Computer and Information Science; Vol. 6, No. 3; 2013
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

Exploiting Parallelism in Query Processing for Web Document Search Using Shared-Memory and Cluster-Based Architectures

Amal Elsayed Aboutabl[1]

[1] Computer Science Department, Faculty of Computers and Information, Helwan University, Cairo, Egypt

Correspondence: Amal Elsayed Aboutabl, Computer Science Department, Faculty of Computers and Information, Helwan University, Ain Helwan, Cairo, Egypt. E-mail: aaboutabl@helwan.edu.eg

Received: June 30, 2013 Accepted: July 24, 2013 Online Published: August 1, 2013

doi:10.5539/cis.v6n3p125 URL: http://dx.doi.org/10.5539/cis.v6n3p125

Abstract

Achieving interactive response times when searching for documents on the web has become a challenge especially with the tremendous increase in the size of information available nowadays. Incorporating parallelism in search engines is one of the approaches towards achieving this aim. In this paper, we present a model for parallel query processing. Then, this model is extended particularly for usage on shared-memory and cluster parallel architectures. A special simulator, reflecting the proposed model, was developed allowing parameters concerning the data set, queries and architectures to be varied. A total of 32 experiments were conducted and the output was studied for the effect of varying different parameters. A number of performance measures such as average response time, speedup and efficiency are computed to study the effect of varying the parameters. Results show that in terms of average response time, speedup and efficiency, the proposed model for parallel query processing on shared-memory architecture outperforms that on cluster-based architecture.

Keywords: parallel processing, query processing, shared-memory systems

1. Introduction

With the dramatic increase in the size of the web, search engines had to scale up to keep up with this growth. In 1997, the top search engines indexed from 2 million to 200 million web documents. Nowadays, estimates of the number of indexed pages has reached at least 15.78 billion pages as reported by Yahoo, Google and Bing in Daily estimated size of the World Wide Web (2013). Nevertheless, there has been a vast increase in the number of internet users due to the fact that the world wide web has become the primary source of information. Hence, the number of queries increased dramatically from an average of about 1500 query/day in 1994 to 20 million queries per day in 1997 as reported by Altavista. An even more dramatic increase in the number of queries per day occurred thereafter. In 2011, the average number of queries per day reached around 4 billion on Google as reported in Google Annual Search Statistics (2013). With the help of the improvement in hardware performance, the task of scaling up search engines to handle such vast amount of data and queries has become more attainable.

There are three main components in any search engine (Brim & Page, 1998); the *Web Crawler* which continuously crawls into the web for new documents, the *Indexer* which indexes these documents and the *Searcher* which receives queries from users and attempts to find relevant documents. The scope of this work lies within the second and the third components. Firstly, we attempt to index the documents in such a way that parallelism is exploited in query processing. Secondly, we propose an approach that parallelizes query processing using shared-memory and cluster-based architectures.

2. Related Work

Various research attempts have focused on the topic of performing query processing efficiently. One such attempt was to cluster user queries according to their contents as well as user logs (Wen, Nie, & Zhang, 2001). With the huge amount of available data and number of queries, the use of parallel processing has become indispensable (Singhal, 2001). Therefore, this area has attracted the interest of many researchers. One of the first attempts in this respect was studying parallel query processing on shared-everything parallel systems (Hong & Stonebraker, 1991). Konstantopoulos, Mamalis, Pantziou and GavalasHong (2009) proposed parallel algorithms for document retrieval of BSP (Bulk Synchronous Processors) and CGM (Coarse-Grained Multiprocessors). They, analytically, proposed cost models for query processing on both architectures in terms of computation,

communication and memory usage. Frachtenberg (2009) studied coarse-grained and fine-grained parallel approaches for web search using indexing to reduce latencies. A study on resource allocation selection and scheduling was performed by Epimakhov, Hameurlain, Dillon and Morvan (2011) with the purpose of parallelizing queries in heterogeneous grid environments. They performed a performance analysis of static, dynamic and hybrid allocation methods as well as incentive-based methods using their own developed simulator. A review on interquery and intraquery partitioning schemes was provided by Cambazoglu, Catal, and Aykanat (2006). Ding, He, Yan and Suel (2009) used a system based on graphic processing unit (GPUs) for high performance query processing by parallelizing specific subtasks such as inverted list compression, list intersection and top k scoring. Delbrua, Campinas and Tummarello (2012) devised a high performance indexing model for semi-structured information which is applicable on heterogeneous environments,

PC clusters have been considered as a low cost parallel solution for query processing and text retrieval in general. Rungsawang, Laohakanniyom and Lertprasertkune (2001) performed parallel indexing of huge data using PC clusters for fast text retrieval using the PVM message passing library. Another attempt (Chung et al., 2001) provided a PC cluster parallel system based on partitioning the inverted index file among the cluster nodes' hard disks such that every term in a query is sent to the relevant node. They used a Distributed Shared Memory (DSM) programming technique and proved that this technique outperforms a Message Passing Interface (MPI) implementation in terms of speedup. Further issues have been considered concerning using PC clusters to retrieve information from a huge amount of data. Kang et al. (2004) studied the problems of node failure and load imbalance and provided a method based on data duplication to achieve both fault tolerance and load balancing.

