An Investigation into Query Throughput and Load Balance Using Grid IR

Ahmad Abusukhon¹, Michael P.Oakes¹
¹School of Computing and Technology
University of Sunderland,
Chester Road, Sunderland
SR1 3SD, UK
Tel.: +44(0)1915152222
ahmad.abusukhon@sunderland.ac.uk, michael.oakes@sunderland.ac.uk

Grid information retrieval (GIR) means using a grid system for retrieving relevant documents that satisfy the user need from within a large-scale data collection. A data collection can be text, audio, video, etc. The grid provides powerful computation while information retrieval provides techniques for retrieving useful information. In previous work, we built our baseline system and we compared three types of IR systems, namely document-based, term-based and hybrid partitioning with respect to average query response time. In addition, we proposed new methods for improving the load balance and query throughput for term-based and hybrid partitioning. We carried out a set of real experiments using a broker and six nodes. In this paper, we propose to repeat our previous experiments using the grid as a new IR environment.

Load balance, query-throughput, query-routing, partitioning methods, query response time, Grid IR.

1. INTRODUCTION

The number of documents available on-line has grown rapidly since the birth of the World Wide Web. Baeza-Yates and Ribeiro-Neto (1999, p.229) mentioned that the web size is growing very fast, nearly doubling in size every six months. Currently, there are more than 20 billion indexed web pages, (Baeza-Yates et al., 2007). Besides the huge number of documents, we have a huge number of queries (billions of queries) submitted by users (Badue et al., 2001). Thus, as the number of queries submitted by the users increases, it becomes necessary to increase the system throughput (the number of queries executed per second), and the IR system must provide a high query processing rate.

T.Meadow et al. (2007) defined Information retrieval as “finding some desired information in a store of information or a database”. They also defined information retrieval as a communication process which includes finding documents, graphic images or sound recordings that are relevant to the user need. Using the information retrieval techniques users can communicate with a librarian, museum or fingerprint identification specialist. Thus, the main aim of an information retrieval (IR) system is to provide users with efficient and easy access to the information relevant to their needs. This task becomes difficult as the size of the data collection becomes large (multi Gbytes).

IR systems are implemented either on a single machine (i.e. sequential IR systems) or on parallel machines. In sequential IR systems, only one machine is responsible for constructing an index and answering the user query. MacFarlane et al. (2005) stated that “indexing large multi-gigabyte text databases can take many hours or even days to complete”. Zobel and Moffat (2006) mentioned that the main problem that made index construction challenging was that the entire index of the whole collection could not fit in main memory. In addition, answering queries using a single machine resulted in low query response time.

To solve the above problems, traditional IR systems must change their architectures and algorithms to parallel and distributed architectures (Baeza-Yates & Ribeiro-Neto, 1999). D.Manning et al. (2008, p.74), mentioned that for large collections (for example the Web) we cannot construct the index on a single machine, so we should use large clusters of computers in order to carry out this task.

In distributed IR systems, a big problem is divided into a small task (i.e. the overall computation is decomposed into a number of small tasks) and then each node is assigned a certain task. The decomposition and assignment of tasks to different nodes in the system is called partitioning. The query response time is greatly influenced by the way in which a large scale data collection is partitioned across multiple nodes. Jeong and Omiecinski (1995) proposed two different methods for partitioning an inverted file among multiple disks, namely document id and term id. One of the major problems that may arise here is the load balance. The problem of load balancing is to develop a partitioning and mapping algorithm in order to balance the workload across nodes and reduce the communication time between the nodes (processors).
In the previous work, we built our baseline system by comparing between three types of IR systems, namely document-based, term-based and hybrid partitioning with respect to the average query response time. In addition, we showed how the load balance and the query throughput of term-based and hybrid partitioning could be improved by using new methods for query routing and collection partitioning (Abusukhon et al. 2008a, b, c), see Sec. 3.

In future, we propose to repeat our previous experiments using the University of Sunderland Cluster Computer (UoSCC) or simply the Grid. The University of Sunderland launched the Grid on 24th of April 2008. The grid consists of central node (Xeon Intel Core Duo 64bit) that communicates with a cluster of nodes called the head nodes. The head nodes are connected to compute nodes (based upon Dell Server type 2950). The total number of nodes in the system is 42 computing nodes. The number of CPU processors per node is two, while the RAM of each node is 8Gbytes. The network bandwidth is 1Gbps, the data storage is a SATA drive, 250Gbyte per node, and thus the total distributed storage is 40 * 250Gb = 10Tb. In addition, the grid uses dual operating systems namely Linux and Windows. The Grid (UoSCC) has been designed in order to be a general purpose machine, and may be applied in different areas. For example, it may be used for solving scientific and engineering problems such as network planning, web based search engines, media applications such as rendering and data transcoding, multimedia and high bandwidth video streaming, general purpose simulation parameter exploration and numerical optimisation, and computational fluid dynamics (CFD).

