This document is a work in progress and will be updated as successive stages of testing are completed. The current report covers Stages 1 and 2 of 5.

HathiTrust Solr Benchmarking

Introduction

The ability to discover content in the HathiTrust repository benefits the archive in a variety of ways. The greater the ability of users to find and use content in the repository, the greater their appreciation of what might otherwise be seen as a preservation effort of hypothetical value. In addition, the process of revealing content in the repository also adds a method for ensuring the integrity of the files; use of those files can reveal problems that might go undetected in a dark archive. While we can facilitate basic discovery through bibliographic searches, deeper discovery through full-text searches across the entire repository provides even greater benefits.

Research into searching a body of content this large is still in its infancy (the repository is approaching one billion pages), and few clear strategies for accomplishing such tasks exist. The major large-scale open source search engine, Lucene, does not provide benchmarking information for data sets this large, and Solr, the most widely deployed implementation of Lucene, has only recently begun gathering benchmarking data. We embark on trying to solve this problem with only general guidance on strategies.

Research programmers in the University of Michigan's Digital Library Production Service have undertaken a process to generate benchmarking data to help shape our strategies. After a preliminary investigation of options, they chose to use Solr and they engaged the Solr development community in helping to define paths. One feature of Solr is its ability to scale searches across very large bodies of content through its use of distributed searching and "shards." When an index becomes too large to fit on a single system, or when a single query takes too long to execute, an index can be split into multiple shards, and Solr can query and merge results across those shards. Although the size of our data clearly points to the need for shards, there are many other variables in designing a successful approach, one that scales to large amounts of data and provides meaningful results.

This report summarizes the strategy we are taking and presents the results of that strategy thus far.

Overview of Strategy

We have attempted to define the variables that have the greatest impact on large-scale searching. We have also tried to stage our benchmarking process so that we start with the simplest approach and introduce each new variable only after collecting benchmarking data on the previous instantiation of the index and environment. Our stages are as follows:

Stage 1 – Growing the index: As the index gets larger, we expect to learn about the size of the index relative to the body of content, time to index with a growing body of content, and degradation of search performance as the amount of content increases. In order to gain a clear sense of the way that these phenomena take place, we are conducting tests on indexes in 100,000 volume steps, from 100,000 to 1,000,000 (with additional increments at 10,000 and 50,000).

Stage 2 – Impact of memory: increasing physical memory and changing JVM configurations will also influence performance. We will increase physical memory from 4GB to 8GB and test several JVM configuration changes in combination with a refined test suite of queries on each of these index sizes.

Stage 3 – Using shards: We will introduce shards in the approach, employing multiple shards with the 8GB memory and optimal JVM configuration. We will test the suite of searches with one shard on each of two physical servers. We will then test the suite of searches with one shard on each of two virtual servers on each physical server (i.e., four shards). Benchmarking data will be gathered for all of the index steps.

Stage 4 – Load testing: We will introduce load testing for the single shard and multi-shard approaches, attempting to see what impact a large number of users will have on performance.

Stage 5 – Faceting results: Full-text searching of a large number of documents will undoubtedly lead to the retrieval of a large numbers of results, and thus usability problems. One obvious strategy for improving navigation of large numbers of results is the use of faceted displays from associated bibliographic data. We will add relatively full bibliographic records to each of the documents and repeat the testing process with a faceted display of results from the bibliographic data.

Stage 1: Growing the Index

Stage 1 investigates the time to index with a growing body of content, and the degradation of search performance as the amount of content increases. In order to gain a clear sense of the way that these phenomena take place, we conducted tests on indexes in 100,000 volume steps, from 100,000 to 1,000,000 (with additional increments at 10,000 and 50,000).

1. Ratio of size of documents to size of index

We found a linear relationship between the sizes of the text base and the index. The collection of documents was typically 2.7 times the size of the resulting index at each increment. The largest body of text, 666.845 GB (one million volumes) resulted in an index of 235.17 GB, or a text to index ratio of roughly 2.8:1 (the index was about 35% the size of the text). A sample of results (up to 606,200 volumes) is shown below.

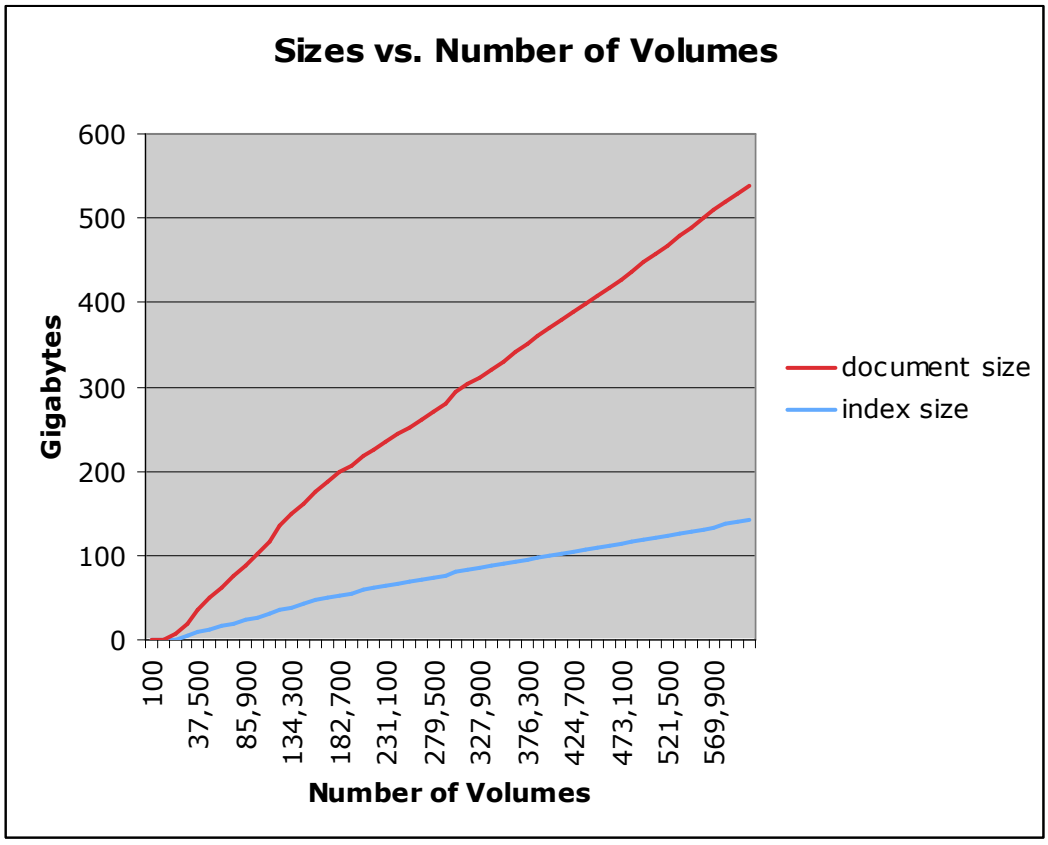

Figure 1. Sizes of the text base and indexes for different numbers of volumes

2. Time to index

The amount of real and machine time needed for indexing varies significantly, depending on a number of factors. With our current environment, the one million-document index took a total of 117 hours in real time to produce. An excerpt of results (up to 402,000 volumes) is given in Figure 2.