3. Index Partitioning Schemes

3.1 Inverted Indices

Currently, search engines use the inverted index data structure for faster and more efficient query processing (Zobel & Moffiat, 2006). Every document to be indexed is associated with a number of terms. These terms may be simply words appearing in the document or topics covered by the document if more sophisticated term extraction methods are used (Svenonius, 2000). An inverted index consists of a number of inverted lists each of which is associated with a term. Each inverted list consists of a number of postings pointing to documents where the term appears. In query processing, a query consists of a number of terms and it is required to search the inverted index for relevant documents. We use a simple example to clarify the concept of inverted index and related partitioning schemes which will be described in the section 3.2. The following is a simple document collection which consists of 6 documents where each document is associated with a set of terms.

$$d_1 = \{t_2, t_5\}$$
$$d_2 = \{t_1, t_2, t_4\}$$
$$d_3 = \{t_3, t_5\}$$
$$d_4 = \{t_1, t_4\}$$
$$d_5 = \{t_3, t_5\}$$
$$d_6 = \{t_1\}$$

The corresponding inverted index structure L is:

$$L = \{l_1, l_2, l_3, l_4, l_5\}$$

$$l_1 = \{d_2, d_4, d_6\}$$
$$l_2 = \{d_1, d_2\}$$
$$l_3 = \{d_3, d_5\}$$
$$l_4 = \{d_2, d_4\}$$
$$l_5 = \{d_1, d_3, d_5\}$$

3.2 Replication and Index Partitioning

Traditionally, the whole inverted index used to be stored on one machine. Despite the simplicity of this approach, it suffers from the drawbacks of having slow response time as well as space limitation. These problems become significantly apparent with large volumes of data.

The two most popular approaches for exploiting parallelism in query processing are *replication* and *index partitioning*. It is assumed here that there are n index nodes. Using the replication approach, a replica of the index is assigned to each of the n nodes. In this case, multiple queries can be processed in parallel but each query is processed sequentially. This approach is also called *inter-query parallelism*. Using the index partitioning approach, the index is partitioned into n partitions each of which is assigned to a separate node. Each query is processed by multiple nodes in parallel where each node works only on its index partition. This approach is also called *intra-query parallelism* and is known to be latency-oriented as it aims at reducing the average query waiting time. On the other hand, inter-query parallelism is known to be throughput-oriented. Partitioning the index can be performed in one of two dimensions; can be based on either documents or terms.

3.3 Document-Based and Term-Based Partitioning

Based on the example document collection given in section 3.1, document-based and term-based portioning are illustrated in Figure 1. In document-based partitioning, each node is responsible for a subset of the document set. The index on each node consists of lists of all terms of documents belonging to the corresponding document partition. When a new query is to be processed, it is passed to a master (frontend) node which sends the query to all the index nodes. Each index node processed the query based on the index partition it has then passes the results to the master node. The master node merges all the results passed by the index servers. The main advantage of such a scheme is its simplicity (Büttcher, Clarke, & Cormack, 2010). In term-based partitioning, each node is responsible for a subset of terms. A node processes a query only if at least one of the terms in the query belongs to the node's terms subset. This scheme is preferred in cases where the index is stored on disks. However, performance degrades as the document collection becomes bigger. So, it is not scalable with the size of the document collection. Term partitioning also suffers from load imbalance since the load associated with a term depends on its frequency in the document collection as well as its frequency in queries. Some hybrid term/document partitioning and hybrid document partitioning and replication have been devised.

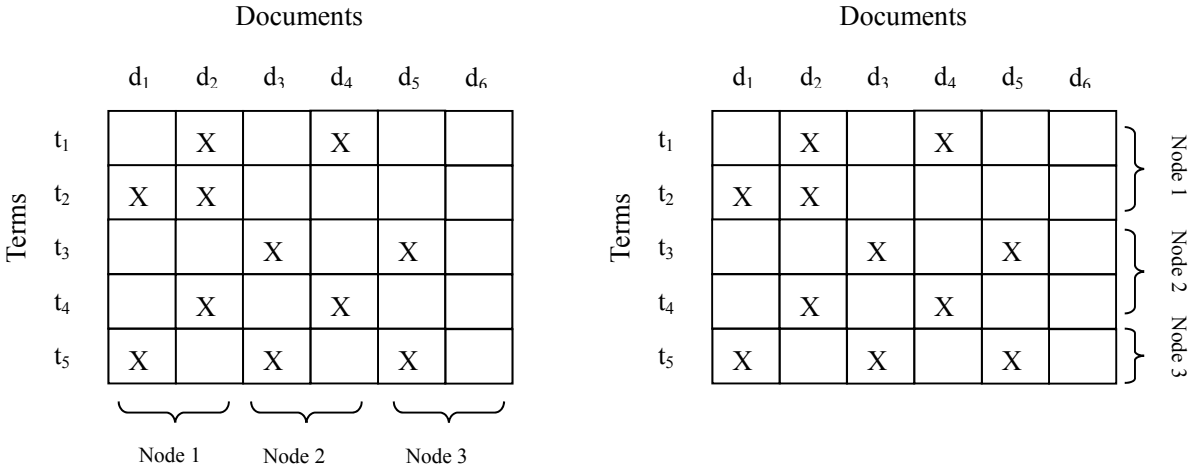

Figure 1. Document-based partitioning and Term-based partitioning

4. Modeling Parallel Query Processing

In this paper, we exploit parallelism in query processing making use of the inverted list structure on both the shared-memory parallel model and parallel clusters. Both architectures are illustrated in Figure 2.

