Scaling Out For Extreme Scale Corpus Data

Matthew Coole, Paul Rayson and John Mariani
School of Computing and Communications
Lancaster University
Lancaster, Lancashire, UK
m.coole@lancaster.ac.uk, p.rayson@lancaster.ac.uk, j.mariani@lancaster.ac.uk

***Abstract*—Much of the previous work in Big Data has focussed on numerical sources of information. However, with the 'narrative turn' in many disciplines gathering pace and commercial organisations beginning to realise the value of their textual assets, natural language data is fast catching up as an exploitable source of information for decision making. With vast quantities of unstructured textual data on the web, in social media, and in newly digitised historical document archives, the 5Vs (Volume, Velocity, Variety, Value and Veracity) apply equally well, if not more so, to big textual data.**

Corpus linguistics, the computer-aided study of large collections of naturally occurring language data, has been dealing with big data for fifty years. Corpus linguistics methods impose complex requirements on the retrieval, annotation and analysis of text in terms of displaying narrow contexts for each occurrence of a word or linguistic feature being studied and counting co-occurrences with other words or features to determine significant patterns in language. This, coupled with the distribution of language features in accordance with Zipf's Law, poses complex challenges for data models and corpus software dealing with extreme scale language data. A related issue is the non-random nature of language and the 'burstiness' of word occurrences, or what we might put in Big Data terms as a sixth 'V' called Viscosity.

We report experiments to examine and compare the capabilities of two No-SQL databases in clustered configurations for the indexing, retrieval and analysis of billion-word corpora, since this size is the current state-of-the-art in corpus linguistics. We find that modern DBMSs (Database Management Systems) are capable of handling this extreme scale corpus data set for simple queries but are limited when querying for more frequent words or more complex queries.

***Keywords*-corpora, scaling, No-SQL, linguistics, distribution, corpus data**

I. Introduction

Corpora and corpus linguistic techniques are utilised for many purposes. From lexicography and sociolinguistic studies to language learning and cyber-security, the applications of corpora are vast. This motivates the need for corpus methods and software that can allow researchers to construct, query and analyse these corpora. Such tools will inevitably revolve around a database used to store and query collections of texts. Historically, DBMSs (Database Management Systems) have relied on SQL (Structured Query Language) as an interface. However recent developments in computing driven by the Big Data trend have given rise to a new wave of databases collectively known as No-SQL. With the scale of corpora of interest to corpus linguists moving from millions to billions of words, traditional SQL DBMSs can no longer meet the needs of such extreme scale corpora. The DBMSs encompassed by No-SQL, many of which were developed with big data in mind, provide a promising alternative to build upon. Presented here is an exploration and comparison of the capabilities of two No-SQL DBMSs as applied to corpus storage and retrieval. We also examine how performance is affected by scaling out. The paper concludes with an outline of further work to analyse how these systems work with a view to designing a truly scalable modern corpus database solution.

Corpus linguistics methods first came to prominence in the 1960s as a means of quantitatively analysing natural language. One of the very early examples is the Brown Corpus [1]. This consisted of a collection of written text from American English in a wide array of categories such as novels, newspaper articles, religious texts etc. The word count for the corpus totalled 1,014,000. By today's standards a 1 million record database would not necessarily seem too large. However the size of corpora did not remain static. In the early 1990s the British National Corpus (BNC) [2] was released containing 100 million words. As well as the raw text, the BNC also contained further annotation of the text such as lemmas and part-of-speech tags for each word, further expanding the storage needs and scope for complex queries required by corpora. This growth in the size of corpora has continued into the 21st century with modern corpora now in the order of billions and 10s of billions of words. The Hansard corpus, a transcription of parliamentary speeches the British House of Commons and Lords, consists of over one billion words in seven million files and is an example of the extreme scale found in many corpora used today. Early English Books Online (EEBO) is another example of a corpus built from sets of historical texts that takes the word count into the billions. Although many corpus linguists also make use of much smaller specialist corpora the need for database systems to store and query corpora at extreme scale is becoming ever more important. In addition to the size issue, corpus linguistics methods such as concordances, frequency lists, keywords, n-grams and collocations have very specific requirements that are typically not well served by the relational model, so

reconsidering standard models is very timely.

