latency between runs of same shard configuration at 50 queries per second interarrival rate

Variance in runs is higher when number of shards is high. 8 shard variation between runs is very unstable, in contrast to the rest of the configurations where the stability is more. Although the rest of the configuration are steady on the variation between the runs, the percentages are high enough to show that multiple runs are needed in order to be precise.

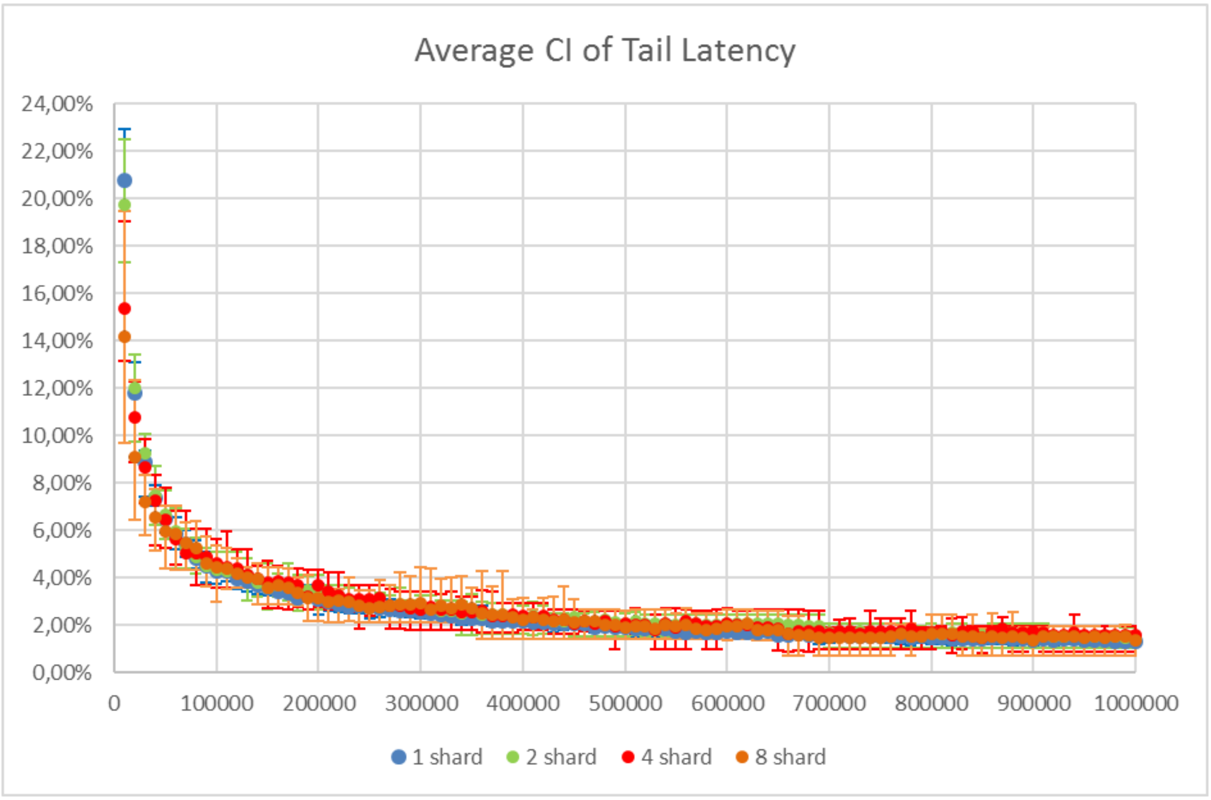

Figure 25. All configurations average CI over Tail at 50 queries per second interarrival rate

In contrast with figure 24, the confidence intervals are the almost the same for every configuration in all the sample sizes. Also we can observe that at 100,000 it converges to 4 -6% in contrast with figure 16 where the average converges at less than 1%, at the same interarrival rate. We also do not see any outliers.