4.1 Inverted Index

An inverted index is a data structure which is commonly used to index large document collections for fast query processing. For a document collection D associated with a set of terms T, an inverted index is composed of a set of inverted lists L. An inverted list $l_i \in L$ is the inverted list for the term $t_i \in T$. Every document $d_k \in D$ is associated with a number of terms such that $d_k \subset T$. For our example document collection $D = \{d_1, d_2, d_3, d_4, d_5, d_6\}$ and the set of terms $T = \{t_1, t_2, t_3, t_4, t_5\}$.

4.2 Inverted Index Partitioning

The previous definitions are slightly modified and extended for the purpose of partitioning the inverted index. Let N be the set of nodes and L_j the set of inverted lists assigned to the node $n_j \in N$. Each inverted list $l_{ij} \in L_j$ is associated with the term t_i and stored on node n_j. Hence, $L_j = \{l_{1j}, l_{2j}, \dots, l_{|T|j}\}$. Each inverted list l_{ij} consists of links to a number of documents such that $l_{ij} \subset D$.

Using the term-based partitioning approach, the set of terms T is partitioned among the nodes such that each term exists on only one node. Therefore, for any two inverted lists l_{ij} and l_{im}, $j \neq m$ where $1 \leq j, m \leq |N|$ and $1 \leq i \leq |T|$. An inverted list l_{ij} consists of links to documents associated with terms assigned to n_j. Hence, $l_{ij} = \{d_k : t_i \in d_k, 1 \leq k \leq |D|\}$.

In document-based partitioning, the set of documents D is disjointly partitioned among the nodes such that all links to documents in an inverted list stored on a particular node appear only on this node. Let D_j be the set of documents assigned to n_j. Consequently, $D_j = \bigcup_{i=1}^{|T|} l_{ij}$ and $d_v \cap d_u = \emptyset \ \forall v \neq u$. It is possible that $l_{ij} = \emptyset$ if $t_i \in d_k$ and $d_k \notin D_j$.

Table 1 shows how this model can be applied on our simple example using both document-based and term-based partitioning using 3 nodes.

Table 1. Document-based and term-based index partitioning applied on our simple document collection using 3 nodes

Node	Document-based Partitioning	Term-based Partitioning
Node 1	**L_1 = { l_{11}, l_{21}, l_{41}, l_{51}}**	**L_1 = { l_{11}, l_{21}}**
	l_{11}= {d_2}	l_{11}= {d_2,d_4,d_6}
	l_{21}= {d_1,d_2}	l_{21}= {d_1,d_2}
	l_{41}= {d_2}	
	l_{51}= {d_1}	
Node 2	**L_2={ l_{12}, l_{32}, l_{42}, l_{52}}**	**L_2={ l_{32}, l_{42}}**
	l_{12}= {d_4}	l_{32}= {d_3,d_5}
	l_{32}= {d_3}	l_{42}= {d_2,d_4}
	l_{42}= {d_4}	
	l_{52}= {d_3}	
Node 3	**L_3={ l_{13}, l_{33}, l_{53}}**	**L_3={ l_{53}}**
	l_{13}= **{**d_6**}**	l_{53}= {d_1,d_3,d_5}
	l_{33}= **{**d_5**}**	
	l_{53}= **{**d_5**}**	

4.3 Using the Shared-Memory Model

Queries are partitioned among nodes in order to exploit parallelism. A query q is represented as a set of terms $q \subset T$. In the shared memory model of N nodes, a query q is processed sequentially by a node n_j. When a query arrives, it is assigned to an available node or waits in the queue if all nodes are busy. The number of inverted lists accessed by q is $|q|$ as every term maps to an inverted list. Let D^q be the set of documents retrieved by q. Then, $D^q \subset D$ such that $\forall t_i \in q$, $t_i \in d_k$ and $d_k \in D^q$ where $1 \leq i \leq |q|$, $1 \leq k \leq |D^q|$.

4.4 Using Clusters

Using the parallel cluster model, every query is assigned to an entire cluster. Within a cluster, terms of a query are partitioned among the nodes in the cluster. Each node in a cluster receives a subset of the terms of the query assigned to the cluster. All nodes in a cluster run in parallel and the documents retrieved by all nodes are compiled. This can be viewed as a hybrid approach between inter-query and intra-query parallelism. This partitioning approach adopts inter-query parallelism on the level of clusters whereas it adopts intra-query parallelism on the level of nodes within a cluster. Let C be the set of clusters and each cluster $c_r \in C$ where $1 \leq r \leq |C|$ consists of a set of nodes. Hence, $c_r = \{n_{1r}, n_{2r}, \dots, n_{|N_r|r}\}$ where N_r is the set of nodes assigned to cluster c_r. If a query q is assigned to cluster c_r, each node $n_{jr} \in c_r$ $1 \leq j \leq |N_r|$ processes approximately $|q|/|N_r|$ terms. A cluster is considered to be available for another query when all nodes finish.

