Exegy at TREC 2007 Million Query Track

Naveen Singla and Ronald S. Indeck
Exegy, Inc.
349, Marshall Avenue, Suite 100,
St. Louis, MO 63119, USA.
{nsingla, rindeck}@exegy.com

October 16, 2007

Abstract

Exegy's submission for the TREC 2007 million query track consisted of results obtained by running the queries against the raw data, i.e., the data was not indexed. The hardware-accelerated streaming engine used to perform the search is the Exegy Text Miner (XTM), developed at Exegy, Inc. The search engine's architecture is novel: XTM is a hybrid system (heterogeneous compute platform) employing general purpose processors (GPPs) and field programmable gate arrays (FPGAs) in a hardware-software co-design architecture to perform the search. The GPPs are responsible for inputting the data to the FPGAs and reading and post-processing the search results that the FPGAs output. The FPGAs perform the actual search and due to the high degree of parallelism available (including pipelining) are able to do so much more efficiently than the GPPs. For the million query track the results for a particular query were obtained by searching for the exact query string within the corpus. This brute force approach, although naïve, returned relevant results for most of the queries. The mean-average precision for the results is 0.3106 and 0.0529 using the UMass and the NEU evaluation tools, respectively. More importantly, XTM completed the search for the entire set of the 10,000 queries on the unindexed data in less than two and a half hours.

1 Introduction

This is Exegy's first year at TREC. Exegy has developed a hardware-accelerated streaming search system, named the Exegy Text Miner (XTM), that enables fast searching of data. The purpose of Exegy's participation in the TREC 2007 million query track was to test XTM's capabilities, with respect to both efficiency and efficacy. Through this process we have learned to express how we can accommodate these sophisticated queries and we hope that it may provide encouragement and insight for others to develop on heterogeneous compute platforms. Given XTM's capability for high-speed massive streaming searching, we decided to query against the raw data itself, i.e., the data was not indexed or otherwise preprocessed for the purpose of searching. We are not suggesting that XTM obviates the need for indexing. In fact, we see XTM complementing indexing. The indexing approach to searching has proved highly successful and is employed widely. However, for applications that require searching in streaming (real-time) data, such as news feeds, indexing adds additional (and unnecessary) latency to the search process. In such a scenario a system like XTM can prove highly beneficial. The same argument applies to quasi-static datasets, i.e., datasets that are updated frequently. Even for data at rest, if the corpus is queried infrequently, then one can fore-go indexing. To reiterate, we envision XTM being used in scenarios where indexing is not the most efficient solution. An application where indexing and XTM can be used together is for data that have indexed metadata. This indexed metadata can be used to trim the dataset and the filtered dataset can be quickly streamed through XTM.

The paper is organized as follows. In Section 2 we briefly describe the architecture and functionalities of XTM. Section 3 describes the experiments we performed on the GOV2 corpus using XTM and discusses the results we obtained. Section 4 concludes the paper.

2 Exegy TextMiner

Exegy TextMiner (XTM) is a streaming text search and data mining engine that combines hardware, software, and firmware to create a powerful solution for quickly searching through massive data stores. XTM is a hybrid system (heterogeneous compute platform) employing general purpose processors (GPPs) and field programmable gate arrays (FPGAs) in a hardware-software co-design architecture optimized for data flow to perform searching and filtering. Thus, XTM combines both the speed of hardware and the flexibility of software to provide a fast, efficient massive search engine. The GPPs handle the input to and output from the FPGAs as well as the post-processing of the results that the FPGA outputs. The FPGA performs the actual searching. Due to the massive parallelism available, the FPGA can perform the searching orders of magnitude more efficiently than a GPP.

XTM includes three search functionalities to address the needs of a real-world search system: exact matching, approximate matching, and regular expression matching. Fig. 2 shows a view of how the search process is structured.