Modern database systems in recent years have followed two general rules for scaling - scale vertically (scale up) or scale horizontally (scale out) [3]. Scaling vertically is perhaps looked on as the simpler solution – if your database server needs to handle more data, more client requests per second etc. simply migrate your database to a more powerful server with more memory, storage, CPU-power etc. Of course scaling up is limited by the hardware that is available - if you have the biggest possible server with fastest CPU, and cannot fit in any more memory or disk storage you have reached the limit of vertical scaling. This is why many companies operating large distributed databases today (such as Google, Wikipedia and Amazon) have chosen to adopt horizontal scaling by adding more servers to distribute the work load. This scaling out has been an integral part of the big data trend. Most recent distributions of popular relational Database Management Systems (DBMSs) such as MySQL and Oracle now have the facility to scale out. Many No-SQL DBMSs such as MongoDB, Cassandra etc. were in fact designed from inception with scaling out in mind. It is clear that any corpus retrieval software system will need to be able to scale out if it is to handle the huge amounts of data that are found in modern corpora.

The remaining sections of this paper outline an initial study into how two modern DBMSs can handle corpus data when scaled out. The background examines the kind of queries that corpus linguists typically use when working with corpora and therefore the types of queries any DBMS used for storing corpora will need to handle. The setup sections describes how the two DBMSs were deployed and the mechanisms used to query them. This section also discusses how problems due to the difference in the DBMSs were overcome in order to conduct a valid and fair comparison between them. The results sections presents a summary of the data found during testing and analyses the differences between the two DBMSs as well as the benefits each one achieved through scaling out. The conclusion discusses the work in the context of corpus linguistics as well as big data and outlines the implications this study has for further work in the field of developing a scalable corpus database.

II. Background

Linguists utilise various different corpus techniques to analyse the language in a given corpus. One of the most common is a concordance analysis. A concordance analysis tends to focus on a keyword or phrase and a set of concordance lines are built from the words that occur immediately prior and following the keyword. This is commonly know as a keyword in context (KWIC) search. Corpus linguists can utilise this technique to look for typical patterns of usage for a word they are interested in such as in a study conducted on the portrayal of Muslims in the British Press [4]. Of course to draw any meaningful conclusions from such an analysis it is necessary to build a sufficient number of concordance lines and support a range of sorting operations on the centre word or the immediate context to the left and/or right. For very common words a relatively small corpus will usually be sufficient to generate enough examples but as the keywords, phrases and other higher-level patterns or features that the linguist is interested in examining become more uncommon it is necessary to construct larger and larger corpora in order to build a sufficient number of concordance lines for any analysis to be valid. This is a major factor driving the increase in corpora size.

When looking at corpora of ever increasing size certain patterns in the data hold. Zipf's law when applied to corpora describes how the nth most frequent word's frequency is roughly proportional to 1/n [5]. As a result many corpora follow the pattern of the most frequent words (typically "the", "a", "of" etc.) accounting for a large proportion of the total words in the corpus. For example the word "the" accounts for approximately 6% of all words in the BNC according to Rayson et al. [6]. Zipf's law further shows that typically half the words occurring in a corpus will occur only once. For any database looking to store and query a large corpus this means when searching for most words there will in fact be very few results to retrieve and only with the most common words will there be an extremely large set of results to retrieve, compile and sort.

Many existing corpus tools exist for querying corpora. AntConc [7] is one of the most popular that allows users to query the corpora from the raw files. Whilst popular, AntConc's versatility means it is not as efficient with larger corpora. IntelliText [8] provides a web interface to a selection of pre-loaded corpora that have already been indexed. The system allows access to some reasonable sized corpora such as the BNC. As described above the system suffers a pitfall incurred by Zipf's law when searching for particularly common words but is very efficient in the general case; to remedy this, limits on the number of results returned are applied. Brigham Young's system (BYU) [9] provides an interface built upon a relational database that gives access to several extreme scale corpora such as the corpus of global web-based english (GloWbE) and the wikipedia corpus. BYU's interface provides a means to generate concordance lines or KWICs as well as a means to sort them in a number of meaningful ways such as lexically on one word left of the search term. As with IntelliText, BYU also limits the number of returned results - a common pattern with corpus systems. Systems based on the IMS Open Corpus Workbench such as CQPweb [16] currently have a hard upper limit (2.1 billion tokens) on the size of corpora that they can index.