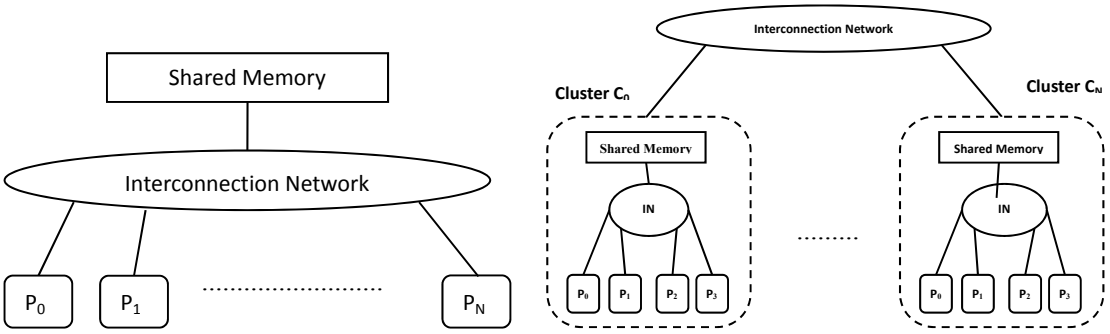

Figure 2. Shared-memory and parallel clusters models

5. Simulation

For the purpose of this work, a specialized simulator was developed. The parameters and their settings are explained. Some values are randomly generated in selected ranges that were found adequate. In our simulator, we are interested particularly in simulating query processing on shared-memory architecture and cluster-based architectures. A performance analysis is conducted when different simulation parameters are varied. Our simulator consists of three main components; the *queries and inverted index generator*, *shared-memory simulator* and *cluster simulator*. Queries and index data generated from the first component are used as input to the two other components. Each of these components will be detailed in a separate subsection.

5.1 Queries and Inverted Index Generator

Before running the simulation experiments, the following two steps are performed to generate query and index files which are the main inputs to the simulation experiments. Simulation parameters that are input to both generators are:

-Total number of queries (Q_{total})

-Maximum number of terms /query ($Q_{Maxterms}$)

-Total number of terms in the index (I_{nterms})

-Maximum number of documents / term (T_{MaxDoc})

-Total number of documents (D_{total})

-Maximum Query Arrival time (Q_{MaxArr})

These are considered to be the primary simulation parameters. Some of the parameters' values are used as they are and some are used as maximum values for data to be generated as detailed in the following subsections.

5.1.1 Generating Queries

A list of queries is generated based on the total number of queries Q_{total} and the maximum number of terms/query $Q_{maxterms}$. Each query entry consists of:

Query ID (Q_{ID})

Query arrival time (Q_{Arr}): a random number from 0 to maximum arrival time (Q_{MaxArr}). For the purpose of varying the average query inter-arrival time among different simulation experiments, the value of Q_{MaxArr} is varied to be one of 4 values as indicated in Tables 2, 3 and 4.

Number of terms in query (Q_{nterms}): a random number from 1 to $Q_{maxterms}$.

Query terms list: a list of Q_{nterms} term ID's. A term ID is a numeric value from 1 to the total number of terms in the index (I_{nterms}).

5.1.2 Generating Inverted Index

A list of index entries is generated based on the total number of terms in the index I_{nterms}. An index entry holds data about one of the I_{nterms} terms in the index file. Each index entry consists of:

Term ID (T_{ID})

Number of documents related to the term (T_{ndoc}): a random number from 1 to T_{MaxDoc}.

Term documents list: a list of T_{ndoc} document ID's. A document ID is a numeric value from 1 to total number of documents D_{total}.

5.2 Query Processing on Shared-Memory Simulator

Queries are arranged in a queue according to their arrival time. At the beginning of each simulation time step, it is checked if there any waiting queries. Idle processors are cleared and marked as available. If there are waiting queries, an idle processor is sought for each query. When a query is assigned to a processor, that processor handles all terms in the query in sequence. All query terms are searched for in the inverted index to retrieve relevant document IDs. Document IDs are compiled and redundancies are removed.

5.3 Query Processing on Clusters Simulator

At the beginning of each simulation step, clusters which have become idle are cleared and marked as available. A cluster is considered available if all processors within the cluster have finished processing. When a waiting query is assigned to an available cluster, all processors in the cluster are dedicated to the query till the whole query is processed. Query terms are distributed among cluster processors and each processor searches for the document IDs relevant to the term being processed. At end, all retrieved document IDs are compiled and redundancies are removed.

6. Parameters and Experiments

In our simulation experiments, we use a synthetic data set that is generated by the Queries and Inverted Index Generators described earlier. The total number of queries (Q_{total}) input to our simulator is 1000 queries with a maximum number of keywords (terms) per query ($Q_{Maxterms}$) taken as 5. The actual number of keywords per query generated by the queries generator is a random integer from 1 to $Q_{Maxterms}$ inclusive. Each query has an identifier Q_{ID} which lies between 0 and Q_{total}. The inverted index consists of a list of entries each of which is associated with a keyword (term). The size of the generated inverted index (I_{nterms}) or the total number of terms is 300 terms. The document set is assumed to consist of D_{total} = 10000 documents. Each document in the document set is associated with a number of terms which is a random integer between 1 and T_{MaxDoc} inclusive where T_{MaxDoc}=15. A query's arrival time Q_{Arr} is a randomly generated integer between 0 and Q_{MaxArr} inclusive. Q_{MaxArr} is varied in 4 values (20, 30, 60 an 100) to simulate different query inter-arrival times.

A total of 32 simulation experiments are run using a single processor, shared-memory architecture and cluster – based architecture. The 4 sets of parameters (varying Q_{MaxArr}) are used in the simulation experiments for all architectures. In the shared-memory experiments, the number of processors is varied (4, 8, 16 and 32). In cluster-based experiments, the number of processors within a cluster is fixed to 4 in all experiments whereas the number of clusters is varied (2, 4 and 8).