GIR (Grid Information Retrieval) is similar to distributed computing. It consists of a number of nodes that are connected by a network. In addition, tasks (or subtasks) are assigned to different nodes, and carried out in parallel. However, the difference between GIR and distributed systems is that GIR provides more finely grained implementation for task assignment and coordination among the grid elements. In fact, GIR is composed of three components that can exist on any machine on the computational grid, namely collection managers (CMs), indexers, and query processors (QPs). The collection manager is responsible for accessing, storing, transforming and delivering data items from the collection, and CMs may have privileged access to data. Indexers make up the heart of the traditional IR system; they build the index of a given data collection in order to provide efficient access to that collection. The query processors are responsible for interacting with the indexers, over time, in order to collect query answers, merge them, and send them to their users (Dovey & Gamiel, n.d.; Gamiel et al., 2004).

The performance of IR systems is measured by retrieval efficiency and retrieval effectiveness. Measures such as recall and precision are commonly used for measuring retrieval effectiveness (R.Korfhage 1997, p.194). Suppose that we have a user query Q and its set (R) of relevant documents. Let the number of documents in the set (R) equal r. Assume that the query answer (after applying a given retrieval strategy) to query Q is the set of documents (A). Let the number of documents in this set equal a. In addition, let Ri be the number of documents in the intersection of the sets R and A, then the recall and precision measures are defined as follows. Recall is “the fraction of the relevant documents (the set R), which has been retrieved” i.e., recall = Ri/r. Precision is “the fraction of the retrieved documents (the set A), which is relevant” i.e., precision = Ri/a (Baeza-Yates & Ribeiro-Neto, 1999 p.75). Retrieval efficiency is defined as “the measure of the time taken by an IR system to do a computation on the database, although this usually means search it” MacFarlane (2000). The gain in retrieval efficiency using distributed IR or Grid computation against sequential IR is measured by the speed up and efficiency of the system as well as the system throughput. For example, how many queries are carried out per second? How many Mbytes are indexed per hour?

2. RELATED WORK

In IR there are three types of collection partitioning, namely document-based, term-based and hybrid partitioning. In document-based partitioning, documents are distributed across nodes in a circular round robin fashion such that each node gets a sub-collection of the whole data collection. Each node builds its local index for the sub-collection it received and becomes responsible for answering queries about its sub-collection. In document-based partitioning, queries are carried out sequentially. In other words, it is difficult to direct queries to their relevant nodes because all nodes are responsible for answering a certain query term (A.Ribeiro-Neto & A.Barbosa, 1998; Badue et al., 2001;Cambazoglu et al., 2006). In order to achieve better load balance when using document-based partitioning, Ma et al. (2002) proposed three different approaches for inverted list partitioning, namely consecutive partitioning scheme, interleaving scheme, and differential partitioning. In consecutive partitioning, each workstation holds a set of consecutive documents identifiers (ID’s). In interleaved partitioning, the mapping between the document number and the workstation was carried out using the equation L(d) = d Mod M, where d is the document identifier (ID), and M is the number of workstations. In differential partitioning, a document weight was assigned to each document based on the probability that a certain term appears in the query, then each node was assigned a set of documents with balanced-weight using the following equation. Balanced-weight= total-document-weight / M, where M is the number of workstations.

In term-based partitioning, a global index is built where the terms and their complete inverted lists are stored in the broker and then the broker distributes the terms associated with their inverted lists across nodes such that each node gets a sub-set of terms, and thus each node is responsible for answering queries about the terms it stores. In term-based partitioning queries are directed to their relevant nodes and the IR system can carry queries concurrently, where one or multiple queries can be submitted to the system at once (Cambazoglu et al., 2006).
In order to improve the load balance, Xi et al. (2002) proposed hybrid partitioning. They divided the inverted lists into a number of chunks and then distributed them across nodes randomly. The broker distributed the query term across all nodes (i.e. query terms were not directed to their relevant nodes) sequentially (i.e. one query at a time), thus all nodes were responsible for answering a certain query term. They increased the query throughput of their system by increasing the multi-programming level in order to carry out more than one query term at a time. They concluded that hybrid partitioning outperformed document-based partitioning especially when the chunk size was small.