Whilst traditional software tools used by corpus linguists may not be built specifically for big data and extreme scale corpora there are various examples of where the big data trend has collided with the world of corpus linguistics. Projects such as Google Books [10], an attempt to con-

solidate a vast amount of historical published texts into a corpus like structure is an example of this. Such projects have in part lead to a research field of culturomics. Various examples of studies conducted in this emerging field include [12] and [13], although these efforts have been criticised by linguists for having no suitable corpus design and noisy OCRed underlying data leading to skewed results, as well as the lack of awareness of large quantities of preceding research in the corpus linguistics field [11]. Various web derived datasets have also emerged such as Google's n-gram web dataset (part of the Google Books project) that are very firmly in the range of extreme scale corpora. Twitter has in recent years provided a strong way of building large scale corpora for various studies, examples of harvesting tweets and their meta data can be found in many recent language studies [14][15]. Clearly the big data trend is leading more and more to take notice of the power that corpus linguistics methods have to offer and many outside the realms of corpus linguistics are looking to utilise corpora-like big data sets.

Modern database systems that claim to handle big data will almost always require some form of clustering in order to achieve scaling. Whilst vertical scaling may be a feasible solution for moving from a medium size database to one two or three times the size, when your data set increases by several orders of magnitude vertical scaling is not enough and horizontal scaling must be considered. As well as the increased capacity for handling large amounts of data, horizontal scaling also allows for replication and therefore data redundancy which is essential when maintaining a large database. Whilst traditional relational DBMSs are all now capable of supporting horizontal scaling it is in the No-SQL movement of databases that the truly interesting systems designed from the ground up specifically to handle this kind of horizontal scaling exist. Cassandra is a clear example of this design philosophy as even when running Cassandra in a stand alone server the system still treats itself as a degenerate cluster of a single node. Cassandra unlike many others treats all nodes within a cluster equally with no need for separate configuration nodes or management nodes. MySQL and MongoDB, despite having clustering capabilities, still require at least one load balancing node and one configuration node.

III. Experimental Setup

To test the capabilities of modern DBMSs, two such systems were deployed into various clustered configurations and loaded with an extremely large corpus dataset. The DBMSs tested were MongoDB and Cassandra. Each database was deployed onto the Amazon Web Servies (AWS) EC2 platform using m4.xlarge instances (4 vCPUs, 16GB Memory, 100GB EBS Volume -500 provisioned IOPS). Each database was deployed in 4, 8 and 16 node configuration so that the scalability of each DBMS when storing corpus data could be examined. The corpus dataset loaded into the databases was the Hansard corpus described above. Each database was then queried for particular keywords. The keywords that made up the queries were derived from pre-generated word lists (each keyword query was run 5 times and an average taken). To gather an accurate reflection of each DBMSs capabilities queries were split into three groups; High frequency - the top 10 most commonly occurring words in the corpus. Medium frequency - 10 randomly selected words from the 60% range of words. Low frequency - 10 randomly selected words from the bottom 50% of words in the corpus.

Low Frequency	Medium Frequency	High Frequency
gauntly	weeny	it
croquet	kilometers	I
patronym	plebs	is
ratpayers	appraiser	a
thugutt	earldoms	in
ogies	candlemas	and
fecias	laudations	that
gacious	coachmakers	to
unspared	heinkel	of
moyland	conegate	the

Table I
KEYWORD FREQUENCIES

A. MongoDB

A minimal cluster configuration for MongoDB requires 3 components; a central query processor front end server, 3 configuration servers and a data node (shard). In practise many of these components can run on the same server. For testing the central query processor (mongos instance) plus the 3 configuration servers were deployed to a single AWS instance. Each data node was then deployed to its own separate AWS instance. This was deemed acceptable as keeping the configuration servers separate would be the best practice to ensure some redundancy in the cluster but this was only a test environment. Having the mongos server on the same AWS was also deemed acceptable as the tests are not designed to stress the database in terms of a high number of queries per second but rather how the database handles a large amount of corpus data. Three configurations were tested with 4, 8 and 16 data nodes.