7. Results and Discussion

A sample of the simulation output file is shown in Figure 3. A number of performance measures are computed and output by the simulation program to show the effect of varying Q_{MaxArr}, the parallel architecture used and the number of processors (N).

```
Simulation parameters
Number of shared memory processors 4
Number of queries= 1000
 Max.KW per query = 5
Number of queries = 300
 Max. docs per KW = 15
Number of docs=10000
Max. Arr. time=60.00
no.of docs in query 0 is 2 no of kw = 1
no.of unique docs in query 0 is 2
P0 processing query 0 arr. time= 3 start time=3 endtime=11 querytime=8
no.of docs in query 1 is 36 no of kw = 3
no.of unique docs in query 1 is 36
P1 processing query 1 arr. time= 4 start time=4 endtime=1372 querytime=1368
no.of docs in query 2 is 12 no of kw = 2
no.of unique docs in query 2 is 12
P2 processing query 2 arr. time= 8 start time=8 endtime=176 querytime=168
no.of docs in query 3 is 29 no of kw = 3
no.of unique docs in query 3 is 29
P0 processing query 3 arr. time= 14 start time=14 endtime=913 querytime=899
no.of docs in query 4 is 12 no of kw = 2
no.of unique docs in query 4 is 12
. . . . .
Summary
P0 busytime=1275.69 tkws=645 Percent util.=1.00
P1 busytime=1270.32 tkws=623 Percent util.=1.00
P2 busytime=1264.38 tkws=650 Percent util.=0.99
P3 busytime=1276.31 tkws=615 Percent util.=1.00
Avg. Proc. Util = 1.00
Avg. Query Response time = 590.08
Total waiting time for all queries=584997.03
Avg waiting time = 585.00
Total no. of kws for all processors=2533
Arrival time Mean=30.47 Variance=28841.95 SD=16.98
time taken (makespan)=1276.50
```

Figure 3. Part of a log file and simulation output of a simulation experiment

The results of a number of performance measures are shown in Tables 2, 3 and 4 using one processor, shared-memory and clusters respectively. The maximum query arrival time Q_{MaxArr} is varied to vary the average query inter-arrival time while keeping the total number of queries constant as explained before. The average query response time decreases with the increase in Q_{MaxArr}. Average processor and cluster utilization are measured as a fraction between 0 and 1. A cluster is busy from the time the first processor in the cluster starts processing till the last processor in the cluster finishes. Consequently, a cluster is considered occupied processing queries even though a number of processors within the cluster may be idle. Therefore, average processor utilization in case of shared-memory is high while average cluster utilization is relatively low and reaches 0.39.

Table 2. Performance measures for simulation results with N=1 varying the maximum query arrival time

Performance Measure	Q_{MaxArr}=10	Q_{MaxArr}=30	Q_{MaxArr}=60	Q_{MaxArr}=100
Makespan	5383.50	5228.73	5086.70	4635.52
Average Query Response Time	2684.77	2641.42	2450.13	2250.75
Average Query Waiting Time	2679.39	2636.20	2445.04	2246.12

Table 3. Performance measures for shared-memory simulation results varying the number of processors N and the maximum query arrival time Q_{MaxArr}

Performance Measure	N	Q_{MaxArr}=10	Q_{MaxArr}=30	Q_{MaxArr}=60	Q_{MaxArr}=100
Makespan	4	1347.78	1309.07	1276.50	1166.69
	8	676.86	655.77	641.12	589.31
	16	342.49	330.84	326.30	302.50
	32	174.72	171.63	172.59	159.61
Average Processor Utilization	4	1.00	1.00	1.00	0.99
	8	0.99	1.00	0.99	0.98
	16	0.98	0.99	0.97	0.96
	32	0.96	0.95	0.92	0.91
Average Query Response Time	4	664.18	649.52	590.08	524.98
	8	327.35	317.66	280.09	237.46
	16	159.04	151.75	125.47	93.97
	32	74.95	69.01	48.76	23.54
Average Query Waiting Time	4	658.80	644.29	585.00	520.35
	8	321.96	312.43	275.01	232.83
	16	153.65	146.53	120.38	89.35
	32	69.57	63.78	43.67	18.91

Table 4. Performance measures for clusters simulation results varying the number of clusters C and the maximum query arrival time Q_{MaxArr}. The number of processors per cluster is kept constant and is equal to 4

Performance Measure	C	Q_{MaxArr}=10	Q_{MaxArr}=30	Q_{MaxArr}=60	Q_{MaxArr}=100
Makespan	2	775.79	777.31	738.11	722.78
	4	388.66	389.17	369.42	361.97
	8	194.96	194.71	185.72	181.55
Average Cluster Utilization	2	1.00	1.00	1.00	1.00
	4	1.00	1.00	1.00	1.00
	8	0.99	1.00	0.99	0.99
Average Processor Utilization	2	0.40	0.40	0.40	0.39
	4	0.40	0.40	0.40	0.40
	8	0.40	0.40	0.40	0.39
Average Query Response Time	2	379.22	376.45	332.54	308.70
	4	185.03	181.17	151.40	129.43
	8	87.95	83.57	60.94	39.87
Average Query Waiting Time	2	377.67	374.90	331.06	307.26
	4	183.48	179.62	149.92	127.99
	8	86.40	82.02	59.47	38.42

Speedup is a performance measure that indicates the gain from parallelism. It is measured as a fraction $S_N=T_1/T_N$ where T_1 is the time taken using one processor and T_N is the time taken using N processors. Efficiency is also another performance measure that shows how efficiently processors are utilized and is measured as $E_N=S_N/N$. Table 5 shows that both speedup and efficiency are higher in case of shared-memory than in case of clusters.

