Journal of Knowledge Information Technology and Systems

ISSN 1975-7700

http://www.kkits.or.kr

Workload Characterization for Web Search

Myeongjae Jeon[1], **Youngkyu Lee**[*2]

[1]*Microsoft Research*
[2]*Daejeon Institute of Science and Technology*

ABSTRACT

Web search distributes query processing across many servers, and each web search server processes multiple queries concurrently to achieve high throughput. A fundamental requirement of web search is responsiveness, which is typically guided by a response time SLA. As this is applied to each and every search query, understanding how system resources are utilized during query execution is important. In this paper, we present workload characterization for web search, with a focus on temporal and spatial locality of index data access. A key finding includes high temporal locality, suggesting that data caching plays an important role in achieving good performance. Moreover, this paper analyzes how effective the preferching of index data is in web search. The prefetching is a very common optimization in web search in order to amortize delay related to data reads from external storage. From the analysis, we observe that the prefetching is not only frequent, but also aggressive, with large size of data to be issued for prefetching. Surprisingly, this optimization is effective in web search. However, some portion of the prefetched data is not accessed by the query due to early termination, and this paper shows that this accounts for 8.3% of total prefetched data. As a future work, we will study how to reconcile many prefetching requests issued by concurrent queries under heavy loads. Lastly, we believe that this work will fuel future endeavors on improving memory management for web search workloads.

KEYWORDS: Web search, Workload characterization, Memory management, I/O, Locality

ARTICLE INFO: Received 5 August 2016, Revised 12 August 2016, Accepted 12 August 2016.

[*]Corresponding author is with the Department of Social Welfare, Daejeon Institute of Science and Technology, 100 Hyecheon-ro Seo-gu Daejeon, 35408, KOREA.
E-mail address: leeyk@dst.ac.kr

1. Introduction

Today, web search engines commonly achieve large-scale parallelism in two complementary ways. They process multiple search queries concurrently, and they distribute the processing of each query over hundreds or thousands of servers [1−3], where the slowest server determines the request latency. On a single server, it is important to understand the critical path in query execution and optimize the system for it in order to provide high responsiveness of the service. There are two important two challenges in doing so. First, queries exhibit large variability, where the service demand of long queries is orders of magnitude higher than the median [4]. Long queries typically process more data than short queries, so are more expensive to provide low response time. Second, queries interfere within the server, making the latency at light loads is different from that in heavy loads.

While research on web search optimizations has been conducted for decades[5-12], there has been little work on the in-depth analysis of server-side web search workloads. We believe that workload characterization of web search presents more challenges and opportunities on the design of effective server systems.

In this paper, we fill this gap by characterizing real-world workloads for a commercial web search engine. Our focus is on the locality of index data accesses and the popularity of the accesses. This is because to make a service responsive, we must process data in memory and avoid generating

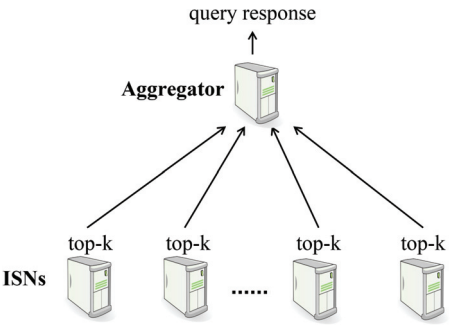

그림 1. 인덱스 처리 시스템 구조

Figure 1. Index serving system architecture.

frequent I/O requests. In this paper, our analysis is based on production index and query requests, and on telemetry data collected from a single search server running he requests on the index experimentally.

2. Background

2.1 Web Search Architecture

System architecture. <Figure 1> illustrates the partition aggregate architecture of an index serving system. It consists of an aggregator (also known as a broker) and search servers called index serving nodes (ISNs). The index contains information about web documents, is document-sharded [1] and distributed among the ISNs. When a user sends a query and the query response is not cached, the aggregator propagates the query to all ISNs hosting the web index. Each ISN searches its fragment of the web index to return the top-k most relevant results to the aggregator. The aggregator receives the results from the ISNs, and merges them to compute the

response to the user query. The aggregator waits for all of its ISNs to respond so it will not miss any search results. ISNs are the workhorse of the index serving system. They constitute over 90% of the total hardware resources and account for the majority of the query processing time.

