UCL DEPARTMENT OF COMPUTER SCIENCE

Research Note
RN/12/14

A Performance Analysis of Distributed Indexing using Terrier

26 November 2012

Amaury Couste

Jakub Kozłowski

William Martin

Abstract

High performance indexing is critical to fast and efficient data retrieval, and underlies mainstream systems such as search engines. A limiting factor for indexing performance is data set size: while this may not be important for small desktop or smartphone applications, the scale of the dataset is problematic when indexing large corpora such as the web. Distributing both the dataset and the computation can solve this issue, and consequently a number of distributed indexers have been developed, but their effectiveness remains to be seen. To shed some light on this issue, we present a detailed performance analysis of one of the leading solutions. Terrier is a scalable piece of software supporting indexing and retrieval both on a single nodes and a distributed cluster. We demonstrate that performance is highly correlated to the specific setup of a cluster and aim to provide useful guidelines for data scientists willing to get the most out of their hardware.

A Performance Analysis of Distributed Indexing using Terrier

Amaury Couste
University College London
a.couste@ucl.ac.uk

Jakub Kozłowski
University College London
j.kozlowski@ucl.ac.uk

William Martin
University College London
w.martin@ucl.ac.uk

ABSTRACT

High performance indexing is critical to fast and efficient data retrieval, and underlies mainstream systems such as search engines. A limiting factor for indexing performance is data set size: while this may not be important for small desktop or smartphone applications, the scale of the dataset is problematic when indexing large corpora such as the web. Distributing both the dataset and the computation can solve this issue, and consequently a number of distributed indexers have been developed, but their effectiveness remains to be seen. To shed some light on this issue, we present a detailed performance analysis of one of the leading solutions. Terrier is a scalable piece of software supporting indexing and retrieval both on a single nodes and a distributed cluster. We demonstrate that performance is highly correlated to the specific setup of a cluster and aim to provide useful guidelines for data scientists willing to get the most out of their hardware.

1. INTRODUCTION

Indexing is the process of parsing a dataset and collecting and storing data to facilitate fast information retrieval. This process allows for searching, and is used by popular search engines such as Google [5] and Yahoo [10]. There are many different indexing algorithms and combinations of these algorithms; some of these perform quickly, whilst others may have alternative advantages such as producing smaller index files.

A performance problem for the vast majority of indexing solutions arises from the size of the dataset: more data means a way of quickly searching it is more useful, however indexing it poses many challenges; this drives the concept of indexing in the direction of very large datasets such as the web. Indexers by default run on a single machine, which often does not have the processing power, memory or even the storage space to process a large dataset. It was for these reasons that Google developed the MapReduce framework.

StuConOS 2012 University College London, United Kingdom

MapReduce [4] follows the divide-and-conquer design paradigm, based on the principle that we can handle more work if we divide it into smaller tasks and distribute it to multiple workers. The work is handled in two phases: Map, where data is processed and an interim result is produced; Reduce, where the results from Map tasks are combined. A typical job consists of splitting the dataset into evenly sized chunks to be processed concurrently by a number of mappers, each of which outputs a result. These results are then collected and aggregated by the reducers to form the final result.

But how can we leverage MapReduce to perform indexing? Apache Hadoop [1] is a Java implementation of the framework which allows us to create MapReduce tasks, and run them on data stored in the Hadoop Distributed Filesystem (HDFS), effectively creating a distributed indexer. We aim to investigate the benefits and pitfalls of using such a distributed indexer, and find if performance can scale linearly with the number of concurrent mappers. Additionally, we will try to identify an optimal configuration for a small test cluster.

In Section 2 we present the Terrier solution, a popular and open-source distributed indexing platform. In Section 3 we discuss the software installed on a cluster of computers that we have set up for testing. Details of the experiment and the setup of the cluster are given in Section 4, results are presented in Section 5. In Section 6 we summarise our findings, and draw conclusions about how the configuration of a cluster can affect results.

2. BACKGROUND

Here we will discuss Terrier and provide a quick survey of previous work.