6.3.2 – 100 queries per second interarrival rate

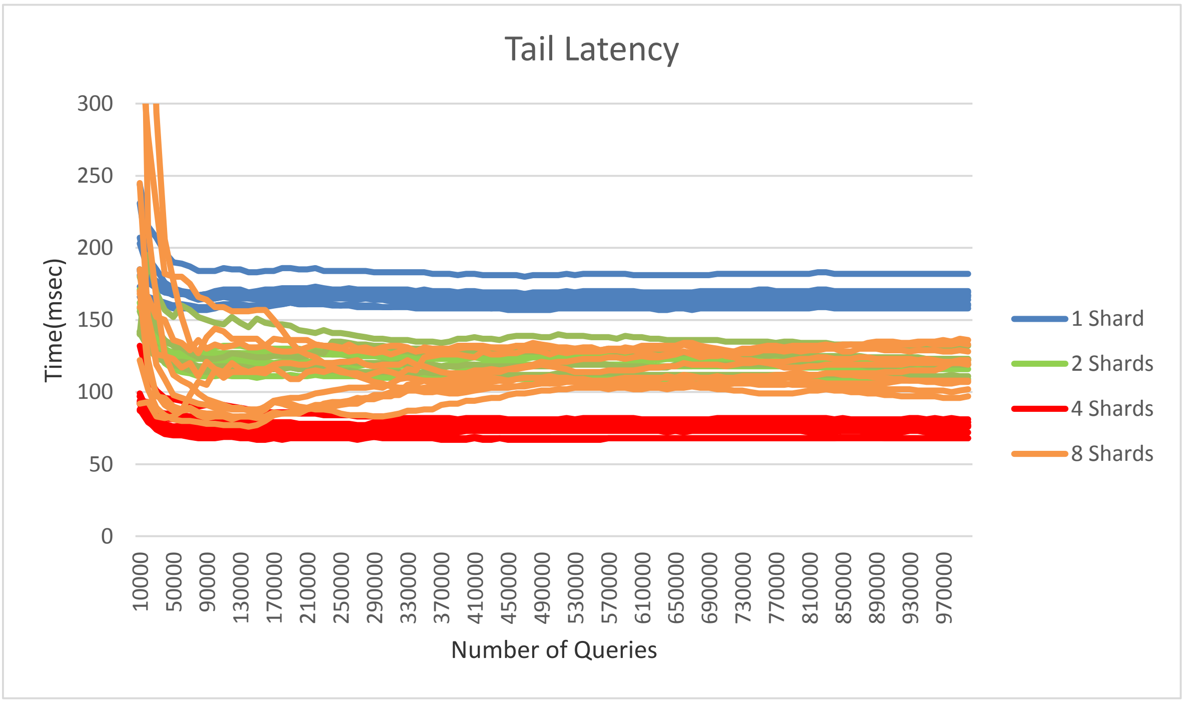

Figure 26. Tail Latency at 100 queries per second interarrival rate

Tail latency of 8 shards increases and shows that excessive sharding at this level is detrimental to the performance of the search server. The rest of the shard configuration are affected. If we compare the tail latency with the 50 q/s interarrival (figure 23) we observe that the tail has less speed up when the shards are more.

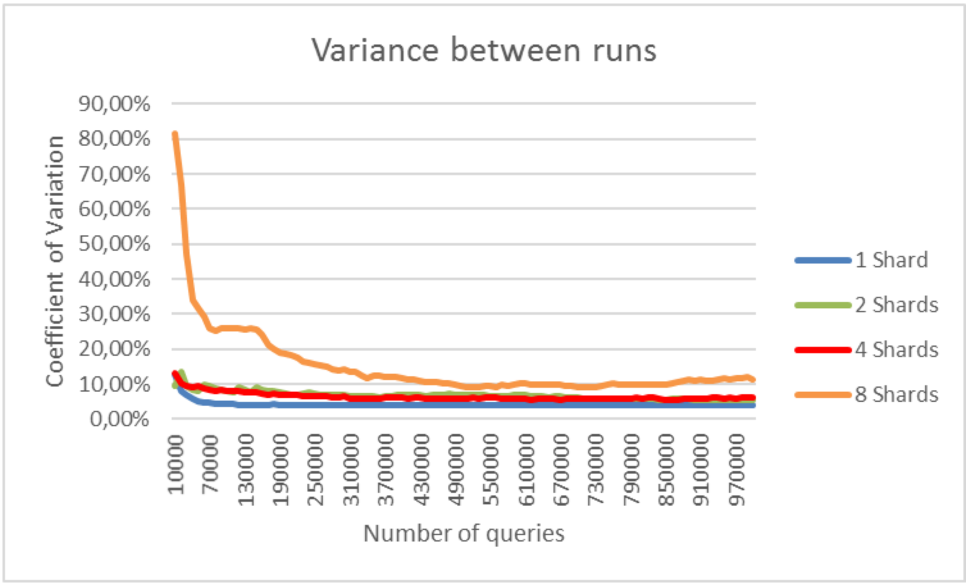

Figure 27. Coefficient of Variation of Tail Latency between runs of same shard configuration at 100 queries per second interarrival rate

High variation in the first 500,000 queries of 8 shards. Rest of configurations converge in 5-10% which shows that a conclusion by a single experiment may result in imprecise conclusion.