Query processing. The ISN manages a number of worker threads that can process several queries concurrently on multiple cores. Newly arrived queries first join the waiting queue of the ISN. When a worker thread is idle, it dequeues a query from the waiting queue and starts to process it. The worker thread searches its web index fragment to produce a list of documents matching the keywords in the query. As there are a fixed number of worker threads, some queries may experience a delay in the waiting queue. Thus, a query's response time consists of both its queueing delay and execution time. The execution time can be further broken down into the time spent on CPU and the time blocked on disk I/O upon data access misses in memory.

표 1. 어플리케이션 정보와 입출력 정보로 구분된 로그

Table 1. Collected log entries categorized into application-level information and I/O-level information

Level	Entry name	Information
App.	Query arrival	query id, time
	Query completion	query id, time
	Foreground access	query id, page id, time
	Prefetching	query id, first page id, size, time
IO	IO issue	query id, first page id, time
	IO completion	query id, first page id of the IO, time

2.2 Data Collection

We collected the traces of query processing using our experimental platform that consists of a server machine and a query generator. The server machine stores 16 GB of index pages in SSD and has its own memory buffer that caches recently accessed pages containing indices. The query generator plays queries from a trace of 100K user queries using a Poisson process in an open loop.

To collect memory page access traces, queries are are issued one per second and executed almost sequentially. This issue rate reveals resource demands and prefetching accuracy, assuming that system resources are dedicated for each query.

2.3 Information in Collected Data

The server application searches the memory buffer to find index data required by the processing of queries, and, if missed, issues page requests to the I/O subsystem. The collected trace data precisely contains all of these applicationlevel and I/O-level events. <Table 1> shows the summary of information revealed in the collected log entries. Applicationlevel events keep tracking query arrival and completion, and log information related to access on pages in the memory buffer during the processing of the query. Page misses correspond to I/O-level events, where a page miss is recorded as a pair of I/O issue and completion events.

Note that application has two access types: foreground access and prefetching. A page accessed in foreground, if missed, is recorded as

I/O issue and completion events. The application often issues a huge prefetching of several pages to improve I/O throughput and avoid long time blocked on I/O. When dealing with prefetching in the memory buffer, it is transformed into access on many several pages and may accompany several I/O issue and completion pairs.

3. Memory Data Access Characteristics

This section analyzes memory data access characteristics of queries. Specifically, this section presents the number of index pages accessed by the query, popularity of index pages, and their access locality. We use foreground access entries for this analysis

3.1 Query Size

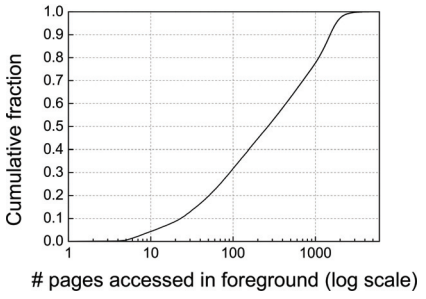

그림 2. 쿼리가 우위 처리 엑세스로 참조하는 메모리 페이지 갯수에 대한 누적분포계수

Figure 2. Cumulative distribution for the number of pages accessed in foreground accesses.

We define the size of a query as the number of pages required for processing that query. If a query is large, many pages should be processed, thus requiring longer time for the query to be finished. This analysis also explains how many I/O operations are potentially needed per request in the worst case.

<Figure 2> shows the cumulative distribution for the number of pages accessed in foreground by queries. The query requires 546.59 memory pages on average to be processed, with median 266 and maximum of 6,175. The median is smaller than the mean, implying that there are some queries that are relatively very large in size. The figure also shows that queries exhibit a wide range of volume of data to be processed. For example, small-size queries (e.g., smallest 10%) process no more than 23 pages while large-size queries (e.g.,largest 10%) process at least 1,510 pages. Therefore, these large queries roughly demand at least 65 times more memory sources than the small queries.

3.2 Page References

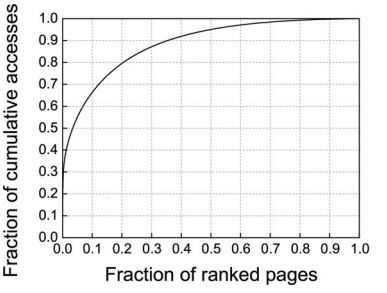

그림 3. 순위에 의해 정렬된 메모리 페이지 참조에 대한 누적분포계수

Figure 3. Cumulative fraction of accesses to the ranked pages.

Indices and associated pages of popular keywords are accessed more frequently. <Figure 3> presents the distribution for the page popularity,

where pages are sorted based on the frequency of access such that the most frequently accessed page is ranked highest, and so on. <Figure 3> shows the cumulative fraction of accesses to the ranked pages. It shows that there exists a high concentration of accesses on popular pages. For example, 10% of most popular pages account for 66.32% of the total accesses, and those 20% account for 79.53% of the total accesses. Unpopular pages are very rarely accessed; 30% pages are accessed equal to or less than 3 times.