Table 5. Speedup and Efficiency in shared memory and clusters experiments

Shared-Memory Experiments			Clusters Experiments		
N	Average Speedup	Efficiency	C	Average Speedup	Efficiency
4	3.9	97.5%	2	6.7	83.8%
8	7.8	97.5%	4	14.7	91.9%
16	15.5	96.9%	8	26.8	83.8%
32	29	90.6%			

The effect of changing the parallel architecture on query response time while keeping the total number of processors constant has been investigated (Figures 4, 5 and 6). In case of cluster experiments, the number of processors per cluster is kept constant and equals to 4). The number of queries which achieve lower response time is higher when using shared-memory compared to clusters.

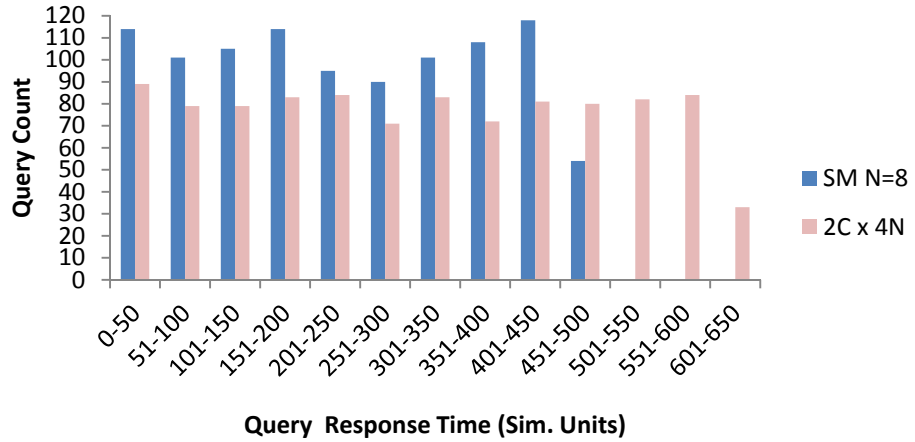

Figure 4. Comparison between shared-memory architecture and clusters having the same number of nodes (8 nodes) showing the number of queries achieving specified ranges of response time

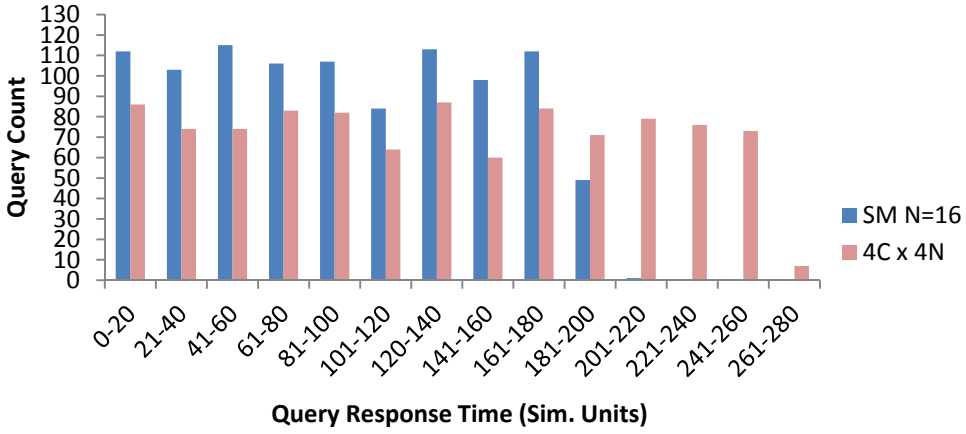

Figure 5. Comparison between shared memory architecture and clusters having the same number of nodes (16 nodes) showing the number of queries achieving specified ranges of response time

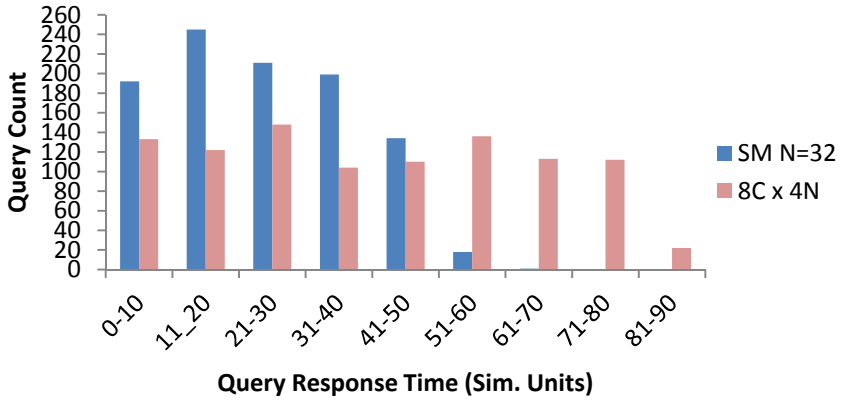

Figure 6. Comparison between shared memory architecture and clusters having the same number of nodes (32 nodes) showing the number of queries achieving specified ranges of response time

Histograms of Figures 7 and 8 show further details on query count of queries achieving different response times for shared-memory and cluster models respectively. In these histograms, query response time is divided into intervals where each column represents the query count in a specific range (between 2 bins). When comparing, for example, the second histogram (N=8) in Figure 7 with the first histogram (C=2) in Figure 8, both having 8 processors and the same bin ranges, it is noted that the number of queries in case of using shared-memory is higher than that in case of using clusters in the same bin range. Similarly, other pairs of histograms can be compared with the same observation.