The schema followed for MongoDB was for each word in the corpus to be inserted as a BSON (Binary Javascript Object Notation) document as described in Listing 1. The JSON document below describes the BSON documents inserted into MongoDB (note that the form of the word is not lemmatised in these tests);

The "docid" field is utilised as the shard key to ensure not only that the data is distributed evenly between the shards in the cluster but also to ensure when querying to build a concordance line for an occurrence of a word, one data node should be able to return all the BSON documents necessary to build the concordance line. A text index is defined in MongoDB on the searchableform field to search for word

instances. A hashed index is automatically built on the docid field to distribute the data but this index is also used when performing a ranged query to build concordance lines.

In MongoDB, data distribution is handled by selecting a shard key from within the data field. This key will be used to determine which data node the BSON document should reside on. Initially the source document ID was selected as the field for the shard key - this would ensure that all words from the same original source document would be present on the same data node - thus making it easier to build a concordance line as for each individual line all the word BSON documents could be gathered from the same node. During initial testing however it became clear that due to the incremental nature of the source document IDs using this field as a shard key lead to slower parse times as the MongoDB cluster was consistently shuffling data around between the nodes to ensure that the data was distributed evenly. To remedy this an artificial shard key was created. This shard key was a randomly generated number between a group of pre-defined ranges. Key range chunks were setup prior to parsing on the MongoDB cluster to define which range of keys to handle. Each alternate BSON document was then assigned a random value in this range when parsed to ensure during the initial insertion of the data the writes were distributed evenly across the cluster.

```
{
  //unique automatically generated id
    _id: ObjectId("5553324ca7986c0c3d6b3a97"),
    //word as it originally appeared
  originalform:   T h e     ,
    //word trimmed of white space and forced to
        lower case
  searchableform    t h e   ,
    //id of the source document the word appeared
        in
  docid:  S6CV0196P0-00481    ,
    //position within the source document
  pos: 103
}
```

Listing 1. JSON word entry in MongoDB

B. Cassandra

Cassandra can be configured into a cluster almost as easily as a standalone server can. Since it is designed with distribution in mind and makes use of several p2p architecture principles such as seeding and the equality of nodes each node need only be configured to point to a single commonly known set of nodes (seed nodes) from which configuration about the cluster is retrieved. As before Cassandra was deployed in 4, 8 and 16 node configuration onto AWS instances. During testing an additional seed was added per 4 nodes. Meaning the 4 node configuration contained 1 seed, the 8 node configuration 2 seeds etc. The seeds are of course nodes themselves and behave in just the same way as the rest of the configuration except they also provide management information to the rest of the cluster.

Although it is considered a No-SQL database Cassandra's query language CQL bares many similarities to SQL and simple commands and queries will often look identical between the two query languages. The schema used again followed a simple one word per record form and is described in listing 2.

```
CREATE TABLE hansard.words (
  doc text,
  pos int,
  s_f text,
  o_f text,
  PRIMARY KEY (doc, pos)
);
```

Listing 2. Cassandra CQL Schema definition

Similar to MongoDB the schema stores 4 values per record. "doc" the original source document, "pos" the words position in the source document, "s_f" the searchable form of the word (lower-case, removed punctuation), "o_f" the word as it originally appeared in the text. The primary key used is a composition of the source document and the position of the word in the source document - guaranteeing its uniqueness. A secondary index is then built on the "s_f" column to allow for keyword searches.

Cassandra allows for insertion to be performed and targeted at each node. This allows multiple large batch insertions to take place on different nodes. Cassandra uses its SSTable loader tool to import large quantities of data quickly. Prior to insertion the Hansard corpus was converted from its original XML to this SSTable format to allow for quicker insertion into the database.

