Simulation of Distributed Search Engines: Comparing Term, Document and Hybrid Distribution[1]

Andrew Kane
arkane@uwaterloo.ca

David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada.

Technical Report CS-2009-10
February 18, 2009

[1] This paper was originally created as a course project for 'CS 856 Advanced Topics in Distributed Computing - Performance Modeling and Analysis' taught by Professor Johnny Wong in Winter 2008.

Simulation of Distributed Search Engines: Comparing Term, Document and Hybrid Distribution

Andrew Kane
arkane@uwaterloo.ca

David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada.

ABSTRACT

A method of simulating distributed search engines is presented. This method measures throughput (queries per second) and resource usage. Disk and network costs are modelled in a simple way, while CPU costs are ignored.

Our simulation results show that term distribution can perform better than document distribution when using a small number of machines. This goes against the accepted view that document distribution is faster than term distribution.

We introduced a hybrid distribution scheme that splits a set of machines and disks into groups and then performs term distribution within a group and document distribution between groups. Our simulation results show that this hybrid distribution scheme has higher throughput rates than document distribution, but can still scale to large numbers of machines.

A special case of our hybrid distribution scheme groups together multiple disks on a machine to produce better throughput without increasing network traffic. Such a scheme could easily be deployed in a web search engine.

General Terms

Algorithms, Measurement, Performance, Design, Experimentation.

Keywords

Search Engine, Information Retrieval, Term Distribution, Document Distribution, Simulation, Modelling.

1. INTRODUCTION

Digital information continues to be created at an ever increasing rate. In 2007, the amount of information created, captured, or replicated exceeded available storage for the first time [1]. As a result, search engines are becoming increasingly important for locating specific information. In addition, search engines are increasing in size while, at the same time, the number of searches are increasing [2].

Distributed search engine performance is increasingly important for very large engines (web search) and for large engines (enterprise search). Such systems provision their hardware relative to max throughput estimations, within some acceptable query latency.

We examine the distribution of data structures within a distributed search engine. Specifically we simulate term and document distribution [3], as well as, a hybrid distribution scheme.

2. SEARCH ENGINE BACKGROUND

A search engine essentially takes a query, identifies the set of objects that match the query, ranks the objects and presents the top-k objects [15]. For most search engines, the objects are documents and the value of k is small.

Search engines have special purpose data structures and algorithms designed to execute queries with low latency and high throughput. In distributed search engines, parallelization is highly exploited to improve performance.

Queries are usually just words or phrases combined together with the default AND operators. As a result, search engines often treat text input as a sequence of words (tokens) in a process called tokenization. Queries can also be augmented with other operators and modifiers depending on what is supported by the search engine.

Search engines try to present the 'best' results. This is done with a ranking algorithm which can use many pieces of information from the documents and the query.

2.1 Indexing

The process of taking objects and constructing the data structures (indexes) required by the search engine is called indexing. This process has to 'read' objects from some source. This source could be a cache (Google has a cache of the web), a set of files (desktop search), or some other source (extracted from a database or data repository).

These objects are converted to text, often removing layout and display information at the same time. The text is tokenized into a sequence of words (tokens) and postings lists are produced (see Section 2.2). The tokenizer usually normalizes tokens by converting them to lower case (case-folding) and removing character modifications like accents.

Objects are usually grouped into batches to produce a set of postings lists called a subindex. These subindexes can be combined together to produce larger subindexes through a process called merging.

In systems that allow updates to objects without re-indexing the entire data set, these subindexes would also contain a list of deleted objects known as a delete mask.

The most common words, such as 'the', appear in almost every object and, therefore, have little meaning for a query. These are called stopwords and are often removed from the index.

2.2 Postings Lists

Each subindex contains a postings list for every token it contains. A postings list contains the location of all occurrences of its token within the set of objects found in the subindex. These occurrences are usually stored in order by object which allows them to be combined sequentially when a query is executed.

Some search engines produce two levels of postings lists, one at the object level and one at the occurrence location (offset) level.

Postings lists are stored in a compressed format, usually using delta encoding [9] and byte or nibble aligned variable length integer compression [10]. Object level postings lists may also use run-length encoding [11] which works well for high occurrence tokens.