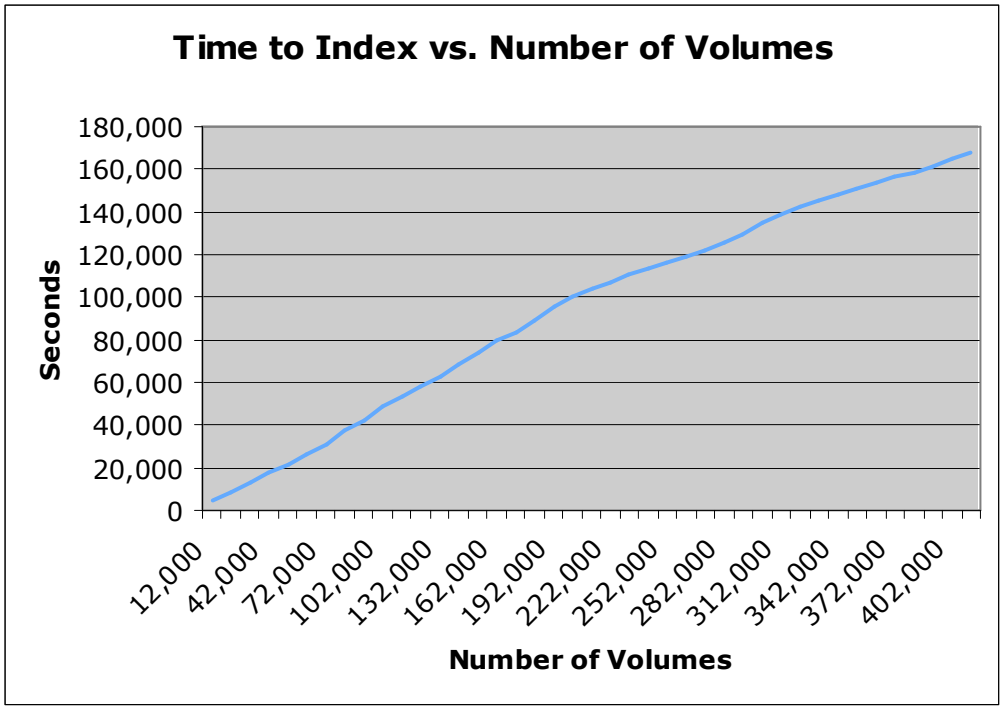

Figure 2. Time to Index vs. Number of Volumes

3. Performance of queries

Results returned using a larger and more representative test suite are reported on in Stage 2 below, but in this stage we were able to make some preliminary observations about performance of a single index in a single Solr instance. Using a small (100 query) sample from the University of Michigan online catalog search logs, we found that across all of the bodies of text indexed, 90% of the queries took fewer than two seconds. For the one million-document index, the most challenging queries exceeded current timeouts or caused memory faults in the current memory/JVM configuration.

Stage 2: Impact of Memory on Search Performance

The objectives of Stage 2 are to:

1. Refine the query suite used in Stage 1
2. Test query response times with increased physical memory (from 4GB to 8GB)
3. Investigate the impact of caching and memory allocation on performance and scalability

1. Refine the query suite

Our goal is to measure the effect of the number of volumes indexed and the size of the resulting index on query response times. Response times, however, can vary greatly depending on the nature of the query. Some of the factors that affect response times are:

- How common the word(s) are in the collection
- How many words are in the query
- Whether the query is a Boolean query or a Phrase query

The queries that take the shortest amount of time are single word queries for words occurring infrequently. The queries that take the longest time are phrase queries which contain several frequently occurring words. As an example, for our 1,000,000 volume index, the phrase query *"the New Economics"* took over 2 minutes to run while the one word query *"golf"* took about a 10^{th} of a second. The slowest query *"the lives and literature of the beat generation"* took nearly 10 minutes (a full distribution of query response times for 1,000 queries run on our 1,000,000 volume index is shown in Figure 3. See the Note on Processing Boolean and Phrase Queries in the Appendix for a more detailed discussion of the relationship between the type of query and query response time). In order to receive results that were close as possible to “real” results that we would see in a production environment, it was important to test with a set of realistic and representative queries.

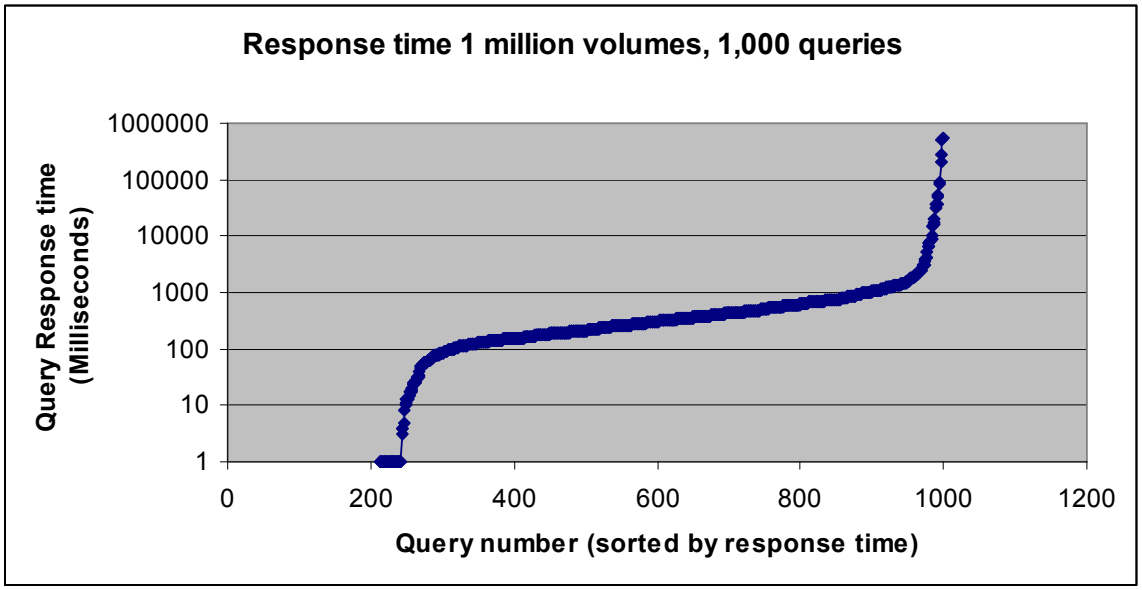

Figure 3. Distribution of query response times for 1,000,000 volume index. Note that response time is shown on a logarithmic scale; the slowest queries took more than 100,000 times longer than the fastest queries.