2.1 Previous Work

McCreadie et al. [7] introduced the potential for near-linear performance characteristics of distributed indexing with Terrier, provided that multiple reducers are used and high-level of data locality can be achieved. The paper shows that indexing retains sub-linear characteristics when scaling the data set size. Additionally, the authors report that the solution compares very favourably with single-machine indexing, potentially achieving a 6.5× speed-up. The paper does not address the potential performance effect of adjusting number of map tasks that run in parallel. Similarly, no previous work has reported on the ease of installation and deployment.

2.2 Terrier

Terrier [9] is a solution that provides support for both single-machine and distributed indexing and retrieval (using Hadoop). The built-in support for test collections such as ClueWeb09, and the implementation of many retrieval models, both make Terrier a solid platform for research and experimentation in text retrieval. Like other free distributed indexers, Terrier is an experimental solution currently in development.

2.3 Questions

Terrier is still in the experimental stages of development. Consequently, most of the information available originates from the papers which have presented it to the community. It is for this reason that we believe an unbiased investigation into its performance is a useful one.

We wish to answer the following questions:

- How quickly can we index a large corpus?
- How quickly can we query a large corpus?

The latter question is a very important one: evaluation typically places emphasis solely on indexing performance; querying performance is arguably more important, as this is the bulk of the work that a live search engine must perform.

3. DESIGN

Here we will discuss the design of the experiments and the setup of the cluster.

3.1 Data

Indexers are used for searching in everything from small smartphone applications to powerful servers running large websites. As such, the data to be queried ranges from small records to large web-page corpora in the tens of terabytes.

Indexing is a popular topic at the Text REtrieval Conference (TREC) [8], which sees many papers published each year in the field. Most indexers presented at the conference in recent years are tested on the ClueWeb09 [6] dataset. This dataset provides a very large corpus to evaluate text retrieval methods. It consists of approximately one billion text files in ten languages along with sample queries, which makes it useful for a wide range of research applications.

3.2 Cluster

We performed experiments using a small heterogeneous cluster, the sort of cluster most likely to be used in practice by organisations. The cluster's specifications are listed in table 1, and individual node specifications are listed in table 2.

Table 1: Cluster specifications.

Physical machines	4
CPU cores	18
Memory	19GB
Hard drive storage	7TB

Table 2: Cluster nodes.

hostname	memory	disk space	cores
node0	7.8GB	1.3TB	8
node1	2.0GB	1.8TB	2
node2	4.8GB	1.8TB	4
node3	3.9GB	1.8TB	4

Every node runs a slightly different version of Linux CentOS 5.x and is configured with different firewall policies.

3.3 Tools

The following tools were used to set up and manage the cluster.

3.3.1 Cloudera Manager

Cloudera (CDH) [2] is a custom distribution of Hadoop and related software, packaged with Cloudera Manager. Cloudera Manager enabled us to set up the cluster aided with some automation, using a simple web-based interface; later we were able to remotely control the configuration of every node in the cluster, start and stop services and monitor the performance of the cluster in real-time.

3.3.2 Hue

Also included in CDH is Hue [3], a web interface that includes a File Browser and a Job Browser, both of which were great tools for monitoring the progress of jobs and viewing logs of tasks, and proved to be indispensable for debugging.

4. EXPERIMENT

Here we will discuss the experimental plan and setup.

4.1 Hypotheses

Our experiments must answer the questions discussed in section 2.3. We therefore formed the following hypotheses, which we will aim to prove or disprove:

1. The time taken to index a corpus scales linearly with the number of concurrent mappers.
2. The time taken to query a corpus scales linearly with the number of concurrent mappers.

The reason we chose to investigate the effect on performance of adjusting the number of concurrent mappers is twofold; firstly, oftentimes software ships may ship with suboptimal settings that may hinder potential performance, hence wasting resources; secondly, our cluster lacked homogeneity and therefore switching off machines was impractical.