2.3 Query Definition

A query is essentially a sequence of tokens or phrases with modifiers that are combined together using boolean operators. In web search engines, very few modifiers are used, phrases are rare and most queries use the default AND operator. So, for performance analysis, queries are often assumed to be tokens combined with AND operators.

2.4 Ranking

Ranking algorithms use document and query information to order the objects that match a given query. Some search engines allow the ranking algorithm to be specified or tuned at query time, some on a per install basis, and others use one static algorithm. Web search engines combine multiple ranking sub-algorithms together to form an overall ranking algorithm, and they modify it over time to produce the best results. This is essentially their 'secret sauce' and the specifics are closely guarded.

For web search engines, improving the ranking for a page can be very valuable to the owner of that page, thus many companies offer to do just that for a fee. The web search engine companies want their results to be unbiased, so they continually change their ranking algorithm to combat this 'cheating'.

2.5 Result Pages

The search results page displays the top-k ranked objects computed by executing the specified query. The results page usually includes metadata for each object and a summary of each object tuned to the given query. The retrieval of metadata and creation of the summary do not use the search engine postings lists.

2.6 Term vs. Document Distribution

In order to scale a search engine for a large number of objects and/or a large number of searches, a distributed system is used. There are two main methods of distributing postings lists across machines, term distribution and document distribution (a.k.a. global distribution and local distribution respectively) [3].

Term distribution puts the entire postings list for a term on a single machine or disk drive. To execute a query, postings lists must often be transferred across the network.

Document distribution separates the documents into partitions and puts all the postings lists for a partition on one machine or disk. Each partition is essentially a self contained search index. This method has low network usage but large amounts of random disk reads.

It is generally accepted that document distribution is faster than term distribution.

3. SEARCH ENGINE MODELLING

Distributed search engines are very complex systems, with few general standards for interfaces or implementations. They involve low level interactions and built-in tradeoffs between hard disk, memory, CPU and network usage.

Such a system is very hard to model if one of the desired metrics is query latency. This is one of the reasons that simulation [6, 12] and modelling [7, 13] of search engines is not common in the literature.

Experiments on search engines are very common in the literature. They often use an existing collection of objects and queries as done in the TREC conference [4]. Sometimes the queries are generated from a model.

In this paper we model a search engine to measure query throughput by assuming that each unit of 'work' (CPU usage, disk access, network transfer) is independent and can be executed out of order. Each resource (CPU, disk, network) is modelled as a queue of work units which are processed in order. When a query is started, all the work units needed to 'execute' that query are put onto the appropriate resource queue for processing. A query is finished when all its work units have been executed.

By assuming that work units are independent and that we are measuring query throughput, we are essentially identifying the throughput of the bottleneck resource. Usage rates for resources are calculated to identify which resource is the bottleneck.

3.1 Postings List Modelling

Text data usually follows a Zipf power law distribution [5]. We use a similar distribution which was presented as Z(j) in equation 1 from [6]. In our case, we use a different value for the number of unique tokens (T). The distribution is:

$$Z(j) = j^{-0.0752528*\ln(j)-0.150669} * e^{16.3027} / S$$

Where S is the summation of the numerator over all terms. The terms are sorted by decreasing Z(j) value (highest first). The function Z(j) represents the probability that any token in the input data is the j^{th} token.

This is encoded in an array Y[j] and modified as specified below:

Stopwords are modelled by setting Y[j] = 0 for the first W tokens, which are the highest occurring tokens.

Compression is assumed to be delta encoded, byte aligned variable length integers. The average number of bits required to represent each posting in the postings list can be approximated by determining how many bits (B) it would take to represent the token if all occurrences were evenly spaced within the data set. The minimum number of bits for byte encoding is 7 (the 8th bit is used to specify if more bytes are needed). We calculate B by multiplying Z(j) by 2 until it is greater than 1, then the number of multiplications by 2 is the number of bits:

```
int bits(double x = Z(j)) {
   int result = 7;
   x *= 128;
   for (; x < 1; x*=2, result++);
   return result;
```

```
}
```

The values in Y[j] are multiplied by the approximation of their average byte count:

```
for (j in tokens)
  Y[j] *= bits(Z(j)) / 7;
```

Then the values in Y[j] are divided by the summation of all values in Y[j].