The tests reported in Stage 1 used 100 queries from Mirlyn to test response times for Solr search. In Stage 2, a total of 5,000 sample queries were taken from Mirlyn query logs to run against the full-text indexes we created. In order to make these queries more representative of queries that would be performed on full-text, the following operations were performed:

1. Removal of special operators (i.e., wildcards, etc.)
2. Removal of duplicate queries that are logged when users view the next page of their own search results[1]
3. Removal of frequently occurring Mirlyn-specific queries, including queries for "proquest" and other databases
4. Removal frequently occurring author and title queries (and other similar queries that are more likely to performed in bibliographic searches, as opposed to full-text searches)[2]

[1] Queries are recorded in Mirlyn logs when users click to a next page of search results as well as when queries are entered for the first time. Since we are interested in queries rather than next page views, we removed duplicate queries that appeared sequentially in the logs, (which were likely a result of a single user viewing the next page of search results). In some cases, these “next page” queries did not occur sequentially in the logs (other users were submitting queries at the same time), but since we were unable to determine if they were identical queries by a new user or “next page” queries from the same user, we did not remove them. This means that there were probably more repeated queries in the samples than there would be if we could accurately remove next page requests.

2. Test Query Response times using 4GB and 8GB of RAM

Results of testing with 4GB of RAM

1,000 queries were run on indexes of 100,000 to 1,000,000 volumes at 100,000 volume increments. In the first set of tests, using a total of 4GB of RAM, 2GB of RAM was allocated to the Java Virtual Machine (JVM) running Solr, and the remaining 2GB was allocated to the operating system. In the second set, reported on below, 8GB of RAM was used, with 2GB allocated to the JVM and 6GB to the operating system. A description of Solr architecture and the implications of memory allocation are given in the Appendix.

Queries are sometimes repeated by users, and a word or words from one query often show up later in another query. Search engines exploit this by caching query results in memory and caching intermediate processing results for query terms in memory. Since memory access is several orders of magnitude faster than retrieving data from disk, this strategy can result in a very significant reduction in response times. In our tests, we warmed up the operating system and Solr caches by running 4,000 Mirlyn queries (cleaned in the same manner described above) before running the 1,000 test queries.[3] Further experiments with cache warming are reported in Section 3 below.

For the current set of tests, the Solr query results cache was set to cache 512 results, and the Solr document cache was set to cache 512 documents. These are Solr's default settings. (Tests with different settings are reported on in Section 3. See the Appendix for a detailed description of Solr architecture and caching.)

Because response times can vary significantly between queries, and it is possible for a small number of slow queries to significantly affect the average

---

[2] In spite of these measures taken to clean the data to produce representative queries, we recognize that in general, the queries reflect those that would be made against MARC metadata such as author, title and LCSH subject, and not a full-text index of content. Future testing will use queries mined from the Hathi trust search demo (http://babel.hathitrust.org/cgi/ls/).

[3] The 4,000 query number was selected as a reasonable compromise between a number small enough to be able to run the test suite in a reasonable amount of time and a number large enough to get reasonable results. Whether 4,000 queries are adequate to warm the cache will require further investigation. Many researchers investigating caching issues use between 10,000 and 100,000 queries to warm the caches. (See for example Baeza-Yates et al. (2008:pp.14,19,22); Moffat, A., Webber, W., Zobel, J., & Baeza-Yates, R. (2007:p217); and Zhang, J., Long, X., & Suel, T. (2008:p 391)). Part of the reason for using so many queries is that in a production situation if cache size is limited, over time the most frequent queries/query terms will end up staying in the cache.

query response time, we used the median and 90th percentile response times to evaluate our results. The median response time is the response time such that 50% of queries are faster and 50% of queries are slower. The 90th percentile response time is the response time such that 90% of the queries are faster and 10% are slower. As shown in the figures and tables below, 50% of the queries returned results in about 2 tenths of a second or less, and 90% of the queries returned results in close to 1 second or less.

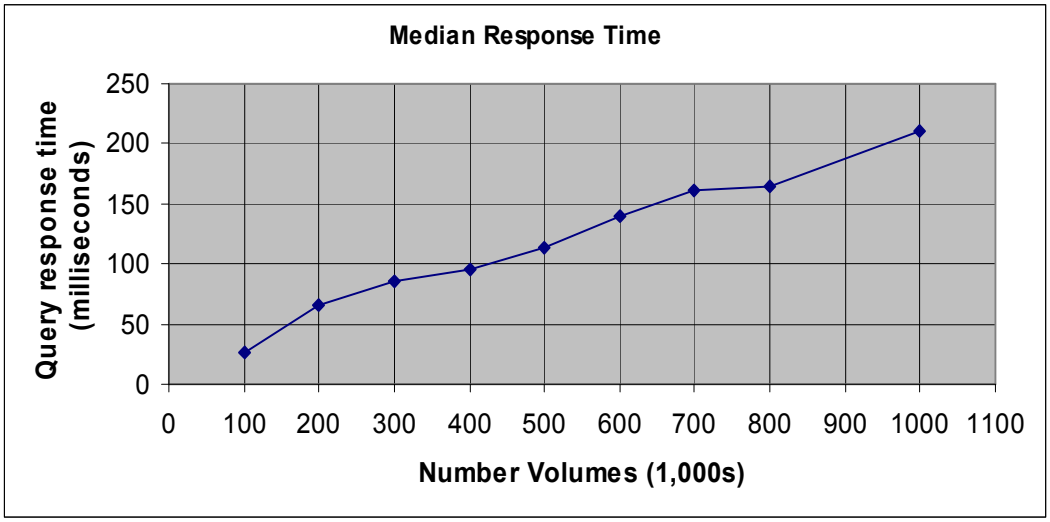

Figure 4. Median Response time (50th percentile) per 100,000 volumes.

Median Response Time									
Thousands of volumes	100	200	300	400	500	600	700	800	1000
Time (milliseconds)	27	65	85	96	113	140	162	164	211

Table I. Median Response time (50th percentile) per 100,000 documents

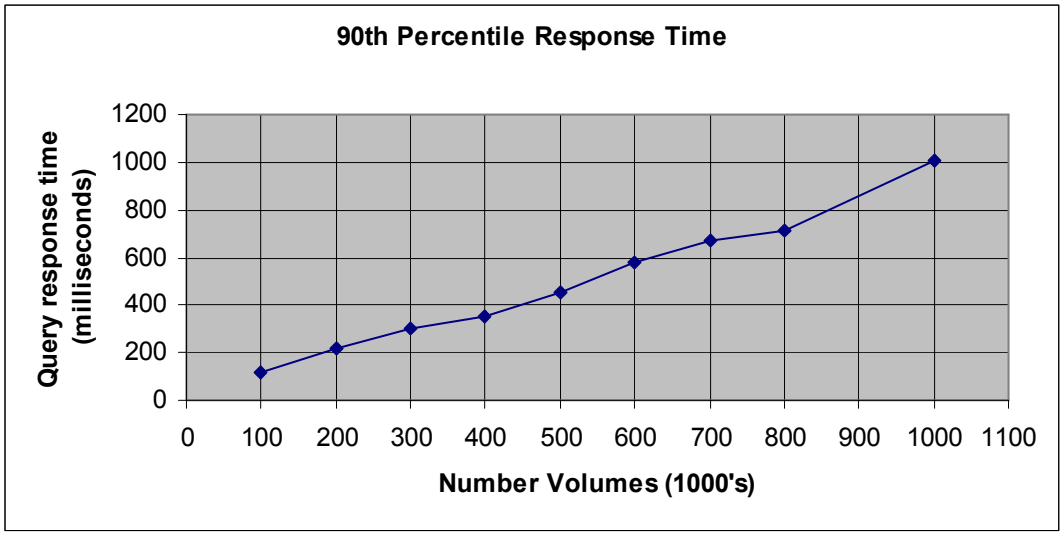

Figure 5. 90th percentile response time per 100,000 documents.