4.2 Experimental procedure

We constructed algorithm 1, aimed at proving our hypotheses defined in section 4.1 and uncovering new performance information about Terrier.

To test query performance we ran batches of 150 queries taken from the Trec Web Track. As querying is much faster

Algorithm 1 Experimental procedure.

for all cluster nodes **do**
  Index subset of ClueWeb09B corpus (approx 15GB compressed).
  Record time taken to index.
  Perform batch of 150 queries.
  Repeat batch of queries for 100 trials.
  Record and average trial time.
end for

than indexing, we were able to run 100 trials for each experiment in order to allow for a meaningful statistical analysis. The results of the experiments will indicate whether a cluster of machines can quickly index and query a large dataset.

4.3 Cluster Setup

Initial setup was performed with use of Cloudera Manager; the only requirement was passwordless `sudo` access setup on all nodes; otherwise all software was installed automatically. Running demo Hadoop jobs was successful, therefore we set out to run Terrier. Unfortunately, there were several problems that had to be overcome. All problems had to be debugged by inspecting Hadoop logs.

Problem 1. Before we could start running jobs, we had to fix a bug in Terrier. The job setup code was copying `jar` file dependencies from the `CLASSPATH` to `HDFS`, without checking whether the files actually exist on the local hard drive.

Problem 2. In order to be able to control the partitioning of data into map tasks, we had to add a new configuration parameter to Terrier configuration architecture.

Problem 3. The cluster was spuriously failing to start jobs and the logs were pointing towards lack of disk space. It turned out that Hadoop logs were mapped to `/var`, which in turn was mounted on an extremely small disk partition. Moving the logs to the same partition as `HDFS` on all nodes fixed the issue.

Problem 4. An issue that remains only partially solved is the stream of Java `OutOfMemoryException`s coming from Terrier code. We were led to believe that because files in the corpus are quite large (on average $\approx$ 150MB), and need to be uncompressed in memory, the `JVM` was simply running out of memory. Therefore, we continued to increase max heap sizes, to a point where at around $\approx$ 2GB heap sizes, we decided that the problem was likely pointing to a bug in Hadoop part of Terrier. We base this claim on the fact that there seems to be no clear connection between the size of the file and the probability of failure (a 204.2MB file would index fine, however the process would fail on a 184.1MB file) and the process always fails on the same exact files.

In order to rule out possible causes, we ran a single-machine configuration of Terrier on the files that failed on the Hadoop configuration and the process was successful. We also think that this issue may have caused the index jobs to sporadically fail or succeed given the same configuration; it was also identified that a greater number of concurrent mappers seems to increase the likelihood of failure. It would be a useful experiment to try to index the files that fail on another cluster, in order to eliminate the possibility of an error in the cluster configuration.

4.4 Dataset

We performed our experiments on a subset of the ClueWeb09 [6] dataset, approximately 14GB compressed or 100GB uncompressed data.

5. RESULTS

Due to the significant differences in specifications between cluster nodes, we decided that selectively removing machines would not lead to any meaningful results. Instead, we attempt to explore the indexing performance by adjusting the number of concurrent mapper tasks on each node and maximum number of map tasks. Unfortunately, the volume of data collected was not sufficient to warrant statistical treatment of execution time measurements from multiple runs to rule out inherently stochastic properties, therefore we will only draw general conclusions about the performance of distributed indexing with Terrier. Results are presented in tabular and graphical form.

5.1 Adjusting Concurrent Map Tasks

We performed the following experiments by adjusting the number of concurrent map tasks on each of the nodes. The hardware specifications vary from node to node, so we scaled the number of tasks according to the number of processor cores on the machine. In order to be able to increase the number of concurrent map tasks we had to keep the `Java Heap Size` at $\approx$ 250MB.

5.1.1 Indexing

As shown in table 3 and fig. 1, the peak performance is achieved at 100% processor core utilisation. Increasing the number of concurrent map tasks beyond this point imposes context switching overhead.