The values in Y[j] now represent the portion of the index used by the postings list for each token:

Postings list size for j = Y[j] * (total size of index)

This is essentially the size of the postings list if it was not split into object and offset postings. It is unclear how to model the size of the object postings list separately, so we approximate it as Y[j]/10. This does not take into account the difference in compression or the distribution of tokens in the text.

3.2 Query Modelling

A query is modelled as a sequence of tokens combined with AND operators. This model is made up of two distributions, one for the number of tokens and one for token values. The token values are used to map between a token in a query and the size of its postings list as specified in Y[j].

The distribution for the number of tokens in a query is dependent on the usage of the system being modelled [7, 8] and it is skewed depending on the types of caching being used [8]. We use a distribution derived from the 'result caching' portion of Figure 6.2 in [8] as specified in Chart 1.

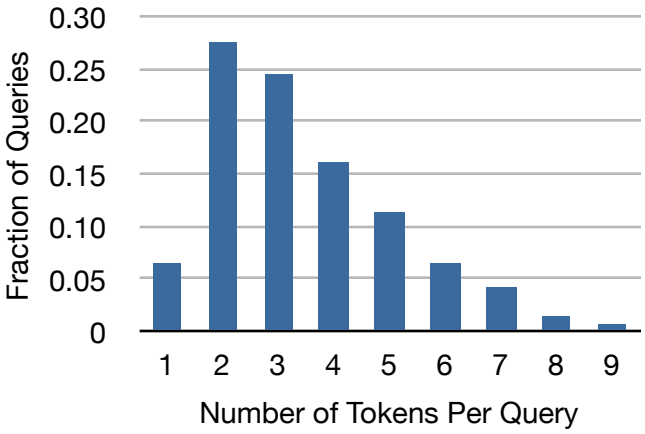

Chart 1. Distribution of tokens per query

The distribution of token values in queries does not follow the same distribution as tokens in text data, as specified in the previous section. There is no standard distribution, so we use the distribution specified as Q(k) from [6] with u=1% and a different value for the number of unique tokens (T):

$$Q(k) = 1/(uT) \text{ for first } uT \text{ tokens, } 0 \text{ otherwise}$$

This distribution function is converted into a cumulative distribution function which can be used to map a random number between 0 and 1 into a token index used to access Y[j].

3.3 Query Generator and Entry Control

The query generator creates queries at a specified rate using an exponential distribution. If the query generator produces queries faster than they can be executed, then the resource queues could grow without bound. To prevent this, only a certain number of queries (E) are allowed to be running concurrently. This is enforced by dropping queries as needed. A query is running when any of its work units are being executed or are waiting on a resource queue.

This setup allows resource usage rates to be examined for specific query arrival rates. It can also determine the maximum throughput of the system by setting the query arrival rate higher than the system can execute.

3.4 CPU Resources

Compression and decompression of postings lists use only a few operations per byte. A modern CPU can execute 3 billion operations per second and a modern hard disk drives (HDD) can stream 60 MB per second giving 50 operations per byte. This means that compression and decompression are negligible and can be ignored.

Since ranking algorithms are implemented in many different ways most of which are not publicly available, it is unclear how to model the CPU cost of ranking. Rather than guessing at the CPU cost of ranking, we assume that CPU resources are not the bottleneck and ignore the CPU work entirely.

3.5 Disk Resources

To execute a query, the search engine reads the postings lists for the tokens in the query sequentially within each postings list, but these reads are interleaved between postings lists. Using large buffers reduces the number of random reads, and prefetching reduces the query latency. If enough memory is available, each postings list involved in the query can be read into memory sequentially and combined afterwards, resulting in better throughput but worse query latency.

Since we are examining throughput, we optimistically model disk accesses as a single random read, followed by a sequentially read of an entire postings list. Modern HDDs can execute approximately 100 random reads per second and sequentially read 60 MB per second.

Since we are dealing with very large indexes, we assume that all disks are entirely full and contain only index data. Each HDD is 300 GBs, and each machine contains only one disk.

3.6 Network Resources

Networks of machines can be configured in different ways using many different types of hardware. For simplicity we assume that all the machines are connected with a single 1Gbps shared connection able to operate at 100% capacity and that each transmission has no overhead or setup cost. This configuration means the network can transfer 125 MB per second which is approximately twice as fast as a HDD, but without the random read costs.