Improving the load balance of term-based partitioning was the focus of other researchers. Current approaches for improving the load balance for term-based partitioning can involve concurrent queries where more than one query is processed at the same time (Cambazoglu et al., 2006). This is done in order to reduce the number of idle nodes in the system, and thus achieve better load balance and increase the system query throughput as well. Alternatively, using partial replication and caching, Moffat et al. (2006) aimed to balance the load for pipelined term-distributed parallel architecture, and they proposed different techniques in order to reduce the query costs. They showed that load imbalance could be addressed by techniques based on predictive inverted list assignments to nodes and selective list replication. In a pipelined system, query evaluation is executed as follows: suppose that we have N nodes, and that we have a query consisting of three terms (t1, t2, and t3) that reside on three nodes: n1, n2, and n3. The query evaluation begins at n1, which retrieves the inverted list of term t1, sends it to node n2, which retrieves the inverted list of t2 and sends it with the inverted list of t1 to n3 and so on. The disadvantage of this system was the load imbalance, which was caused by a number of terms with long inverted lists. In order to solve this problem they proposed replicating the inverted lists of those terms over nodes.

Marin and Costa (2007) investigated improving the load balance for term-based and document-based partitioning by balancing query ranking and query fetching. They mentioned that on current cluster technology the most expensive steps (most dominant factors) in query evaluation are fetching lists from disk and query ranking. Thus, in their proposed system they detached ranking from fetching. List fetch was balanced by caching the most frequent terms in queries. A third technique is to balance query ranking and query fetching (Marin and Costa, 2007).

In fact, searching inverted lists is heavily disk dependent. In other words, most of the searching time is spent on retrieving the inverted lists from the disk. Dewitt & Gray (1990) mentioned that query response time decreased when more processors and disks were used (there is a point beyond which increasing the number of processors and disks increases the query response time), this is because the index size searched at each node becomes smaller and thus the response time decreases. Badue et al. (2001) used shared-nothing architecture in order to compare term-based and document-based partitioning. They concluded that term-based partitioning was better than document-based partitioning because term-based allowed the parallelization of disk seeking. MacFarlane (2000) compared document-based and term-based partitioning and they concluded within their system that document-based is the preferable method for search. Frieder & Tova Siegelmann (1991) concluded that the performance of multiprocessor information retrieval systems depends on both the underlying parallel technology and the organization of the data to be retrieved. They stated that “poor data allocations result in minimal performance gains on a parallel engine as compared to a uniprocessor system”. In other words, different data partitioning (allocation) algorithms lead to different retrieval efficiency (i.e. different query response times). Thus it is very important to decide where the inverted lists must reside.

Partitioning the tasks across nodes is not an easy task. For example, partitioning the index (e.g. term-based index) by sending an equal number of terms across multiple nodes may not always result in equal amount of work (Frieder et al., 2000; Moffat et al., 2006). Although different algorithms have been implemented in order to improve the load balance of term-based and document-based partitioning, this problem is still an open question in distributed IR systems (Baeza-Yates et al., 2007).

Our research looks at the problem of efficiently retrieving (i.e. reducing the query response time) for documents that are relevant to the user needs in a large-scale static data collection (multiple Gigabytes). In our research, we focus on improving the load balance and the query throughput as factors of the query response time.

3. WORK COMPLETED TO DATE

3.1 Baseline system
Abusukhon et al. (2008a), compared three types of IR systems namely, document-based, term-based and hybrid partitioning with respect to the average query response time. In their experiment they used six nodes (256 RAM) connected to a broker via 10/100 Ethernet switch. They used the data collection WT10G, and 50 queries (451-500) from TREC-9. In document-based and hybrid partitioning, query terms were broadcasted over all nodes. In term-based partitioning, query terms were directed to their relevant nodes. Unlike Xi et al. (2002), they did not find that hybrid partitioning was any better than document-based partitioning in terms of the average query response time although they chose the chunk size to be as small as possible. They justified their conclusion with respect to communication and merging time. However, as in the previous work, they found that document-based partitioning and hybrid partitioning outperformed term-based partitioning in terms of average query response time. In hybrid
partitioning and document-based partitioning a certain query term may appear on more than one node, thus queries were broadcasted over all nodes because the broker does not have any pre-knowledge about where each term and its associated list are located.

3.2 An investigation into improving query throughput and load balance of hybrid partitioning

Abusukhon et al. (2008b) mentioned that hybrid partitioning suffered from low query throughput because it was difficult to direct queries to their relevant nodes. In order to solve this problem they proposed to divide the nodes into clusters such that each cluster of nodes is responsible for answering a query term (note that in Xi et al.’s method, all nodes were responsible for answering a certain query term). They built the term-based index and then they divided each inverted list into (m) chunks, where m equals the number of nodes in each cluster. They distributed the chunks associated with their terms across clusters such that all terms started with a certain set of letters resided on a certain cluster. In addition, they distributed the chunks of a certain set of terms (for example, all terms starting with the letters A to D) across the nodes of a specific cluster such that each node in the cluster stored part of the complete inverted list of a certain term. They implemented the hybrid IR system and then they ran 50 queries and measured the load balance and query throughput, and then they implemented their IR system in which queries were directed to their relevant clusters and they measured the query throughput and the load balance.