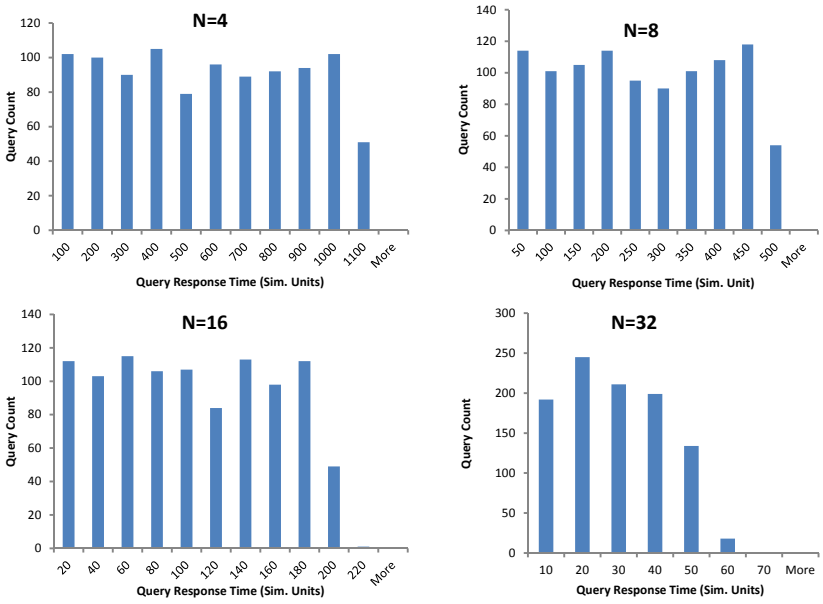

Figure 7. Histograms showing the number of queries having response times in different ranges in shared memory simulation. Response time is measured in simulation time steps. N is the number of processors

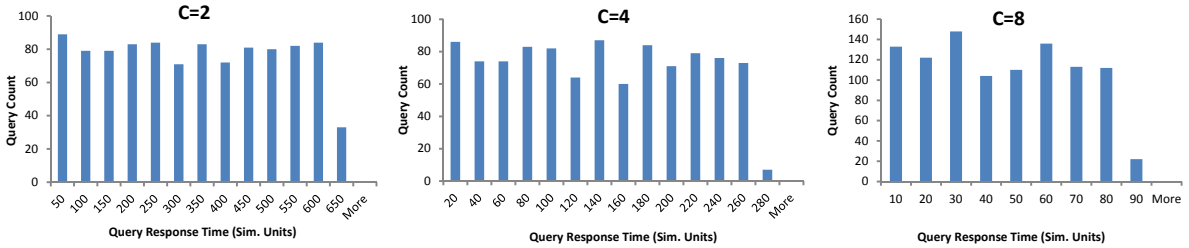

Figure 8. Histograms showing the number of queries having response times in different ranges in clusters simulation. Response time is measured in simulation time steps. C is the number of processors where the number of processors per cluster is fixed to 4

Figure 8 represents a total view or log for the actual number of queries of every possible response time. The execution profile (Figure 10) of a number of experiments shows the effect of changing Q_{MaxArr} on the percentage of queries processed as simulation time passes (also represented as a percentage of total simulation time). This figure represents 4 experiments for C=8 varying Q_{MaxArr} and shows that a higher percentage of queries are processed earlier as Q_{MaxArr} increases. This is due to the fact that with higher values for Q_{MaxArr}, average query inter-arrival time increases and hence, queries do not have to wait as long as in case of lower Q_{MaxArr}.

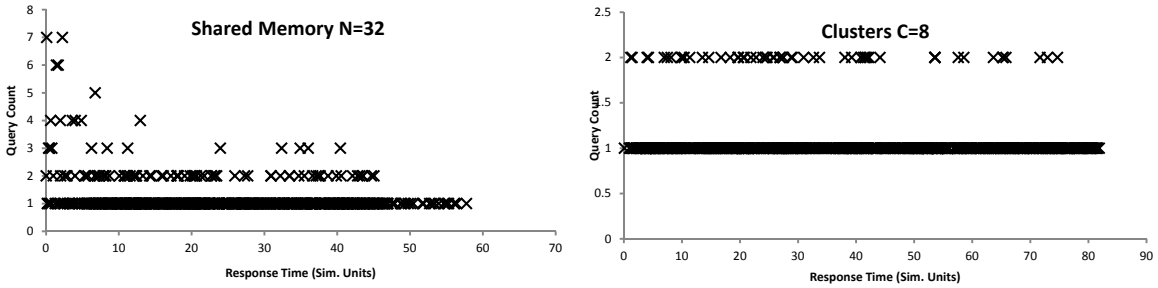

Figure 9. Example simulation experiments showing the number of queries of every possible response time