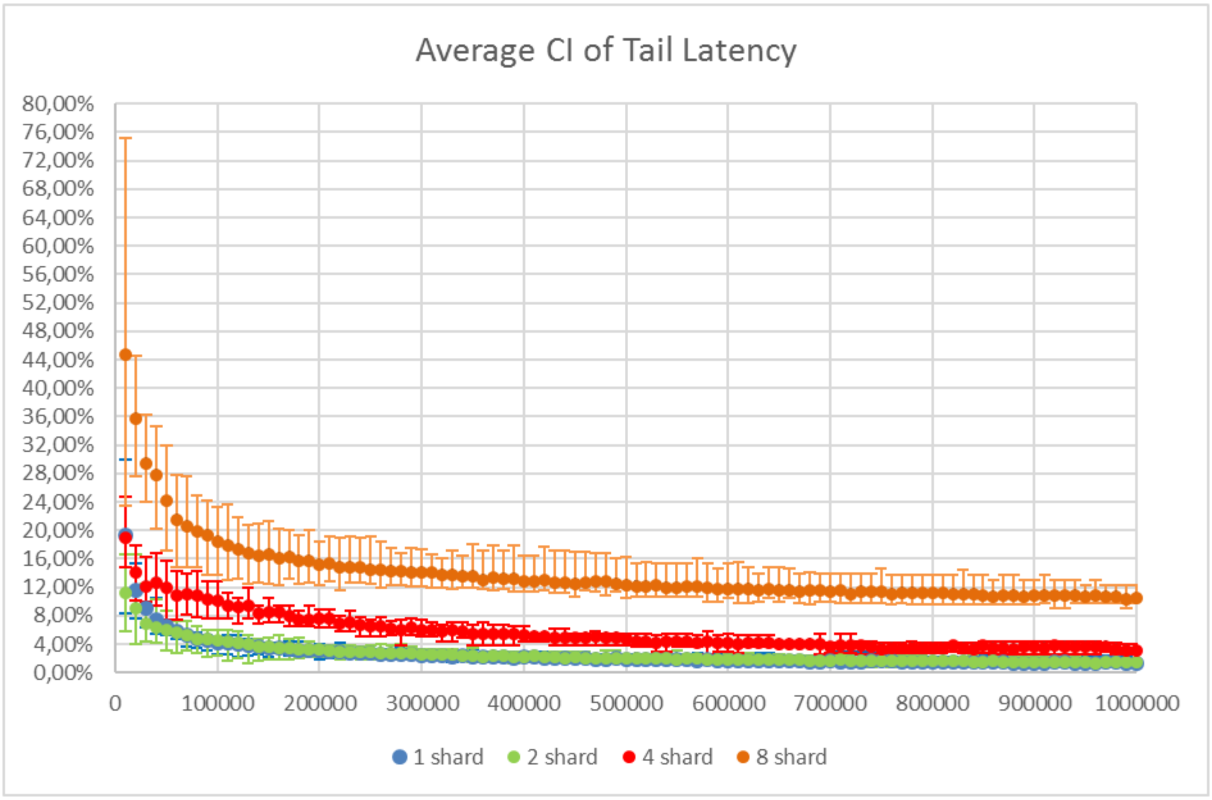

Figure 28. All configurations average CI over Tail at 100 queries per second interarrival rate

1 and 2 shard configuration remain on 4% at 100,000 queries.

4 shard configuration is affected by the increase of the interarrival rate. At 100,000 queries the CI over mean is 10% which is double the CI of the 50 q/s interarrival rate on figure 24.

8 shard configuration is affected significantly by the increase of the interarrival rate. It starts with 75% CI over mean and then converges to 10% on more than 500,000 queries which means that there is high imprecision on the measurement.