Table 3: Results for indexing times adjusting concurrent map tasks.

total concurrent map tasks	time taken
9	2h 15m 5s
18	2h 2m 0s
27	2h 5m 58s
36	2h 36m 16s

Figure 1: Plot for indexing times adjusting concurrent map tasks.

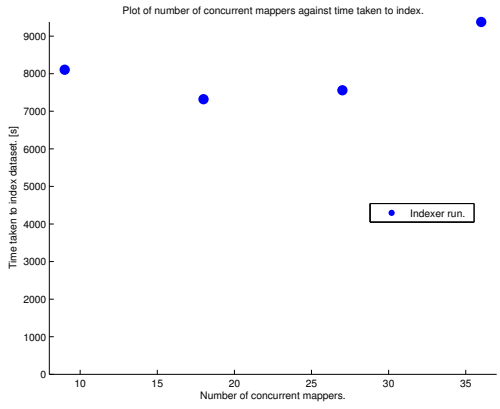
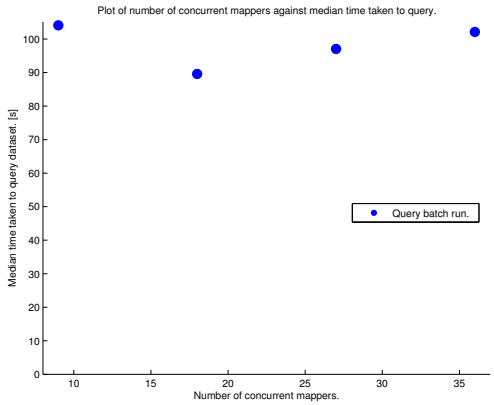


Figure 2: Plot for median querying times adjusting concurrent map tasks.



5.1.2 Querying

Table 4 and fig. 2 show a strong correlation with the results from indexing the corpus, reaching maximum performance somewhere around 100% processor core utilisation and achieving suboptimal performance at both ends of the spectrum. When the system operates below the processor capacity, some mappers may finish sooner than others and sit idly; similarly, overcapacity may impose unnecessary context switching cost. This is a good result in itself, showing that we can potentially improve query time with indexing time.

5.2 Adjusting the Maximum Map Tasks

We performed the following experiments by adjusting the number of maximum mappers. This has the effect of increasing the number of documents processed per-map task.

As we were adjusting maximum map tasks and therefore leaving the concurrent tasks constant at 100% processor core utilisation, we were able to increase the maximum java heap space to $\approx$ 768MB without fear of running out of physical memory on any of the nodes. This results in better performance than the previous experiment.

5.2.1 Indexing

We repeated the indexing trials as in the previous experiment, adjusting maximum mappers instead. Table 5 and fig. 3 show that limiting the maximum mapper tasks to 25 hinders performance, but other configurations perform similarly. A likely cause for this is under-utilised CPU time: each map task was larger, but the files in the corpus are not of exactly the same size, therefore some nodes will finish early and be idle.

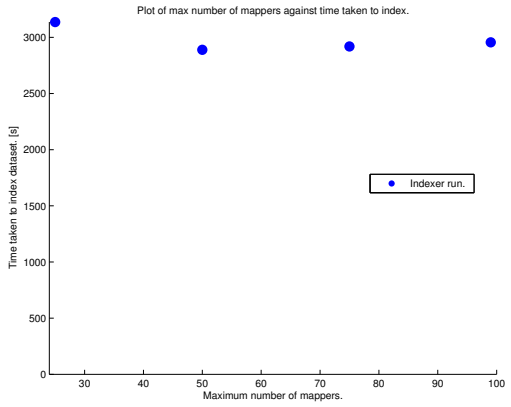
Table 4: Results for query times adjusting concurrent map tasks.

map tasks	time taken (s) min	median	max	stddev
9	103.13	104.11	128.25	3.86
18	89.04	89.60	92.61	0.65
27	95.65	97.05	126.65	3.13
36	100.57	102.13	134.66	5.53

Table 5: Results for indexing times adjusting maximum map tasks.

maximum map tasks	time taken
25	52m 17s
50	48m 9s
75	48m 39s
99	49m 16s

Figure 3: Plot for indexing times adjusting maximum map tasks.