IV. Results & Discussion

A. MongoDB

Fig. 1 shows the average query times retrieved using MongoDB for the sample tokens listed above for 4, 8 & 16 node cluster configurations. As would be expected based on other work around big data and the concept of scaling out the average query time improves significantly as the number of nodes in the cluster doubles - further parallelizing the index look up and retrieval tasks. The graph also illustrates how, despite the queries limiting the number of results returned to 20, the query time actually increases with the frequency of the word in the corpus. This implies that the text index used by MongoDB requires a significant amount of time to read the inverted file entry when it has performed the look up - if the entry in the index is larger because the word being looked up contains significantly more occurrences then the query response time will be significantly impacted.

Interestingly the raw numbers illustrated a pattern whereby the first test run of each query for high frequency words would be significantly slower (by an order of magnitude) than subsequent queries for the same word. This served to push up the average query times reported in Fig. 1. This pattern is likely the result of the text index being too large

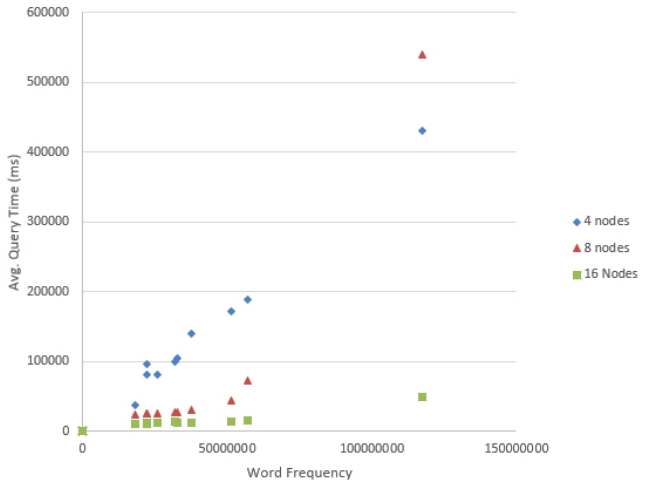

Figure 1. MongoDB Query Times

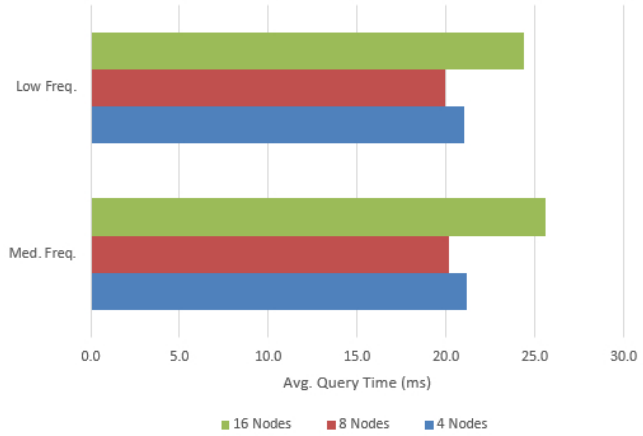

Figure 2. MongoDB Avg. Query Times (medium & low frequency words)

to contain wholly in memory meaning that when the text index is first searched it is read from disk and subsequently the entry for the word is then cached in memory. It is still however important to consider the figures whilst including this initial slow query as when working with big data sets such as Hansard it is likely indexes will not fit entirely into memory and will be read from disk.

A further anomaly observed in these results is the longer query time for the most frequent word ”the” when querying on an 8 node cluster. Whilst typically this could be heralded simply as an outlier the method of gathering these numbers through the average of several tests runs suggests this is not the case. Furthermore the point raised above of the initial high query time for the first query in the run could have contributed to push the average query time higher than that of a 4 node cluster for the same word however the raw results consistently showed longer query times across all test runs for the word ”the”.

The results of the low and medium frequency words are difficult to see in Fig. 1 and are presented expanded in Fig. 2. For the case of low frequency words that occur only once in the corpus it seems that increasing the number of nodes in the distributed setup has no clear effect on the retrieval time. This can also be seen looking at the retrieval times for medium frequency words with no obvious performance improvements gained by increasing the number of nodes in the cluster. Whilst for low frequency words occurring only once this might not be too interesting as a corpus linguist would likely need an even larger corpus to perform a meaningful concordance analysis with these words - the medium frequency words do have enough results for some kind of concordance analysis to be performed which shows that at this range of word (i.e. a minimum level for a concordance analysis) on this scale of corpus there is little to no benefit of large cluster setups with dozens of machines.