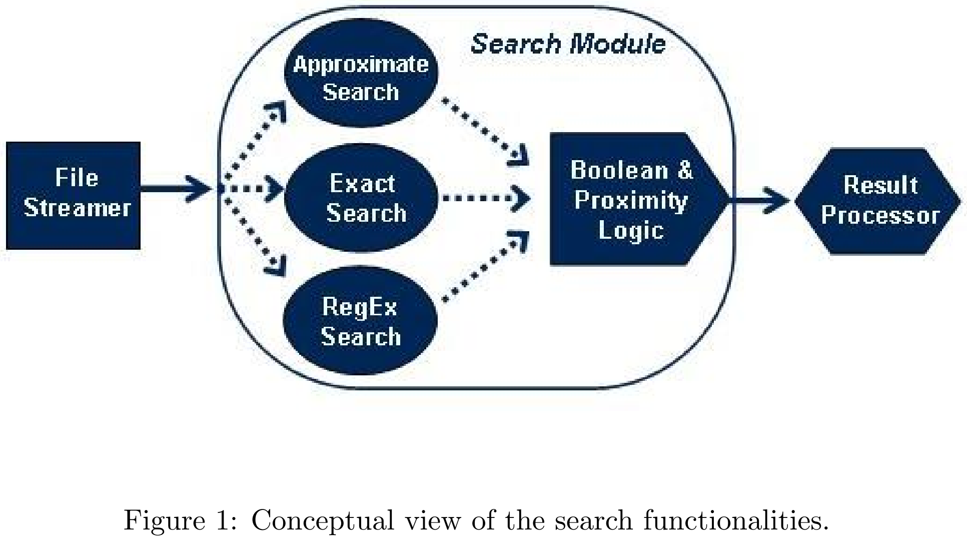

Figure 1: Conceptual view of the search functionalities.

In fig. 2, file streaming is the process of the GPPs controlling the streaming of files in to the FPGA. The FPGA contains the three types of search modules. When the user specifies a search query, the system dynamically loads the appropriate configuration (or configurations) into the FPGA and streams the data set through the FPGA to execute portions of the search algorithm. These search modules can be used in different combinations using the Boolean and proximity operators. The search results are streamed out to the GPPs which then performs various flavors of post-processing depending upon the input query. Following is a brief description of the search modules and the Boolean and proximity operators [1].

- Exact Matching: XTM can search for up to 10,000 exact terms simultaneously. A "term" is defined as an 11 byte string. The exact matching module is essentially a Bloom filter [2] which uses Rabin-Karp hash functions [3]. A Bloom filter is a space-efficient probabilistic data structure that is used to test whether or not an element is a member of a set. Using Bloom filters it is possible that false positives can occur, but false negatives are never possible. The number of false positives increases as the number of elements in the set increases (and all other parameters of the Bloom filter remain fixed). The algorithm is as follows. The terms to be searched for are hashed into a bit-vector position. This hashing is performed by the GPP and the hash table is passed to the FPGA. The Bloom filter is implemented on the FPGA. The text to be searched is streamed into the FPGA, then hashed, and then the pre-computed hash table is checked for the presence of a term. As mentioned earlier, a hit can either be a match or a false positive (hashing collision). In either case, the hit is delivered from the FPGA to the GPP where software determines whether it was an actual match or a false positive. XTM actually uses two different hash functions for the Bloom filter to reduce the number of false positives.

- Approximate Matching: XTM provides the capability to search for terms using approximate or "fuzzy" matching criteria [4]. With approximate matching, terms can be specified with a number of character substitutions. Approximate matching also allows for terms to be specified as case sensitive or case insensitive. Finally, there is provision for wild-carding, where an individual character can be designated as "don't care" and will match any character. This functionality is especially useful for applications that use optical character recognition or speech to text processes as such applications may mistakenly substitute similar letters. There is provisioning for simultaneously searching up to 16 terms using approximate matching.

- Regular-Expression Matching: XTM provides the ability to search for text that matches a set of rules or patterns, such as looking for phone numbers, email addresses, social-security numbers, monetary values, etc. This regular-expression matching can be performed concurrently for up to 50 rules. XTM provides support for the entire PERL regular-expression set. The implementation of the regular-expression matching module is described in more detail in the paper by Brodie, Taylor, and Cytron [5]. Briefly, the simplest and most practical mechanism for recognizing patterns specified using regular expressions is a Finite State Machine (FSM). The regular-expression matching engine uses an array of high-performance FSM processors instantiated in the FPGA to simultaneously search for multiple regular-expressions. In addition, each FSM employs several novel optimizations to maximize the throughput. One of the key optimizations is examining multiple input symbols per transition in the FSM.

- Boolean and Proximity Operators: In addition, XTM also has provisions for complex Boolean, proximity, and position operators to allow using combinations of the search functionalities. Examples of Boolean operators are *OR*, *AND*, and *NOT*. An example of the proximity operator is *NEAR* that allows searching for two strings within a certain distance of each other. For example, the query (query number 85 in the 10,000 query set):

  ```
  (swimming NEAR[10] pools) AND (tacoma NEAR[20] dc)
  ```

  expresses the conditions: the term "swimming" is found within 10 characters of the term "pools"; the term "tacoma" is found within 20 characters of the term "dc"; and both the previous conditions should hold. Software is responsible for Boolean and proximity operations; the FPGA returns results that include position information to the GPP and then software sorts through the results to weed out cases where the Boolean or proximity constraints do not hold.