An immediate implication of this result is the effectiveness of caching, since caching a small proportion of popular files can lead to a significantly high hit ratio. By perfectly catching up and storing 10% of long-term most popular pages that occupy approximately 650 MB in size, the memory buffer could serve up to 66.32% of page access requests.

3.3 Locality of Page Access

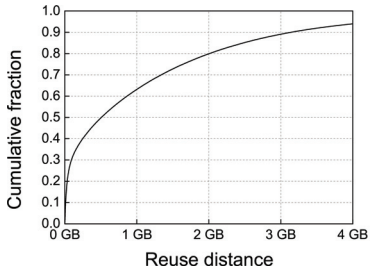

그림 4. 메모리 페이지 재참조 거리에 대한 누적분포계수

Figure 4. Cumulative distribution of page access reuse distances.

Temporal locality. <Figure 4> shows the cumulative distribution of reuse distances for page accesses. The reuse distance is the number of unique pages accessed between subsequent accesses to a particular page. This analysis lets us know the hit rate on the memory buffer that is managed with LRU replacement. The analysis for the reuse distance is performed using 1,048,576 entries of stack, which is equivalent to 4 GB memory buffer (4 KB page 1,048,576 entries = 4 GB). Therefore, this analysis can estimate the hit rate for the memory buffer that ranges from zero to 4 GB in size, which is shown in X-axis in the <figure 4>.

<Figure 4> shows that even with 4 GB memory operating under LRU replacement policy, the maximum hit rate will approximately be 93.9%. This result illustrates that the web search workload exhibits high locality of access, and caching is indeed an effective technique that substantially reduces the amount of disk I/O performed by the server.

In web search, responsiveness of query processing is the first-order design constraint. From our analysis, there are two phenomena that make this challenging. First, the query size is typically large (546.6 pages on average), and small buffer miss rate may have an substantial impact on the query response time. While the prefetching may alleviate the impact, if query issue rate is very high, individual prefetching requests may not be serviced at a proper time due to I/O contention. Second, web search hosts larger and larger index data in multiple I/O devices, and uses relatively less memory to cache them. We expect to see lower buffer hit rate over time, making judicious memory caching policy an important optimization.

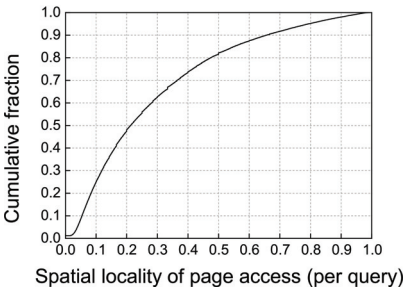

그림 5. 메모리 페이지 참조의 공간적인 지역성에 대한 누적분포계수

Figure 5. Cumulative distribution for the spatial locality of page accesses.

Spatial locality. To analyze the spatial locality for page accesses, we classify access on each page as either sequential or non-sequential. We use LBN (logical block number) distance between successive page accesses for such classification; any access that is within 1024 KB of the preceding access is classified as sequential. This threshold is chosen to be large enough to correctly detect sequential readahead, which on Windows file systems results in a small amount of request re-ordering at the block level, and hits on disk track buffer.

Spatial locality is represented as a fraction of sequential accesses out of the total page accesses at query-level. For example, if a query issues 10 subsequent page accesses after the initial access and 4 of them are sequential, the spatial locality for the query is 0.4. <Figure 5> shows the cumulative distribution for the spatial locality, and shows that queries tend to show overall low spatial locality. We see that half of queries have the spatial locality of 0.21 or smaller, meaning that at least 80% of page accesses in each of these queries are random. On the other hand, 18% of total queries exhibit larger than 0.5, which means that for these queries sequential accesses are more often identified than non-sequential accesses. We also observe that queries with good spatial locality are being interleaved with queries with poor spatial locality. Given a hybrid storage architecture using SSD and HDD, using SSD for serving most of non-sequential access will be a useful optimization.

4. Effectiveness of Prefetching

We will show the basic characteristics of prefetching for query such as size and popularity, and explain its effectiveness in web search workload.

4.1 Locality of Prefetched Data

Processing a query requires access on two types of data: index and metadata. The metadata index tells where each document starts and finishes in a flat index structure, and is accessed when finding matching documents for searched keywords. Since metadata is accessed by every query, its pages are cached in the memory and accessing them does not go to I/O devices. thus,

Prefetching targets on pages containing index data, and there are a large number of pages associated with one prefetching request. In particular, almost all queries issue three or more prefetching IO requests, with each reading 1400−12000 pages.