90th Percentile Response Time									
Thousands of volumes	100	200	300	400	500	600	700	800	1000
Time (milliseconds)	117	222	301	354	453	577	675	715	1004

Table II. 90th percentile response time per 100,000 documents

Response times for queries near and above the 90th percentile were as follows:

Query Response Time	Percentage
over 1 second	10.90
over 5 seconds	2.30
over 10 seconds	1.60
over 20 seconds	1.10
over 30 seconds	1.10
over 1 minute	0.60
over 2 minutes	0.40
over 3 minutes	0.40
over 4 minutes	0.30
over 5 minutes	0.20

Table III. Percent of all queries with 1 second or longer response times (for 1,000 queries against the 1,000,000 volume index).

As shown, over 97% of queries results were returned within 5 seconds. This is an encouraging outcome, but the query times that lie in the upper range, above the

90th percentile, are of particular concern to us because of the effect they may have on user experience. The range of “acceptable” response times varies from one user to the next, but it can be safely assumed that response times over a minute will be unacceptable to most users, and that response times over 30 seconds or even less will be unacceptable to many users.

Table IV shows the queries that took over 30 seconds for the 1 million document index. As shown, most of the slow queries are phrase queries containing common words. There are two Boolean queries, one is a call number query and the other contains somewhat common words. A broader analysis of the entire set of 5,000 queries indicates that the slowest queries are phrase searches containing common words, and call number searches. Finding ways to deal with these “slow” queries in the future will be important to the usability of our full-text search.

Rank	Time (ms)	Hits	Query
1	532,051	43	"the lives and literature of the beat generation"
2	494,177	312	"night and the city"
3	269,006	0	"A First Course in Statistical Programming with R"
4	201,113	2,466	"The New Economics"
5	89,502	116	"army of shadows"
6	81,025	11	"hip a history"
7	52,838	33	"principles of biochem"
8	50,961	7,403	"immigration and naturalization service"
9	37,738	95,914	histoire AND de AND l'art
10	35,675	706	"7th international congress 1985"
11	32,193	64	hg AND 6024 AND a3 AND h851

Table IV. Queries taking longer than 30 seconds on 1,000,000 volume index.

Results of testing with 8GB of RAM

For our next set of tests, the memory on our test machine was increased from 4GB to 8GB. The amount of memory allocated to the Solr JVM was kept at 2GB. This resulted in a tripling of the memory available for operating system disk caching (from 2GB to 6GB). The response times observed are shown below.

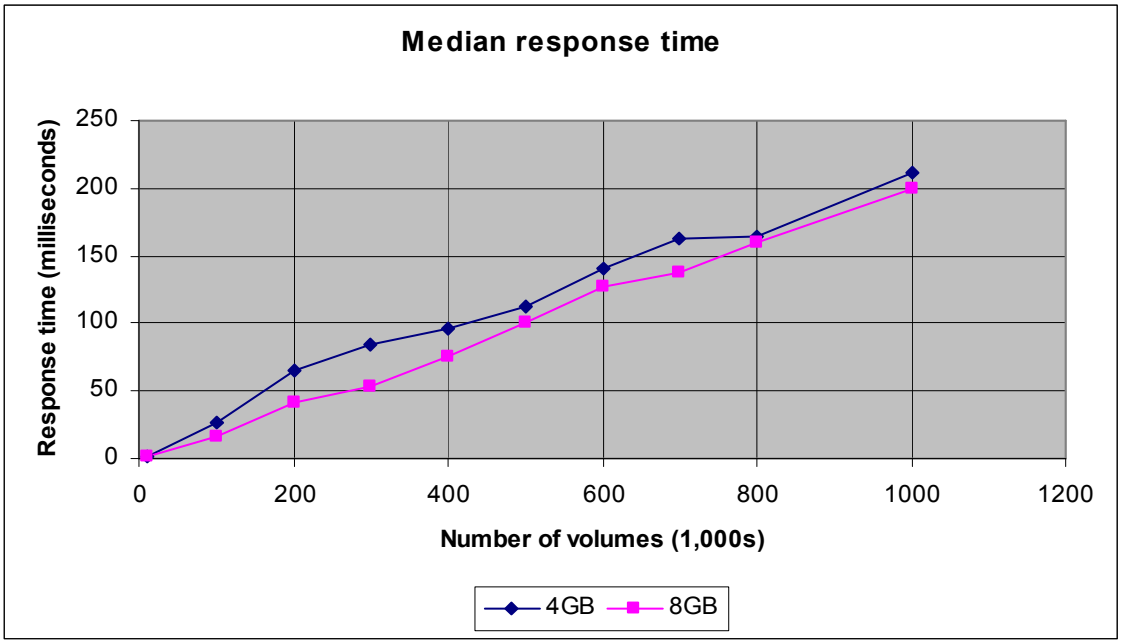

Figure 6. Median response times for 4GB and 8GB of RAM

Median Response Time											
Thousands of volumes		**10**	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**1000**
Total memory	4GB (2GB) JVM	2	27	65	85	96	113	140	162	164	211
	8GB (6GB) JVM	2	17	42	53	75	100	127	138	160	200
Percent decrease		0.00	37.04	35.38	37.65	21.88	11.50	9.29	14.81	2.44	5.21