Their results showed that their proposed method improved the query throughput of Xi et al.’s by 60%. In addition, they investigated carrying out multiple queries (μ queries) at once. They defined the hybrid queries which resulted from combining μ queries into one query (Q) then removing all duplicated terms from (Q) and then split Q into N streams, where each stream contains all terms (from all queries) starting with a certain set of letters (for example, A to D), and then they directed each stream to its relevant cluster (node). They studied how the query throughput and the load balance were affected by μ. They found that increasing the μ value led to better load balance and query throughput and thus the average query response time was reduced. They compared term-based partitioning and hybrid partitioning when μ = 50 queries (carried out at once) and they found that within their system the hybrid partitioning performed better than term-based partitioning. This was the first work which compared term-based and hybrid partitioning when queries were directed to their relevant nodes.

3.3 An investigation into improving the load balance of term-based partitioning

The main disadvantage of term-based partitioning was the load imbalance. Directing queries to their relevant nodes results in load imbalance and thus some nodes (those which store the most frequent terms) are heavily loaded while other nodes may be idle or lightly loaded (Badue et al., 2001). Recent research (Baeta et al., 2007), showed that it is still unclear on the circumstances under which each of the two partitioning algorithms (term-based and document-based) is suitable. They also stated that another open problem is how to find an efficient way of data partitioning for both term-based and document-based partitioning that achieves load balance among the different servers.

In order to solve the above problem, Abusukhon et al. (2008c) proposed two new techniques for query routing and index partitioning, namely term-total-frequency and term-length partitioning. In the first technique, they proposed to distribute the term-based index equally across nodes with respect to the total term frequency calculated from the inverted lists. For example, suppose that term (x) appears in the following inverted list: 3:1, 6:3, 9:2, 20:4, where each pair represents the document identifier and the term frequency in that document, then the total term frequency of term x equals 10. They passed through the term-based index and they calculated the total frequency (F) for each term, and then they stored (F) and the term associated with (F) into a lookup file. The lookup file was sorted with respect to the total term frequency and then the term-based index was distributed among nodes with respect to the lookup file.

The aim of the above experiment was to distribute the load (the summation of the total frequency of all terms) equally across nodes. The motivation for their work was the Zipf’s law, which is used to capture the distribution of the frequencies of the words in a given text. This law stated that “the frequency of the i-th most frequent word is 1/i times that the most frequent word”, (i being between 1.5 and 2.0), thus the frequency of any word is inversely proportional to its rank (i.e. i-th position) in the frequency table. Their hypothesis was that if the most frequent terms in the term-based index were distributed equally across nodes then the load balance might be improved. In the second technique, they proposed to distribute the term-based index equally among nodes with respect to the term length (in characters).

The motivation for term length partitioning comes from the observation that queries in Excite-97 have a very skewed distribution of term lengths with some predominant lengths. Their hypothesis was that if the predominant-length terms resided on one or two nodes in the system then most of the user query terms will be answered by one or two nodes while other nodes are lightly loaded or idle. Thus, the terms of the term-based index were distributed equally across nodes with respect to the term length in order to improve the load balance. In addition, they studied the load balance of term-based partitioning when terms were distributed among nodes in a circular round robin fashion, as well as, when terms were distributed equally among nodes with respect to the inverted list length (in bytes). They ran 10,000 queries from Excite-97, and they measured the load balance of the above four techniques.
Their results showed that within their system, the term-length partitioning performed better than other techniques in terms of load balance.

4. CONCLUSION

Fig. 1 shows a summary of work by previous authors on improving the load balance and the query throughput as important factors of the query response time, to illustrate where our own work (labelled [1], [2] and [3] as in the reference list below) has attempted to fill in the gaps in the literature. Overall, we have addressed the problems of how to improve the query throughput and load balance for hybrid partitioning and how to improve the load balance for term-based partitioning. In future, we will repeat the experiments we have already carried out using 7 nodes, this time using the Grid System at Sunderland. The Grid provides powerful computation because it consists of a large number of computational nodes (42) that work in parallel in order to solve large-scale problems.

5. ACKNOWLEDGMENT

Our thanks to Al-Zaytoonah University of Jordan and The University of Sunderland for their sponsorship.

REFERENCES.