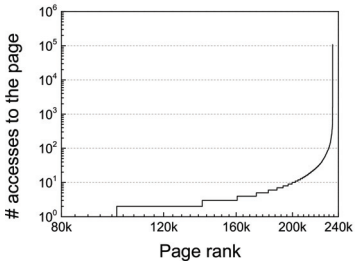

그림 6. 순위에 의해 정렬된 메모리 페이지 참조에 대한 누적분포계수(프리패칭)

Figure 6. Distributions for access frequency according to the ranked pages in prefetching.

Then, what is the popularity of prefetching requests? <Figure 6> shows access frequency for prefetched data, where the least popular pages are ranked first (i.e., highest) and the most popular ones are ranked last (i.e., lowest). We see that there is a significant difference in access frequency between highest ranked data and lowest ranked data, and popular data in lowest rank are frequently prefetched.

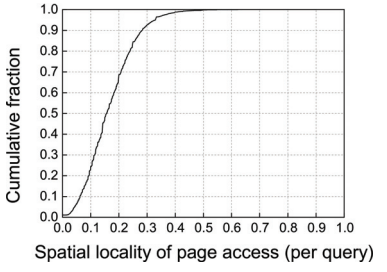

그림 7. 메모리 페이지 참조의 공간적인 지역성에 대한 누적분포계수(프리패칭)

Figure 7. Cumulative distribution for the spatial locality of page accesses in prefetching.

We perform the analysis for spatial locality of prefetched page access similarly to foreground case, and <Figure 7> shows that prefetched data reveal relatively low spatial locality. This is partially because a prefetch request is big in size and one request already contains access on contiguous pages.

4.2 Effectiveness of Prefetching

The effectiveness of prefetching is defined as how many pages that a query access were actually prefetched. We analyze it from two dimensions; 1) query-level, where how many pages accessed in foreground by a query were prefetched during the query execution, and 2) system-level, where how many pages accessed in foreground by a query were prefetched any time in the past (by any query). The system-level analysis assumes an "ideal" case: infinite buffer size where a page, once prefetched, is never going to be evicted. Characterizing under non-ideal case is future work.

Query-level effectiveness. We observe that query-level effectiveness of prefetching is high: almost all of foreground pages were actually prefetched by the same query. This explains that prefetching covers foreground page accesses effectively and would help reduce the impact of blocking on I/Os.

System-level effectiveness. The system-level effectiveness of prefetching is also high. Of the total 1,665,704 foreground pages, 1,665,478 pages were previously attempted in prefetching. Further, our analysis shows that the total number of prefetched pages in the system are 1,803,341, indicating that 8.3% of pages are over-prefetching.

We think this is a consequence of early termination of web search query processing, which is a common optimization to reduce query execution time [3].

5. Conclusion

We characterize real-world workloads for a commercial web search engine, with a focus on locality of index data accesses, their popularity, and effectiveness of prefetching. Analysis results presented in this paper will help system optimizations especially on how to design effective in-memory index data processing.

References

[1] L. A. Barroso, J. Dean, and U. H¨olzle, *Web search for a planet: The google cluster architecture.* IEEE Micro 23, 2, pp. 22-28, Mar. 2003.

[2] J. Dean, *Challenges in building large-scale information retrieval systems*: *invited talk*, In WSDM 2009.

[3] M. Jeon, Y. He, S. Elnikety, A. L. Cox, and S. Rixner, *Adaptive parallelism for web search*. In EuroSys '13, 2013.

[4] M. Jeon, S. Kim, S.-W. Hwang, Y. He, S. Elnikety, A. L. Cox, and S. Rixner, *Predictive parallelization: Taming tail latencies in web search,* In SIGIR, 2014.

[5] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. *The impact of caching on search engines*. In SIGIR, 2007.

[6] F. Blagojevic, D. S. Nikolopoulos, A. Stamatakis, C. D. Antonopoulos, and M. Curtis-Maury. *Runtime scheduling of dynamic parallelism on accelerator-based multi-core systems,* Parallel Comput., 33, pp. 10-11: pp. 700-719, Nov. 2007.

[7] B. B. Cambazoglu, F. P. Junqueira, V. Plachouras, S. Banachowski, B. Cui, S. Lim, and B. Bridge. *A refreshing perspective of search engine caching,* In WWW, 2010.