Table V. Median response times for 4GB and 8GB of RAM

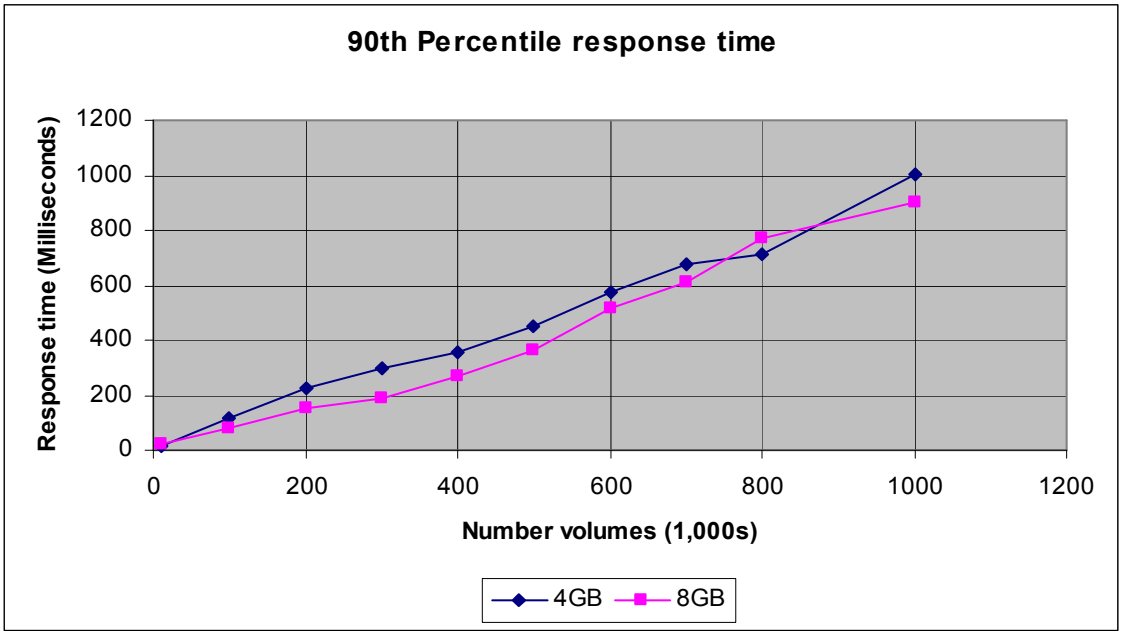

Figure 7. 90th percentile response times for 4GB and 8GB of RAM

90th Percentile Response Time											
Thousands of volumes		**10**	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**1000**
Total memory	4GB (2GB JVM)	18	117	222	301	354	453	577	675	715	1004
	8GB (2GB JVM)	19	83	155	191	269	366	518	614	768	901
Percent decrease		-5.56	29.06	30.18	36.54	24.01	19.21	10.23	9.04	-7.41	10.26

Table VI. 90th percentile response times for 4GB and 8GB memory

Note: some anomalies appear in the results for the 10,000 and 800,000 volume sets where the 90th percentile response time was greater with 8GB memory than 4GB. This was likely due to interference from non-search related processes that were running on the operating system when the tests were done. Additional testing will determine if this was the case.

When compared with results of our first set of tests, differences in response times between 4GB of RAM and 8GB can be grouped roughly in ranges of index sizes as follows: a 35-37% decrease in median response time for the 100,000-300,000 volume collections and 10-15% decrease in median response time for

the 500,000-700,000 volume collections; a 30-35% decrease in 90th percentile response time for the 100,000-300,000 document collections and about a 10% decrease in 90th percentile response time for the 600,000-800,000 document collections. For the 10,000 document collection, there is not a significant change. This is probably due to the fact that almost the entire index was able to fit in the cache (the cache sizes were 2GB and 6GB for the 4GB and 8GB RAM tests, respectively). Table VII below shows the size of the indexes for different numbers of volumes, and the percent of those indexes that are able to fit in a 2GB and 6GB cache.

Thousands of volumes	Index size (GB)	% of index in 2GB disk cache	% of index in 6GB disk cache
10	2.5	80.00	100.00
50	14	14.3	42.9
100	29	6.9	20.7
200	58	3.5	10.3
300	80	2.50	7.5
400	100	2.00	6.0
500	120	1.67	5.0
600	141	1.42	4.3
700	165	1.21	3.6
800	186	1.08	3.2
900	na	na	na
1000	226	0.88	0.7

Table VII. Index size and percent of index that can be held in the operating system disk cache at any one time.

Conclusions

We observed in each set of experiments that the relationship between response time and index size as index size increases is linear. This means that for the 1,000,000 volumes we have tested so far, response time is scaling with the number of indexed volumes. Future testing will determine whether this remains the case for greater numbers of volumes, and whether the majority of the response times are still within reason. The response times for larger indexes may have implications for load handling in scenarios where multiple users are querying the system at the same time. Developing strategies to deal with these longer query times will be an important area of future work.

As far as memory is concerned, in general as noted above, the increase in memory from 4GB to 8GB (2GB to 6GB for operating system memory) made a greater difference at smaller index sizes, where the percent decrease in response time was higher. This is what we would expect because a larger percentage of the index is able to fit into memory when the index is small, and suggests that in order to improve response time at larger index sizes, we will need to add more memory. We hope to discover the reason for the lack of variation within the groupings, particularly for the median response times in the 100,000 to 300,000 volume range, in future testing.

3. Caching, Memory, I/O and Scalability

Since retrieving data from directly from memory can be several orders of magnitude faster than using I/O to retrieve the data from disk, caching data in memory is used extensively in production search engines. It is commonly applied to the caching of query results and/or term postings lists (see the Appendix for descriptions of each). The caching of query results means that for an exact repeat of a query Solr can skip both the processing of postings lists and scoring, so it is very fast. Even a small query results cache can be beneficial since query repetition is most likely to occur in the same search session. Caching of term postings lists still requires Solr to process the postings list and calculate ranking scores, so it does not save as much time as caching query results. However, cache hits are more likely since a word in one query can be repeated in a different query.

Because the term postings list is cached in the operating system disk cache and query results are cached in Solr, which gets its memory from the JVM, we believed results of performance testing for each cache separately could have implications for how we allocate memory between the operating system and the JVM. We therefore performed two series of tests to evaluate the effects of caching on performance. The first series measured query response times with and without operating system cache warming when the size of the query results cache was kept constant. The second series measured response times for two different sizes of the Solr query results cache. In both series of tests 2GB of RAM was allocated to the JVM. This left 2GB of RAM for the operating system disk cache for the tests with 4GB of RAM, and 6GB for the operating system disk cache for 8GB of RAM.

Comparison of response times with and without caching

The two charts below compare the median and the 90th percentile query response times for 1,000 queries on the 10,000, 100,000, 400,000 and 700,000 volume indexes with and without cache warming.[4]

Median Response Time					
Thousands of volumes		**10**	**100**	**400**	**700**
Response Time (milliseconds)	1000 queries no cache	9	48	108	166
	1000 queries warm cache	2	27	96	162
Percent decrease		77.78	43.75	11.11	2.41

Table VIII. Median query response time
with and without 4000 query cache warming.

90th Percentile Response Time					
Thousands of volumes		**10**	**100**	**400**	**700**
Response Time (milliseconds)	1000 queries no cache	44	185	419	705
	1000 queries warm cache	18	117	354	675
Percent decrease		59.09	36.76	15.51	4.26

Table IX. 90th percentile query response time
with and without 4000 query cache warming.