These results also further indicate the idea of a strong correlation between query times and index entry size - clearly when the index entry size is small enough very little variation can be seen in query time even if this index is distributed and the look up is taking place in parallel. This further supports the need for better indexing techniques to be developed for large text indexes which cannot fit into memory to allow for simple queries to be fulfilled more readily. For simple queries such as those utilized in these experiments a strategy of index entry paging and lazy loading could be utilized.

B. *Cassandra*

The results for Cassandra are immediately interesting as the raw figures seem to illustrate little to no correlation between the word frequency and the query time for the simple query being performed by our tests. Unlike with MongoDB where as the frequency of the word increased so to did the typical query time (suggesting increased time required to read the index as discussed above) Cassandra appears to demonstrate minimal overhead to reading large index entries for highly frequent words. In exploratory testing it appeared that Cassandra is far more influenced by the total number of results which are returned, indicating that it is capable of reading its index in a far more efficient way than MongoDB. This means that for simple queries such as those performed during testing which limit the results set to just 20, Cassandra was able to return results with far more consistent times across all frequency ranges than MongoDB.

Figure 3 shows the average query times for each word group listed above (low, medium & high frequency). From these results it is clear to see that for the scale of the Hansard corpus on the AWS infrastructure used there was a noticeable increase in performance between a 4 node and 8 node cluster configuration. However there was negligible difference seen between results between the 8 and 16 node

configurations suggesting that at this data scale there would be little to no benefit of scaling out such a configuration any further - albeit for these relatively simple corpus queries. It should be noted that some queries were marginally faster on the 8 node configuration than the 16 node configuration, although the difference in many cases was less than 1ms so may be explained by any cluster management information shared on the network between the nodes at the time of the tests.

Examining the raw results for Cassandra it also becomes clear that there is no initial index read period as noted in the discussion of MongoDB's results above. Perhaps due to Cassandra's seemingly better handling of indexes for these simple queries it is not burdened by the need to load the index entry into memory from disk and therefore the significantly longer query response time seen on the first query for each word in MongoDB is not seen at all when querying Cassandra. This would mean that in any system put in place in the real world for say a corpus search tool users would not experience significantly slower response times for a search on a keyword that has not been searched for before - or recently enough for its index entry to still remain in memory as in the case of MongoDB.

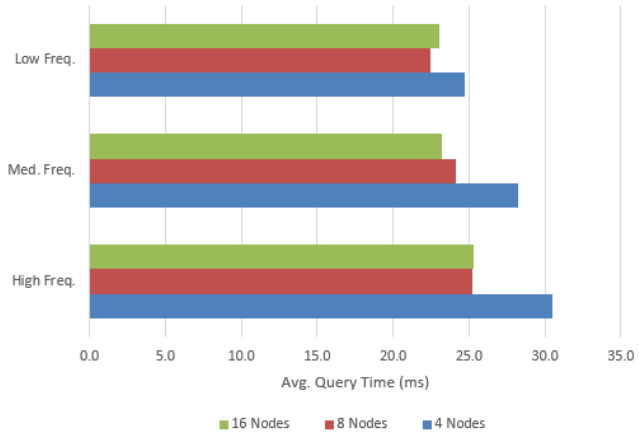

Figure 3. Cassandra Avg. Query Times

C. BYU & CQPweb

To put these results into context they were compared to query and processing times of existing corpus data systems, specifically BYU and CQPweb. Both systems now incorporate the Hansard corpus. Figures from CQPweb show that CWB takes over a day to build an index for the Hansard corpus which is significantly longer than MongoDB which in the configurations examined here took less than an hour. Query results from BYU also demonstrated comparable query response times for low and medium frequency words which would be expected as the experimental setup in this paper loosely follows the database pattern employed by BYU. For these simpler queries BYU could not match the query times of any Cassandra cluster configurations. However BYU did appear to outperform MongoDB for higher frequency words such as "in" for all but the 16 node cluster configuration (3956ms) giving marginally better query response times (5254ms) than the 8 node configuration (7243ms) and significantly better times than the 4 nodes configuration (99844ms).