[8] E. Frachtenberg. *Reducing query latencies in web search using fine-grained parallelism*. In WWW, 2009.

[9] S. Jonassen, B. B. Cambazoglu, and F. Silvestri. *Prefetching query results and its impact on search engines,* In SIGIR, 2012.

[10] Q. Gan, and T. Suel. *Improved techniques for result caching in web search engines,* In WWW, 2009.

[11] C. Macdonald, N. Tonellotto, and I. Ounis. *Learning to predict response times for online query scheduling,* In SIGIR, 2012.

[12] S. Ren, Y. He, S. Elnikety, and K. S. McKinley. *Exploiting processor heterogeneity in interactive services,* In ICAC, 2013.

[13] M. E. Haque, Y. H. Eom, Y. He, S. Elnikety, R. Bianchini, and K. S. McKinley. Few-to-many: *Incremental parallelism for reducing tail latency in interactive services,* In ASPLOS, 2015.

[14] M. J. Jeon, B. C. Jeon, and Y. K. Lee, *Empirical analysis of a large-scale blog workload,* Journal of Knowledge Information Technologt and Systems, Vol. 6 No. 1. pp. 16-26, 2011.

[15] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou, C-Miner: *Mining block correlations in storage systems*, FAST, pp. 173-186, 2004.

웹 검색에 대한 워크로드 특성 분석

전명재[1], 이영규[2]
[1] *마이크로소프트 연구소*
[2] *대전과학기술대학교 사회복지과*

요 약

웹 검색은 많은 서버들로 구성되어 있으며 각 서버는 다수의 검색 쿼리들을 동시다발 적으로 처리한다. 웹 검색에서 요구되는 필수적인 특성은 빠른 응답성이며, 웹 검색에서는 이를 위해 응답속도 SLA가 존재한다. 웹 검색은 각각의 그리고 모든 쿼리에 대해 빠르게 응답을 하여야 하기 때문에, 쿼리들이 시스템 리소스를 어떻게 활용하는지에 대한 이해가 중요하다. 이 연구에서 우리는 서버 시스템이 쿼리에 빠르게 응답하기 위해 가장 필요한 요소 중 하나인 데이터 엑세스에 대한 특성을 분석한다. 특히 연속적인 데이터 엑세스에 대해서 시간적, 공간적인 지역성에 대한 분석 결과를 중점적으로 제시한다. 우리는 본 연구를 통해 웹 검색 데이터 엑세스에서 공간적인 지역성은 부족하지만, 시간적인 지역성이 크고 인기있는 데이터에 대한 많은 엑세스가 있음을 밝힘으로써 데이터 캐슁에 대한 중요도를 알려준다. 또한 이 연구는 웹 검색에서 외부장치 데이터의 접근에 대한 비용을 줄이기 위해 사용하고 있는 대표적으로 최적화 기술중 하나인 데이터 프리패칭에 대한 특성도 제시한다. 우리는 웹 검색의 데이터 프리패칭은 꽤 빈번하고 한번에 큰 데이터를 읽고 있음을 알아냈고, 이런 공격적인 프리패칭이 놀랍게도 꽤 큰 효용성을 가지고 있음을 설명하고 있다. 또한 쿼리 실행을 최적화하는 전통적인 방법중에 하나인 얼리 터미네이션으로 인해 8.3%의 프리패칭된 데이터가 실제로 사용되지 않고 있음을 밝혔다. 우리는 차후 연구로 많은 쿼리를 처리하는 상황에서 쿼리들이 동시다발적으로 발생시키는 프리패칭으로 인한 외부 기억 장치에 대한 병목현상의 위험성을 알아 볼 것이다. 마지막으로 이 연구는 웹 검색 워크로드를 위한 향후 메모리 관리 시스템 연구에 많은 도움이 될 것으로 생각된다.

Myeongjae Jeon received the B.E. degree in computer engineering from Kwangwoon University in 2005, the M.S. degree in computer science from KAIST in 2009, and the Ph.D. degree in computer science from Rice University in 2014. He has been a senior research engineer at Microsoft Research since 2015. He was with ARM R&D from 2014 to 2015. His research interests include systems, computer architecture, and applied machine learning.
E-mail address: myeongjae@gmail.com

Youngkyu Lee received the M.S. degree in the Department of Computer Science from Hannam University in 1994. He received the Ph.D. degree in the Department of Computer Science Engineering from Soonchunhyang University in 1999. He is currently a professor at the department of social welfare in Daejeon Institute of Science and Technology, Korea. His current research interests include parallel processing and database.
E-mail address: leeyk@dst.ac.kr