Conclusions

Caching has a much greater impact on the smaller indexes as is seen in the percentage decrease in response times. This is most likely because at any one time, the smaller indexes can get a larger percentage of the index into memory (see Table VII). If this is the case, we could conclude that query response times for larger index sizes are Input/Output bound rather than CPU bound. In other

[4] For the test without warming, the operating system disk cache was cleared by running 5,000 queries on a different Solr index, so it was filled with what the OS would see as blocks from completely different files. The Solr caches were cleared by restarting Solr. For the test with cache warming, the same procedure was followed to clear the caches and then 4,000 warm up queries were run prior to running the 1,000 test queries.

words, that response time depends more on the time it takes to retrieve indexed information from the hard drive than the time to process information (whether retrieved from the hard drive or one of the caches). To improve overall performance in future tests, we will be guided by the need to reduce I/O demands and/or increase I/O performance.

Effect of increasing the query result cache and document cache

The results of the above tests show primarily the effects of operating system disk caching. Solr has several additional caches, the most important of which for our purposes is the query results cache. In all of the experiments described thus far, we used the out-of-the-box settings for this cache, which configure it to store 512 query results. This is a relatively small number. Additional experiments were performed to determine the effects of increasing the query results cache (and document cache as well) on query response time. Since memory for Solr caches comes out of the memory allocated to the JVM, we also attempted to identify from these results any changes we should make to the allocation of memory between the operating system and the JVM.

In these experiments, done at the 8GB memory level, the size of the query results cache was increased from 512 queries to 100,000 queries, and the document cache from 512 to 50,000. The top ten results for each query were saved in the query results cache and 4,000 queries were run to warm the cache. The results are shown below.

Thousands of volumes		**100**	**200**	**300**	**600**	**700**	**800**
Median response time	Small cache (512 query results/512 documents)	17	42	53	127	138	160
	Large cache (100,000 query results / 50,000 documents)	15	40	45	116	130	141
Percent decrease		11.76	4.76	15.09	8.66	5.80	11.88

Table X. Median response times for 1,000 queries on different size indexes with small and large caches.

Thousands of volumes		100	200	300	600	700	800
90th percentile response time	Small cache (512 query results / 512 documents)	83	155	191	518	614	768
	Large cache (100,000 query results / 100,000 documents)	81	162	183	495	560	615
Percent decrease		2.41	-4.52	4.19	4.44	8.79	19.92

Table XI. 90th percentile response times for 1,000 queries on different size indexes with small and large caches.

The effect of increasing the query results and documents cache sizes varies with different size indexes as shown, but overall we observed that using a larger cache resulted in a decrease in response time (progressively so for the 90th percentile test series). This is what we would expect, but to get a better indication of the effects of the larger cache size, we ran additional tests on the 100,000 and 600,000 indexes using 24,000 and 54,000 warm-up queries on each index, respectively. The results, compared with those above, are shown in Tables XII and XIII.

100,000 Volumes					
Cache size	Cache warm-up	Median Response Time (ms)	Percent decrease	90th Percentile Response Time (ms)	Percent decrease
512 query results/ 512 documents	4,000	17	11.76	83	2.41
100,000 query results / 50,000 documents	4,000	15	20.00 (29.41 Total)	81	19.75 (21.69 Total)
100,000 query results / 50,000 documents	24,000	12		65	

Table XII. Median and 90th percentile response times for 1,000 queries on a 100,000 volume set using 4,000 and 24,000 warm up queries.

600,000 Volumes					
Cache size	**Cache warm-up**	**Median Response Time (ms)**	**Percent decrease**	**90th Percentile Response Time (ms)**	**Percent decrease**
512 query results/ 512 documents	4,000	127	8.66	518	4.44
100,000 query results / 50,000 documents	4,000	116	19.83 (26.77 Total)	495	6.06 (10.2 Total)
100,000 query results / 50,000 documents	54,000	93		465	

Table XIII. Median and 90th percentile response times for 1,000 queries on a 600,000 volume set using 4,000 and 54,000 warm up queries.

For both the 100,000 and 600,000 volume indexes we observed that increasing the number of cache-warming queries (with the larger cache sizes) significantly decreased the response time. Since the larger number of cache-warming queries is more representative of the system in realistic production conditions, our previous experiments with only 4,000 cache-warming queries (shown in tables X and XI above) probably underestimate the effect of increasing the Solr caches on response time in a production situation.

Conclusions

Although the decrease in response time due to increasing the Solr caches varied, we see that increasing these caches did decrease response time. Limited observations of JVM memory allocation during the tests and the absence of any "out of memory" errors suggest that the 2 GB memory allocation was adequate to allow the 100,000 query result cache and 50,000 document cache settings. Further evaluation will be required to determine the optimum cache settings and any possible trade-offs between memory allocated to Solr, memory allocated to the Solr caches, and memory available for the operating system disk cache.

Our results from sections 2 and 3 of this stage of testing lead us to believe that allocation of 2GB of memory to the JVM for Solr caching is adequate for a significant amount of query result caching and document caching, but that memory allocated to the operating system (which is used for disk caching), is too small in comparison to index size at the larger index sizes. From this we can

conclude that the slowest part of our search system for larger indexes is I/O (the process of retrieving data from disk).

Summary of Stage 2 Results and Conclusions

The primary areas of testing covered in this stage of testing, with their results, concerns, and conclusions are as follows:

Tests of query response times at 4GB and 8GB of memory, using the new set of test queries

> Results: 1) there is a linear relationship between index size and query response time as the size of index volumes increases (confirming what we observed in Stage 1); 2) increasing memory reduced response time; and 3) the reduction in response time was greater for the smaller indexes.
>
> Concerns: there is a small percentage of queries in this distribution that take an “unacceptable” amount of time. This is a concern for individual users, and may have an effect on the response times of other users who are performing searches at the same time, as well.
>
> Conclusions: 1) experimenting with larger indexes will determine if the relationship of index size to query response time continues to hold; 2) experimenting with more memory will help determine the relative contribution of memory to performance; 3) strategies will need to be developed to deal with slower queries so they do not pose a usability problem for single or multiple users.

Tests with Caching

> *Testing of the operating system disk cache (by varying the number of warm-up queries)*
>
> > Results: tests showed that response times are significantly lower when large amounts of the indexes are able to fit into memory.
> >
> > Concerns: this leads us to believe that operating system memory could be the most significant bottleneck in search performance.

Conclusions: To the extent that indexes are too large to fit into memory, improving I/O performance will be required to improve overall performance

Testing of the Solr caches

Results: testing revealed that increasing the Solr query results and document caches can significantly reduce query response time.

Concerns: Although with the cache settings tested, the amount of memory used by the caches did not appear to adversely affect the use of memory by other Solr processes, this could be a concern with either larger cache settings or with a different operating regime that requires more memory for Solr processing (processing requests from multiple users simultaneously, requesting more search results to be returned, sorting results, etc.).

Conclusions: We were not able to determine from these experiments the maximum benefit we could receive from increasing the size of the Solr caches, or how much memory those caches use. There may, therefore, be a benefit to continuing to increase the size of the Solr caches, but this will have to be weighed against possible adverse effects on the performance of other Solr processes, and the performance tradeoffs that may come from not allocating additional memory to the operating system disk cache.