3.7 Term vs. Document Distribution

For document distribution we assume that every disk contains an equal portion of every postings list. The network communications costs for combining the results from multiple machines together is small and therefore ignored in this simulation.

For term distribution we assume that each postings list occurs on only one disk. This is simulated by randomly picking a disk containing the postings list for each token in the query. For multiple tokens in a query, the postings lists are transferred across the network if they occur on a machine other than the machine containing the largest postings list used in the query. As before, any other network communications are ignored in this simulation.

4. SIMULATION RESULTS

The simulation is event-driven and implemented in Java using JavaSim [14] from the Department of Computer Science, University of Newcastle upon Tyne, UK.

The simulation was set up as follows: Each run simulates the search engine for 1000 seconds. The number of stopwords (W) is 100. The number of unique tokens (T) is 1,000,000. The maximum number of concurrent queries (E) is 100. All simulations were repeated 10 times. For the experiments presented, each simulated machine contains only one disk.

In the results presented below, the following abbreviations are used:

- M = Number of machines
- QPS = Queries per second (averaged over runs)
- STDEV = Standard deviation of QPS
- RR = Random disk reads relative to maximum
- SR = Streaming disk reads relative to maximum
- D = Overall disk usage (combines RR and SR)
- N = Network usage

Note: The performance numbers for term distribution and document distribution should be the same when using one machine. This can be used as a verification of our simulation results.

Table 1. Full postings lists using term distribution

M	QPS	STDEV	D	RR	SR	N
1	0.693	0.027	99%	2%	97%	0%
2	0.687	0.040	100%	1%	98%	17%
3	0.687	0.053	99%	0%	98%	34%
4	0.645	0.037	98%	0%	97%	51%
5	0.667	0.024	98%	0%	98%	68%
6	0.660	0.036	97%	0%	97%	84%
7	0.643	0.041	96%	0%	96%	96%
8	0.621	0.025	95%	0%	94%	99%
9	0.556	0.024	90%	0%	89%	99%
10	0.505	0.038	83%	0%	83%	99%

Table 2. Full postings lists using document distribution

M	QPS	STDEV	D	RR	SR	N
1	0.681	0.036	99%	2%	97%	0%
2	0.694	0.030	99%	2%	97%	0%
3	0.692	0.029	99%	2%	97%	0%
4	0.700	0.019	99%	2%	97%	0%
5	0.685	0.022	99%	2%	97%	0%
6	0.688	0.022	99%	2%	97%	0%
7	0.696	0.031	99%	2%	97%	0%
8	0.697	0.022	99%	2%	97%	0%
9	0.701	0.029	99%	2%	97%	0%
10	0.675	0.040	99%	2%	97%	0%

When using the full postings lists to compute queries, we expect the postings lists to be large. This results in the disk streaming read (SR) usage rate being a bottleneck as shown in Table 1 and Table 2.

The term distribution scheme essentially reduces the random reads (RR) by increasing the network usage (N). Since the RR usage is very small when using full postings lists, the term distribution does not improve the throughput (QPS) rate.

As shown in Table 1, when using 8 or more machines, the network becomes the bottleneck for the term distribution scheme.

Table 3. Object postings lists using term distribution

M	QPS	STDEV	D	RR	SR	N
1	5.709	0.068	99%	19%	80%	0%
2	6.203	0.099	99%	10%	88%	14%
3	6.393	0.096	97%	7%	90%	28%
4	6.439	0.127	97%	5%	91%	44%
5	6.502	0.147	96%	4%	92%	58%
6	6.345	0.099	94%	3%	90%	72%
7	6.344	0.110	94%	3%	91%	87%
8	6.266	0.094	91%	2%	89%	99%
9	5.622	0.116	82%	2%	80%	99%
10	4.972	0.112	73%	1%	71%	99%

Table 4. Object postings lists using document distribution

M	QPS	STDEV	D	RR	SR	N
1	5.702	0.074	99%	19%	80%	0%
2	5.667	0.076	99%	19%	80%	0%
3	5.733	0.083	99%	19%	80%	0%
4	5.704	0.041	99%	19%	80%	0%
5	5.671	0.069	99%	19%	80%	0%
6	5.647	0.040	99%	19%	80%	0%
7	5.721	0.067	99%	19%	80%	0%
8	5.694	0.047	99%	19%	80%	0%
9	5.642	0.040	99%	19%	80%	0%
10	5.675	0.042	99%	19%	80%	0%

When using object postings lists to compute queries, we expect the postings lists to be much smaller. This results in the QPS rate being higher, the RR usage being larger and the SR usage being smaller.