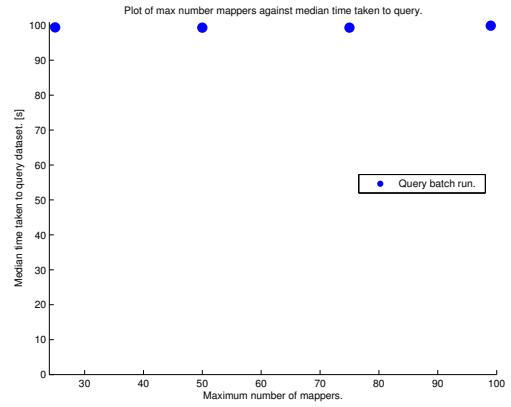
5.2.2 *Querying*

We repeated the trials as in the previous experiment, but running just 25 trials for each different configuration. Table 6 and fig. 4 show that querying is unaffected by adjusting the maximum number of mappers.

Table 6: Results for query times adjusting maximum map tasks.

max. tasks	time taken (s) min	median	max	stddev
25	97.69	99.38	99.89	0.50
50	98.68	99.32	105.75	1.37
75	98.44	99.31	131.17	8.50
99	98.79	99.88	134.27	8.31

Figure 4: Plot for median querying times adjusting maximum map tasks.



6. CONCLUSION

We have assessed the ease of running a distributed indexing solution based on Terrier, and the performance that can be achieved. This software proved to be fairly hard to set up, and we have managed to uncover some issues. The overall performance, however, was very promising and our results emphasise the importance of tuning cluster configuration to improve indexing time.

In summary, we believe the following key points can be learned from this experience:

- Cloudera Manager and Hue are an excellent Hadoop distribution. The ability to easily deploy and monitor the cluster through a web browser proved very valuable.
- A cluster with homogeneous nodes is preferable.
- Experiment results are highly dependent on the particular cluster configuration; it is possible to achieve better performance by tweaking the number of concurrent mappers.
- Distributed indexing performance scales with the number of concurrent mappers until 100% core utilisation is achieved. Optimal performance may be achieved from the correct configuration, but not from simply adding mappers.
- Current indexing solutions are experimental software and seldom work out-of-the-box. Additional time tweaking configuration files must be accounted for when planning a deployment.
- Debugging distributed software is difficult. Logs are sparse and often obscure.

References

[1] Apache Software Foundation. Hadoop. URL `http://hadoop.apache.org/`.

[2] Cloudera Inc. Cloudera, . URL `http://www.cloudera.com/`.

[3] Cloudera Inc. Hue, . URL `http://cloudera.github.com/hue/`.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL `http://doi.acm.org/10.1145/1327452.1327492`.

[5] Google Inc. Google. URL `http://www.google.com/`.

[6] Lemur Project. ClueWeb09. URL `http://lemurproject.org/clueweb09.php/`.

[7] R. McCreadie, C. Macdonald, and I. Ounis. On single-pass indexing with mapreduce. *Proceedings of the Annual International ACMSIGIR Conference on Research and Development in Information Retrieval*, pages 742–743, 2009. URL `http://eprints.gla.ac.uk/39994/`. isbn: 9781605584836.

[8] National Institute of Standards and Technology (NIST) and the Intelligence Advanced Research Projects Activity. Text Retrieval Conference. URL `http://trec.nist.gov/`.

[9] I. Ounis, G. Amati, Plachouras V., B. He, C. Macdonald, and Johnson. Terrier Information Retrieval Platform. In *Proceedings of the 27th European Conference on IR Research (ECIR 2005)*, volume 3408 of *Lecture Notes in Computer Science*, pages 517–519. Springer, 2005. ISBN 3-540-25295-9.

[10] Yahoo! Inc. Yahoo. URL `http://www.yahoo.com/`.