Appendix – Solr Search Architecture

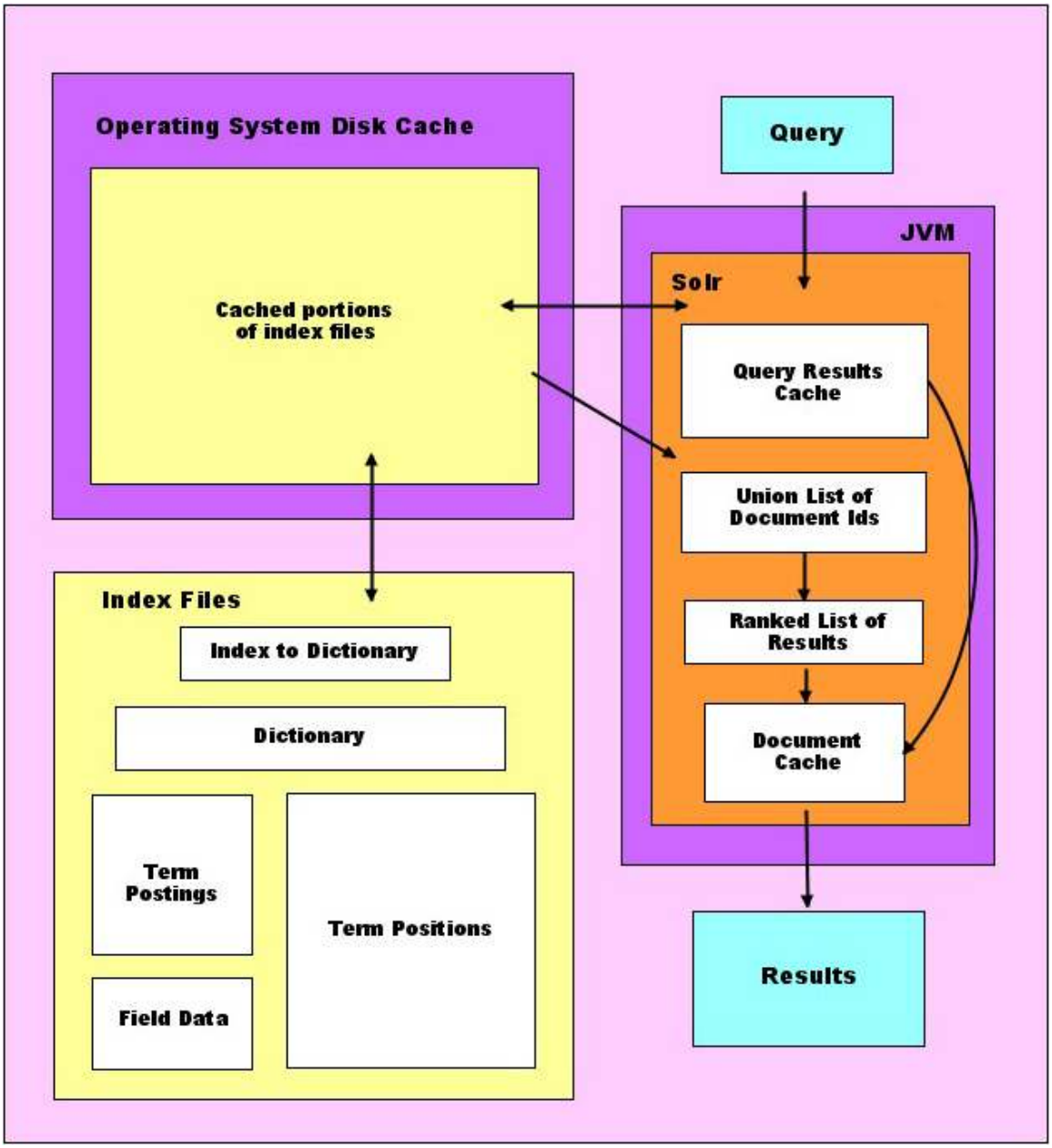

Figure 1. Solr Search Process Architecture

The following discussion presents a simplified overview of how Solr processes search requests with an emphasis on how each step is affected by disk I/O and CPU processing. We use the term "document" here instead of "volume" to be consistent with the Solr documentation and discussions in the information retrieval literature.

Solr Search Anatomy

As shown in the diagram above, there are three main components to the Solr Search Process 1) the Solr search index, which is composed of multiple files stored on the hard drive, 2) the operating system disk cache[5], which contains a cache of recent disk reads in memory, and 3) the Java Virtual Machine (JVM), where Solr itself is run. Each of these will be discussed below, followed by a description of the Solr Search Process.

> Note: Memory allocation refers to the amount of memory that is allocated to the operating system and to the Java Virtual Machine. The Linux operating system used for our Solr implementation uses any available memory that is not in use by other processes for disk caching. This allows some portion of the Solr index files to be cached in memory as Solr processes queries and requests disk reads. In our tests with 4GB and 8GB total memory, we allocated 2GB of memory to the JVM running Solr and there were no other significant processes running. This means that for 4GB total memory, 2GB were available for caching of Solr disk reads; for 8GB total memory, 6GB were available.

1) The Solr Index is composed of a number of files. For purposes of this discussion the following files are relevant:

1. The Dictionary – a list of all terms in the Solr index. The dictionary entry for each of the terms has pointers to locations on the hard drive where lists of ids for documents containing that term (Term Postings) are stored, and to locations where the list of the positions of the term in each document (Term Positions) are stored. This is to facilitate fast access to the postings or positions lists for a particular term in a query.
2. Dictionary Index – an index of the Dictionary that provides rapid access to entries in the Dictionary. The Dictionary index is kept to a relatively small size so it will fit in available memory. For the 1 million document index the Dictionary Index is about 75 MB, whereas the Dictionary is about 5 GB.
3. Term Postings – for each term in the Solr index this file lists the document ids of all the documents containing that term. It also lists the frequency of the term in the each document. This is pointed to by the Dictionary.[6]

[5] The operating system disk cache is not a part of Solr, but Solr is designed to take advantage of operating system disk cache, so we consider it a component of the Solr search processing architecture.

[6] Caching of the term postings lists has a greater likelihood of cache hits than caching of query results since terms can be combined in numerous queries. In contrast to query results caching,

4. Term Positions – for each term in the Solr index, this file contains a list of the positions of that term within each document. Term positions are used when phrase queries are submitted to determine if the terms in the phrase are adjacent to one another in the volume. Because we are indexing entire books (as opposed to web pages or articles), this file can grow extremely large. In each of the indexes we produced, the Term Positions file made up about 85% of the total index.
5. Field Data – a file containing stored metadata fields such as author and title for each document. This metadata is used for displaying lists of search results.[7]

2) The operating system disk cache holds as much of the index files in memory as possible. If the cache is not large enough to hold all of the index files or is not completely warmed up, only a portion of each of these files is stored at any one time.