6.3.3 – 150 queries per second interarrival rate

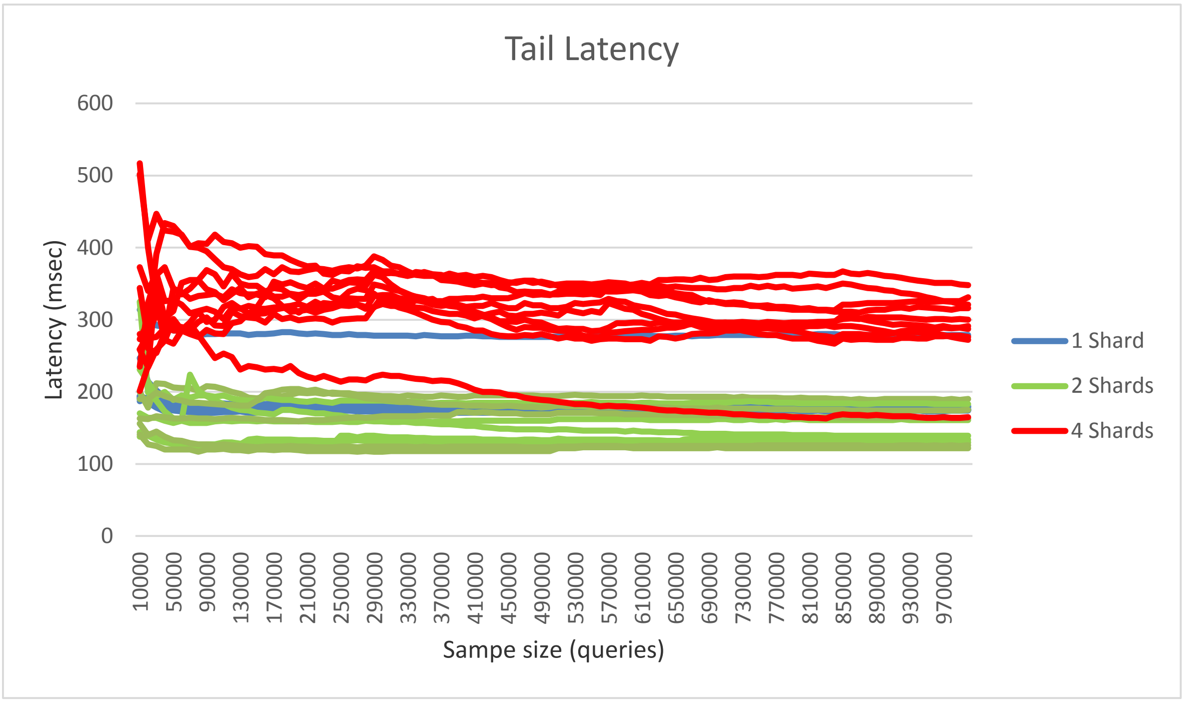

Figure 29. Tail Latency at 150 queries per second interarrival rate

4 shard configuration is the most affected the increase of interarrival rate. It not beneficial for the latency on 150 q/s as its tail is higher than the tail of 1 shard configuration. It does not seem to converge to a specific value regardless of the sample size.

1 shard is steady on 180ms-200ms and it is not affected by the interarrival rate except from the outlier that is shown. Another point on running more than once the same experiment.

2 shard configuration almost overlaps with 1 shard. The sharding benefit is almost not existent.