The performance rates stay constant when machines are added in the document distribution case, as shown in Table 4.

When the number of machines is increased in the term distribution case, performance rates change as shown in Table 3. The RR usage rates decrease and the network usage rates increase until it becomes a bottleneck at 8 machines. The throughput (QPS) rate increases initially, then decreases substantially when the network becomes the bottleneck.

These numbers show that term distribution can be faster than document distribution when using a small number of machines. This goes against the standard accepted assumption that document distribution is faster.

Unfortunately, term distribution does not scale to large numbers of machines. We introduce a hybrid distribution scheme to overcome this. Our hybrid scheme splits the set of machines into groups each containing a small number of machines. Document distribution is used between the groups, but term distribution is used within each group.

Table 5. Object postings lists using hybrid distribution with group size of 2

M	QPS	STDEV	D	RR	SR	N
2	6.206	0.093	98%	10%	88%	13%
4	6.166	0.082	99%	10%	88%	28%
6	6.124	0.065	98%	10%	87%	42%
8	6.172	0.072	98%	10%	87%	56%
10	6.210	0.096	98%	10%	87%	70%

Table 6. Object postings lists using hybrid distribution with group size of 3

M	QPS	STDEV	D	RR	SR	N
3	6.353	0.082	97%	7%	90%	28%
6	6.390	0.085	97%	7%	90%	57%
9	6.294	0.109	96%	7%	89%	85%

Table 7. Object postings lists using hybrid distribution with group size of 4

M	QPS	STDEV	D	RR	SR	N
4	6.392	0.097	96%	5%	91%	43%
8	6.363	0.094	96%	5%	90%	86%

The results found in Table 5 to 7 show that the hybrid distribution scheme has faster throughput (QPS) rates than the document distribution scheme and lower network usage rates than the term distribution scheme. The network communication between groups in the hybrid scheme is negligible. If a hierarchical network structure is used, the hybrid scheme could scale to large numbers of machines.

This simulation assumes that each machine contains only one disk drive. If multiple disks are used on each machine, the hybrid distribution scheme can be used to group all the disks on a machine. Such a grouping would produce the throughput increase but it would not incur additional network traffic since the disks in the group are on the same machine. Such a setup would be somewhat more fragile than document distribution since a failure on one disk would prevent the other disks on that machine from continuing to service queries. For highly redundant systems, such as those used for web search engines, this fragility would not be an issue. Such a machine could also be reused as a group with a smaller number of disks, after reconstituting the index.

For the object postings list example, the setup with the best throughput appears to be the hybrid distribution using groups of 4 machines. This is illustrated as the purple line in Chart 2. When scaling that scheme, no more than two groups (8 machines in total) should be placed on the same network switch/router/hub, to prevent the network from becoming a bottleneck.