3) The JVM is where Solr processing and Solr caching occurs. For the purposes of discussing caching in Solr, we can consider Solr as composed of the following parts which exist in memory:

1. Query Results Cache – a cache containing an ordered set of document ids of the top "n" search results for particular search queries, "N" is by default the number of documents requested in the query. For all of our tests we used the default setting, which returned 10 results for each query, but this number can be configured.
2. Document Cache – a cache containing metadata about the documents, such as title and author. The number of documents whose metadata is cached can also be configured.
3. Union list of document ids – a list (actually a data structure) generated by Solr during search processing. It contains the document ids of all documents containing all words in the query.
4. Ranked Results List – a list (also a data structure) of relevance ordered document ids for the top "n" search results of a particular search. This is also generated during search processing.

---

when term postings are cached relevance ranking and scoring calculations must still be performed on the new term combinations. Query result caching is done by Solr whereas caching term postings lists is done by the operating system disk cache

[7] There are actually two files relating to field data. There is a field index file that is used to provide fast access to the field data file, which contains the actual data.

Solr Search Processing

There are two basic types of searches that Solr can process, Boolean queries (i.e. cat AND dog) and Phrase queries (i.e. "to be or not to be"). An overview of how these are processed with an emphasis on how the processing involves disk reads (I/O) and CPU resources follows.[8] A label at the end of each step indicates whether the step is dominated by I/O or processing.

For Boolean "AND" queries:

1. *For each term in the query, fetch the list of document ids for the documents containing that term*. This is done by 1) Looking up the term in the Dictionary Index 2) Reading some of the dictionary into memory and looking up the term in the dictionary to retrieve the location on disk of the terms listings in the Term Postings file 3) Reading the relevant portion of the Term Postings file into memory and retrieving the lists of document ids and the term frequency of the term in each document. **[I/O]**
2. *Create the union of the lists of document ids*. When the ids for documents matching the relevant query terms have been retrieved, Solr walks through each list and creates a union list which contains the document ids of only those documents that contain all the terms. **[Processing]**
3. *Relevance rank all the documents in the union list.* Solr processes the union list to rank the documents according to relevance. **[Processing]**
4. *Produce the Ranked Results List for the top ranking document ids*. As mentioned, the number of results that are returned can be configured. Since we requested10 results for all of the testing we performed, Solr returns a list of the top 10 document ids in relevance order **[Processing]**
5. *Return metadata for the top ranking documents for use in the display of results.* For each document id, Solr retrieves metadata from the Field Data File disk. **[I/O]**

For Phrase queries:

1. *For each term in the query, fetch the list of document ids for the documents containing that term.* As above. **[I/O]**
2. *Create the union of the lists of document ids*. As above **[Processing]**

[8] The process described is accurate at a general level but the implementation details involve combining some of the various steps, so this should not be taken as a exact description of how Solr implements query processing.

3. *For each document id, for each term in the query, fetch the list of term positions in the document*..
   For a common word such as "the", the term positions list can be extremely long. Zobel and Moffat (2006) found that for one large collection, the compressed position lists for a few common words accounted for about 10% of the total index size. *(p 23)* . **[I/O]**
4. *Based on the list of term positions, determine whether the phrase occurs in the document.* **[Processing]**
5. *If the phrase does occur, add the document to the list of documents to be scored*. **[Processing]**
6. *Relevance rank all the documents in the final list of documents containing the phrase.* As above. **[Processing]**
7. *Produce the Ranked Results List for the top ranking document ids.* As above **[Processing]**
8. *Return metadata for the top ranking documents for use in the display of results*. As above. **[I/O]**

When Solr processes a query it first checks the query results cache (refer to Figure 1). If the results for that query are in the query results cache, all the steps except the last step, returning document metadata (steps 1-4 for Boolean and steps 1-7 for Phrase queries), can be skipped. This avoids any disk reading or processing in those steps, making response times very fast.

Excluding again for a moment the final step of search, when a query does not match the query results cache, Solr walks through the steps for a Boolean or Phrase query (depending on the type of query), making requests for information from disk as needed (steps labeled "I/O" above). Every time it does this, the operating system first checks to see if the information is in the disk cache (stored in memory). If the information is in the disk cache, the operating system sends the information to Solr without having to read the disk. If the information is not in the disk cache, the information must be read from the disk itself, which requires additional time. Disk caching, therefore, speeds up any of the steps listed above that require disk reads (labeled I/O).

In the final step of search, the ids from the Ranked Results List (gathered either from disk seeks or from the query results cache) are looked up in the Solr document cache. If any of the document ids are not in the document cache, Solr must retrieve the metadata for them from disk. When Solr requests a disk read from the operating system, as discussed for the other I/O operations above, the operating system first checks to see if the information is

in the disk cache and if it is, a disk read is avoided. The process of searching the document cache before going to disk (or disk cache, as the case may be) is the same for both Boolean and Phrase queries.

Note on Processing Boolean and Phrase Queries

Our experiments indicate that queries containing common terms, and especially phrase queries containing common terms take the longest. As the number of documents in an index increases, so does the index size. For common terms (i.e. terms such as "the" or "of" that appear in almost every document), the size of the Term Postings and Term Positions lists for those terms grows approximately linearly with the number of documents. This means that any Solr processes that request disk reads for those lists need to read in more data.

Boolean queries with words that occur in many documents take longer than queries with infrequently occurring words. This is primarily because the Term Postings lists (list of documents containing particular terms) for words that occur in many documents are much longer and therefore more data must be read from disk. Consider for example a query containing the word "the". For a one million document collection, the Postings List for this term might contain close to a million document ids. A second reason that these queries take longer is that processing the longer lists to produce the union list of document ids takes longer.

Phrase queries with common terms take longer than Boolean queries, primarily because phrase query evaluation involves reading from the Term Positions file and entries in the Term Positions file tend to be much longer than the corresponding entry in the Term Postings list. The Term Positions file is generally about 85% of the size of the entire index. A second reason that phrase queries take longer than Boolean queries is that the processing of the position lists to check to see if words in the phrase are adjacent involves significant additional processing.

Boolean or Phrase queries that result in a large number of hits also take longer than other queries because the process of relevance ranking a larger number of hits requires more processing. As the number of documents indexed increases, the number of potential hits also increases.

References

Baeza-Yates, R., Gionis, A., Junqueira, F. P., Murdock, V., Plachouras, V., & Silvestri, F. (2008). Design trade-offs for search engine caching. *ACM Trans. Web*, 2(4), 1-28. doi: 10.1145/1409220.1409223.

Moffat, A., Webber, W., Zobel, J., & Baeza-Yates, R. (2007). A pipelined architecture for distributed text query evaluation. *Information Retrieval*, 10(3), 205-231. doi: 10.1007/s10791-006-9014-4.

Zhang, J., Long, X., & Suel, T. (2008). Performance of compressed inverted list caching in search engines. In *Proceeding of the 17th international conference on World Wide Web* (pp. 387-396). Beijing, China: ACM. doi: 10.1145/1367497.1367550.