V. Conclusions

The results for MongoDB and Cassandra illustrate that data scaling through clustering is a feasible solution to extreme scale corpus data sets. However they also demonstrate the limitations of some indexing strategies used by modern DBMSs and help to clarify the issues that must be addressed by any database or system wishing to manage extreme scale text corpora. It can also be concluded that if the motivation for utilizing a larger corpus is to broaden the base for a concordance analysis (i.e. to have a corpus with enough word occurrences for a valid analysis) then existing DBMSs such as those explored in this paper are viable solutions for performing KWIC searches when the word frequency is in the lower two thirds of the Zipf scale for the corpus. Here we have considered one of the major methods in corpus linguistics (concordancing) but we can expect that other methods (n-grams and collocations in particular, alongside annotated corpora) will place an even heavier burden on retrieval software and existing database designs.

From the results seen in this paper it is clear that Cassandra can outperform MongoDB for data at this scale when dealing with queries with a limited results range. However it is likely that due to limiting the number of results in the queries here that different results may be expected for more difficult to process queries. More difficult to process queries may include sorting the results that are retrieved - in such a case it can be assumed that it would be necessary to read the entire index entry for a keyword search and then sort the records even if the final results were then limited again. In these circumstances it is felt less disparity would exist between the two databases examined here.

As a result of the findings of this paper further work will be undertaken to explore corpus database patterns and how they can be applied to distributed setups similar to those used in this papers experimental tests. In particular the approaches taken towards index look up will be explored to ascertain what properties make certain indexing strategies more or less efficient for certain types of corpus queries. It is hoped that this further work will better define patterns that can build a truly scalable corpus database that can efficiently query extreme scale corpus datasets.

References

[1] Francis, W.Nelson (1964). "A standard sample of present-day English for use with digital computers."

[2] Leech, Geoffrey. "100 million words of English: the British National Corpus (BNC)." Language Research 28.1 (1992): 1-13.

[3] Cattell, Rick. "Scalable SQL and NoSQL data stores." ACM SIGMOD Record 39.4 (2011): 12-27.

[4] Baker, Paul, Costas Gabrielatos, and Tony McEnery. "Sketching Muslims: A corpus driven analysis of representations around the word Muslimin the British press 19982009." Applied Linguistics 34.3 (2013): 255-278.

[5] Manning, Christopher D., and Hinrich Schtze. Foundations of statistical natural language processing. MIT press, 1999.

[6] Leech, G., P. Rayson, and A. Wilson. "Word frequencies in written and spoken English." Harlow, England: Longman (2001).

[7] Anthony, Laurence. "AntConc: A learner and classroom friendly, multi-platform corpus analysis toolkit." Proceedings of IWLeL (2004): 7-13.

[8] IntelliText, Centre for Translation Studies (CTS), University of Leeds, http://corpus.leeds.ac.uk/

[9] Corpus BYU, Mark Davies, BYU, http://corpus.byu.edu/

[10] Google Books, Google Inc., https://books.google.co.uk/

[11] Hardie, Andrew. "Big Data in Language studies: from cargo cult science to phantom revolution." Proceedings of 7th International Conference on Corpus Linguistics (CILC2015).

[12] Hand, Eric. "Culturomics: Word play." Nature News 474.7352 (2011): 436-440.

[13] Leetaru, Kalev. "Culturomics 2.0: Forecasting large-scale human behavior using global news media tone in time and space." First Monday 16.9 (2011).

[14] Page, Ruth. "The linguistics of self-branding and micro-celebrity in Twitter: The role of hashtags." Discourse & Communication 6.2 (2012): 181-201.

[15] Gonzlez-Ibnez, Roberto, Smaranda Muresan, and Nina Wacholder. "Identifying sarcasm in Twitter: a closer look." Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2. Association for Computational Linguistics, 2011.

[16] Hardie, Andrew. "CQPweb – combining power, flexibility and usability in a corpus analysis tool." International Journal of Corpus Linguistics 17.3 (2012): 380-409.