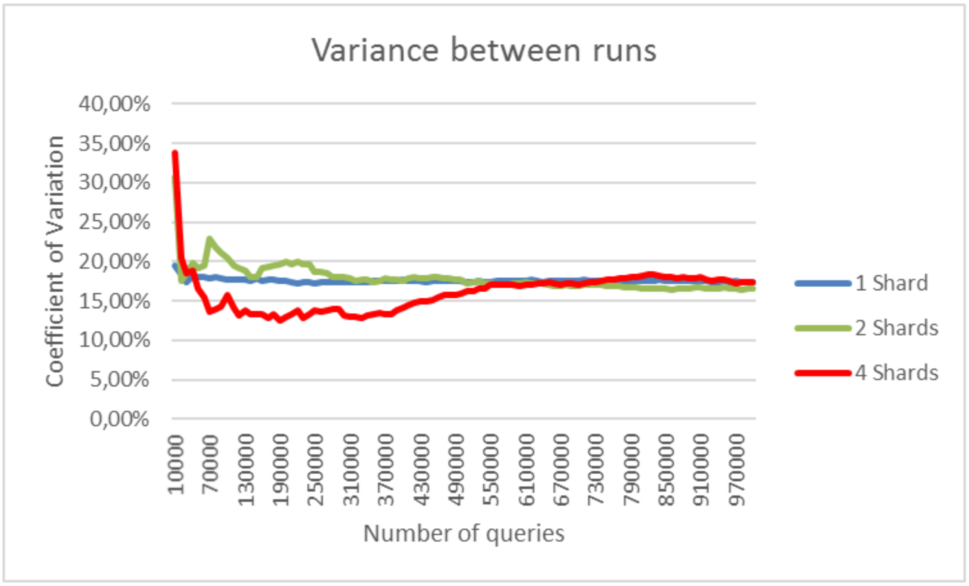

Figure 30. Coefficient of Variation of Tail Latency between runs of same shard configuration at 150 queries per second interarrival rate

Coefficient of variation for all the configurations is higher than 10%.

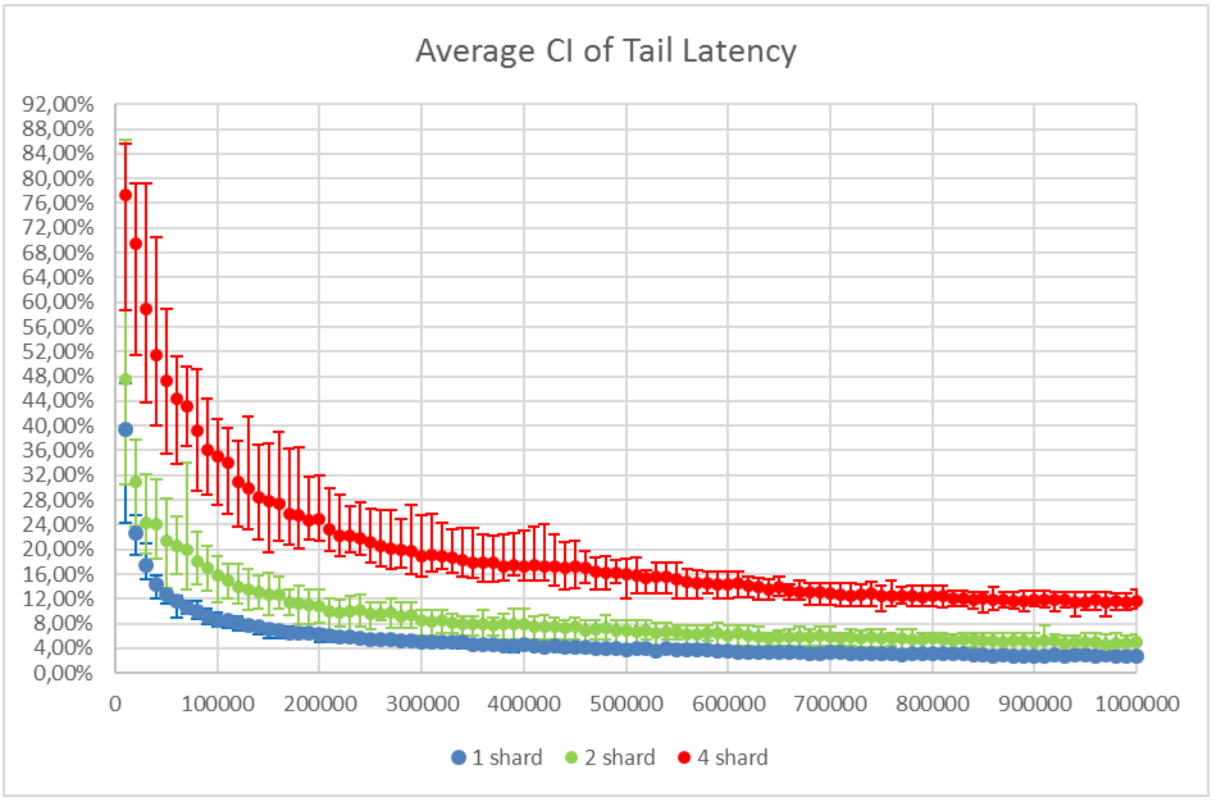

Figure 31. All configurations average CI over Tail at 150 queries per second interarrival rate

1 shard from 4% CI at 100,000 on 50qps and 100 qps interarrival rate (figure 25 & figure 28) went to 8%.