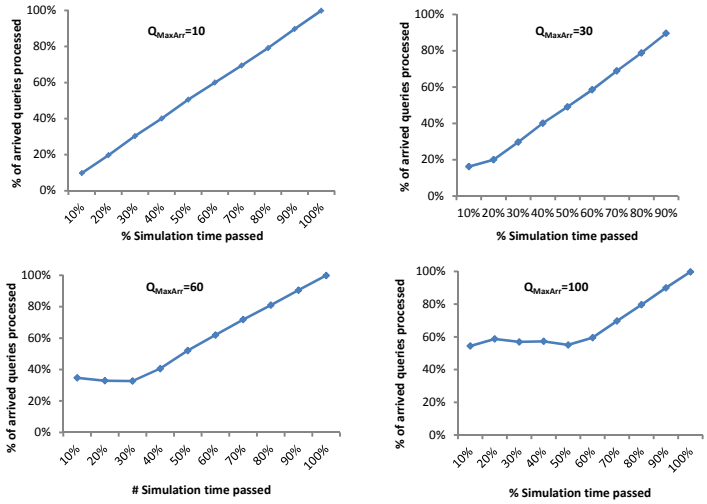

Figure 10. Percentage queries processed versus the percentage of simulation time passed when C=8 varying maximum arrival time in each graph

References

Baeza-Yales, R., & Ribeiro-Neto, B. (1999). *Modern Information Retreival* (1st ed.). Addison Wesley, Longman.

Büttcher, S., Clarke, C. L. A. & Cormack, G. V. (2010). Parallel Information Retrieval . In *Information Retreival: Implementing and Evaluating Search Engines*, 492-510. MIT Press.

Brin, S., & Page, L. (1998). The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems (Proceedings of the Seventh International World Wide Web Conference), 30*(1-7), 107-117. http://dx.doi.org/10.1016/S0169-7552(98)00110-X

Cambazoglu, B., Catal, A., & Aykanat, C. (2006). Effect of inverted index partitioning schemes on performance of query processing in parallel text retrieval systems. *Lecture Notes in Computer Science, 4263*, 717-72. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/11902140_75

Chung, S., Kwon, H., Ryu, K., Chung, Y., Jang, H., & Choi, C. (2001). Information retrieval on an SCI-based PC cluster. *Journal of Supercomputing, 19*(3), 251-265. http://dx.doi.org/10.1023/A:1011178530932

Daily estimated size of the World Wide Web. (2013). Retreived May 9, 2013, from http://www.worldwidewebsize.com/

Delbrua, R., Campinas, S., & Tummarello, G. (2012). Searching Web Data: an Entity Retrieval and High-Performance Indexing Model. *Web Semantics: Science, Services and Agents on the World Wide Web, 10*, 33-58. http://dx.doi.org/10.1016/j.websem.2011.04.004

Ding, S., He, J., Yan, H., & Suel, T. (2009). Using Graphics Processors for High Performance IR Query Processing. In *Proceedings of the 18th International Conference on World Wide Web, WWW'09,* 421-430. ACM Press. http://dx.doi.org/10.1145/1526709.1526766

Epimakhov, I., Hameurlain, A., Dillon, T., & Morvan, F. (2011). Resource Scheduling Methods for Query Optimization in Data Grid Systems. In J. Eder, M. Bielikova, & A. M. Tjoa (Eds.), *Lecture Notes in Computer Science (ADBIS 2011),* (pp. 185-199). Springer-Verlag Berlin Heidelberg.

Frachtenberg, E. (2009). Reducing Query Latencies in Web Search Using Fine-Grained Parallelism. *World Wide Web, 12*(4), 441-460. http://dx.doi.org/10.1007/s11280-009-0066-4

Google Annual Search Statistics. (2013). Retreived May 9th, 2013, from http://www.statisticbrain.com/google-searches/

Hong, W., & Stonebraker, M. (1991). Optimization of Parallel Query Execution Plans in XPRS. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems, PDIS 1991,* 218-225. http://dx.doi.org/10.1109/PDIS.1991.183106

Kang, J., Ahn, H., Jung, S., Ryu, K., Kwon, H., & Chung, S. (2004). Improving load balance and fault tolerance for PC cluster-based parallel information retrieval. In *Proceedings of the 5th International Conf on Parallel Process and Applied Mathematics (PPAM 2003), Lecture Notes in Computer Science, 3019*, 682-687. Springer. http://dx.doi.org/10.1007/978-3-540-24669-5_89

Konstantopoulos, C., Mamalis, B., Pantziou, G., & GavalasHong, D. (2009). Efficient parallel Text Retrieval techniques on Bulk Synchronous Parallel (BSP)/Coarse Grained Multicomputers (CGM). *Journal of Supercomputing, 48*(3), 286-318. http://dx.doi.org/10.1007/s11227-008-0225-x

Mamalis, B., Spirakis, P., & Tampakas, B. (1999). Optimal High-Performance Parallel Text Retrieval via Fat-Trees. *Theory of Computing Systems, 32*(6), 591-623. http://dx.doi.org/10.1007/ s00224000013

Rungsawang, A., Laohakanniyom, A., & Lertprasertkune, M. (2001). Low-Cost Parallel Text Retrieval Using PC-Cluster. *Recent Advances in Parallel Virtual Machine and Message Passing Interface (Lecture Notes in Computer Science, 2131)*, 419-426. http://dx.doi.org/10.1007/3-540-45417-9_56

Singhal, A. (2001). Modern Information Retrieval: A Brief Overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 24*(4), 35-42.

Svenonius, E. (2000). *The Intellectual Foundation of Information Organization* (1st ed.). MIT Press.

Wen, J., Nie, J., & Zhang, H. (2001). Clustering User Queries of a Search Engine. In *Proceedings of the 10th International Conference on World Wide Web, WWW'01,* 162-168. ACM Press. http://dx.doi.org/10.1145/371920.371974

Zobel, J., & Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys, 38*(2), Article No. 6. http://dx.doi.org/10.1145/1132956.1132959