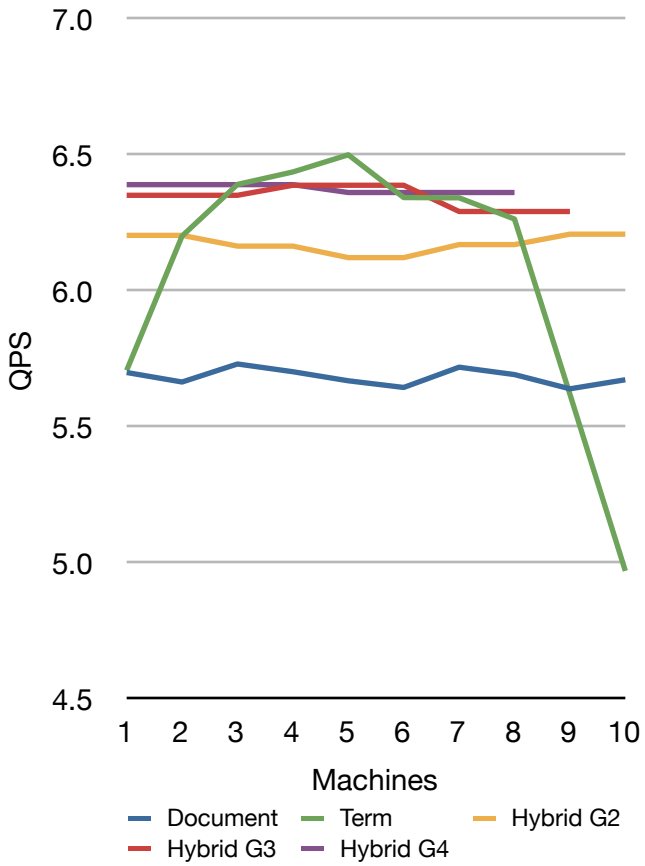

Chart 2. QPS throughput rates for term, document and hybrid distribution schemes

5. CONCLUSION

Our results from simulating distributed search engines show that term distribution can have higher throughput than document distribution when using a small number of machines. This goes against the accepted view that document distribution is faster than term distribution.

We introduced a hybrid distribution scheme that splits a set of machines and disks into groups and then performs term distribution within a group and document distribution between groups. Our simulations show that this hybrid distribution has higher throughput rates than document distribution, but can still scale to a large number of machines.

A special case of our hybrid distribution scheme groups together multiple disks on a machine to produce better throughput without increasing network traffic. Such a scheme should be examined for deployment in large search engines, especially for web search engines.

6. FUTURE WORK

The models used for disk access and network usage are very simplistic therefore more complex models should be examined. The CPU work is ignored in this simulation, thus the CPU usage of existing search engine should be examined and hopefully modelled.

The distribution of tokens in queries used in this model is rather arbitrary. Other distributions should be examined to see if similar results are produced. Also, query traces should be examined in the hope of new models being proposed.

Other distribution for the number of tokens in a query should be examined as this greatly affects network usage rates for the term distribution and hybrid distribution schemes.

The distribution used for object postings list sizes is not accurate therefore existing search engine installs should be examined to produce better distributions.

7. REFERENCES

[1] IDC Report: "The Diverse and Exploding Digital Universe" - http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf

[2] Number of searches increasing - http://blog.searchenginewatch.com/blog/080130-162515

[3] Badue C., Ribeiro-Neto B., Baeza-Yates R. and Ziviani N. Distributed query processing using partitioned inverted files. String Processing and Information Retrieval, 2001. SPIRE 2001. Proceedings. Eighth International Symposium on, (13-15 Nov. 2001), 10-20.

[4] Text REtrieval Conference (TREC) - http://trec.nist.gov/

[5] Zipf, G. Human Behaviour and the Principle of Least Effort. Addison-Wesley, 1949.

[6] Tomasic A. and Garcia-Molina H. Query processing and inverted indices in shared nothing text document information retrieval systems. The VLDB Journal, 2, 3 (1993), 243-276.

[7] Silverstein C., Marais H., Henzinger M. and Moricz M. Analysis of a very large web search engine query log. SIGIR Forum, 33, 1 (1999), 6-12.

[8] Long X. and Suel T. Three-level caching for efficient query processing in large Web search engines. (2005), 257-266.

[9] Delta compression - http://en.wikipedia.org/wiki/Delta_compression

[10] Variable length integer compression - http://en.wikipedia.org/wiki/Variable_length_unsigned_integer

[11] Run-length encoding - http://en.wikipedia.org/wiki/Run-length_encoding

[12] Cacheda, F., Plachouras, V., and Ounis, I. 2004. Performance analysis of distributed architectures to index one terabyte of text. In Proceedings of the European Conference on IR Research, Sunderland, UK, S. McDonald and J. Tait, Eds. 395--408. Lecture Notes in Computer Science, Springer, vol. 2997.

[13] Baeza-Yates R. and Navarro G. Modeling Text Databases. In Anonymous Springer US, 2005, 1-25.

[14] JavaSim - http://javasim.ncl.ac.uk/

[15] Witten I. H., Moffat A. and Bell T. C. Managing gigabytes : compressing and indexing documents and images. Morgan Kaufmann Publishers, San Francisco, Calif., 1999.