2 Shard configuration is affected more, and went to 16% at 100,000 queries. 4 shard configuration is significantly affected and the CI at 100,000 queries is at 36%.

Chapter 7

Conclusion

7.1 Server Characterization conclusions

Index sharding is beneficial for both average and tail latency on low utilization but it is less beneficial or even detrimental on high traffic. More specifically, the higher the number of shards, the more beneficial it will be on low traffic and the more detrimental it will be on high traffic.

This can be observed on figures of average and tail where the 8 shard have the lowest latencies in low traffic and the worst on high traffic. It cannot even run on 150 qps interarrival rate and that is showing the instability of 8 shard configuration on high traffic.

4 shard configuration benefit is observed to be less while the traffic gets higher and at the 150 qps, it has the higher latencies.

The theory behind this behavior is that the more workers serve a request, the less likely a worker will delay the whole request. Because the rest of workers have to wait for everyone to finish, in order to have complete response, the variability in latencies increases and also queueing effect becomes more intense because more resources are assigned to a single request and the throughput is low.

7.2 Statistical analysis conclusions

For every configuration on all traffic levels we show the precision (in CI width over parameter percentage) of the measurement on an increasing range of sample size. We give an expectation of the level of precision a search server experiment with this setup can give. We indicate where is worth having large sample and where the experiment can be precise with smaller sample. We show how the precision of experiments is affected by the interarrival rate. In other words, we show that configurations with high number of shards are affected more than configurations with low number of shards on the width of confidence interval and therefore on the precision. Even with high sample size, the CI percentage of high number of shards configurations converges to a much higher value than the CI percentage of lower number of shards configurations.

For every parameter we give an error range (a confidence interval with 95% statistical confidence) in order to guarantee QoS. We show precision vs size of experiment to give insight to where to draw the line between cost of experiment and precision of experiment. Furthermore, outliers are observed in the experiments and that suggests multiple runs per experiment.

Recommended experiment sizes for measuring Tail Latency*.

#ofShards/Interarrival	**50 qps**	**100 qps**	**150 qps**
1	**2/100,000 (+-5%)**	**2/100,000 (+-5%)**	**3/300,000(+-4%)**
2	**2/100,000 (+-5%)**	**3/100,000 (+-4%)**	**7/300,000(+-10%)**
4	**2/100,000 (+-5%)**	**3/300,000 (+-6%)**	**7/500,000(+-16%)**
8	**2/100,000 (+-5%)**	**5/300,000(+-14%)**	**-**

***Interpretation**: x/y (+-a%) means that recommended experiment size is x runs of y queries. The error margin will be +-a% on the value of the Tail Latency.

7.3 Limitations

Limitations of this work are the failed experiment of 8 shards configuration at 150 q/s and the garbage collection that caused outliers and it needs tuning. The cluster that the experiments were running had a few technical issues and needed hard reset. We do not know whether the technical issues affected the experiments.

7.4 Future Work

As a future work the same analysis can be made on different benchmarks. Also, the evaluation and the improvement of limitations. The analysis can be made with more interarrival times or different system configuration such as SMT disabled or Turboboost disabled. Furthermore, analysis of more percentiles with different confidence levels can be conducted.

References

[1] http://www.eecs.umich.edu/eecs/about/articles/2016/treadmill.pdf

[2] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7095818&tag=1

[3] https://cra.org/cra-w/wp-content/uploads/sites/5/2018/04/VUTH-14-combined-slides.compressed.pdf

[4] http://lucene.apache.org/solr/#intro

[5] https://lucene.apache.org/core/3_0_3/fileformats.html

[6] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2545906/pdf/bmj00286-0037.pdf

[7] https://towardsdatascience.com/an-introduction-to-the-bootstrap-method-58bcb51b4d60

[8] https://lucidworks.com/2011/03/10/solr-relevancy-function-queries/

[9] https://www.wisdomjobs.com/e-university/research-methodology-tutorial-355/sample-size-and-its-determination-11520.html

[10] https://lucene.apache.org/solr/guide/6_6/distributed-search-with-index-sharding.html

[11] https://lucene.apache.org/solr/guide/7_4/

[12] https://support.minitab.com/en-us/minitab-express/1/help-and-how-to/basic-statistics/summary-statistics/normality-test/interpret-the-results/key-results/

[13] https://googleblog.blogspot.com/2010/09/google-instant-behind-scenes.html.

[14] https://jeffhuang.com/search_query_logs.html