Along with all these features, XTM operates at a sustained bus-bandwidth limited throughput of about 5 Gb/s. The available bus-bandwidth is 8 Gb/s and the peak throughput of XTM is as-high-as 7.2 Gb/s.

3 Experiments

The task for the million query track was to run a set of 10,000 queries against the GOV2 corpus. The GOV2 corpus is a collection of web data crawled from web sites in the .gov domain in early 2004. The collection is believed to include a large proportion of the .gov pages that were crawlable at that time, including HTML and text, plus the extracted text of PDF, Word, and Postscript files. Any document larger than 256 KB was truncated to that size. Binary files are not included in the data. The GOV2 collection includes 25 million documents for a total size of 426 GB. We tried different combinations of the XTM functionalities to accomplish the searching task in the most efficient manner and to obtain the most relevant results.

The queries posed a lot of challenges. For example, there were several sources of ambiguities in the queries like misspellings or grammatical errors or use of abbreviations for names of U.S. states, there were a few Spanish queries or queries with proper nouns making detection of spelling mistakes hard, and some queries consisted of a single word or a phrase that was very common in the corpus.

The first approach we tried was the prevalent "bag-of-words" approach. The stop words were removed from the queries (we used Oracle's default stop-list for this purpose) and the remaining words were searched in the corpus. Approximate matching was used to deal with misspellings and proved very successful. However, as mentioned in the previous section, the approximate matching module can only deal with 16 terms at a time making its use impractical for the 10,000 queries. Another approach that we tried was that after the removal of all the stop words from the queries, treat the query as a single string and use the proximity operators between the terms of the query. More specifically, we inserted a *NEAR* operator between successive terms of the query and made the search distance twice the length of the query string or 100 characters, whichever was larger, in order to not miss any relevant results. The individual query terms were matched using the exact-matching engine. This approach, however, was again too slow as the Boolean and proximity checking is performed by software which is unable to handle the large number of results returned by the FPGA.

As in the previous approach, it was obvious that the exact matching module of XTM with its simultaneous 10,000-term matching capability was the right choice for the search task. This time, however, we decided not to use any proximity operators. For each query, the exact match engine searched for instances of the exact query string in the corpus. The queries were split into subsets of 1,000 queries (to keep the number of terms under 10,000) and each subset was run against the dataset using the exact matching module. The throughput for each subset of queries was around 4 Gb/s, i.e., for each subset, the entire GOV2 corpus was queried in a little over 14 minutes. Thus, all the 10,000 queries were completed in less than two and a half hours. We would like to point out that single-word queries (such as query 1932, the word "back") and queries consisting of very common phrases (such as query 440, "the department of state") returned a lot of results which took unusually long to process. Such queries severely impacted the throughput. Relevance was calculated by counting the number of occurrences of a query string in the returned documents, i.e., the relevance score was set equal to the number of occurrences of the query string in the document. The relevance calculations were performed in software on the GPP.

Trying to match the entire query string exactly is a rather naïve approach that has the downside that it did not return any results for many of the queries. Out of the 10,000 queries, the exact matching approach returned results for only 3,029 queries. However, again since the whole query was being searched for, the results returned were indeed very relevant. A preliminary investigation of the results shows that the mean-average precision for Exegy's results is 0.3106 and 0.0529 using the UMass and the NEU evaluation tools, respectively.

4 Conclusion

The brute-force approach to searching that Exegy employed for the TREC 2007 million query track had mixed results. Matching the entire query string for the 10,000 queries against the corpus yielded results for only about 30% of the queries, a definite downside. On the upside, the documents that were retrieved for the 3,029 queries were indeed relevant as is apparent from the high mean-average precision obtained using the UMass evaluation tool. Another positive was that the data was not indexed and even then using XTM the entire query process was completed in under two and a half hours. Comparing this to the conventional index generation and searching approach we see situations were XTM has clear advantages. For e.g., streaming data or quasi-static datasets where throughput is a very sensitive issue. For data sets that are queried infrequently XTM can obviate the need to provide extra storage for the index. XTM can also be envisioned as complementing indexing-based search engines in situations where the indexed metadata can be used to pare down the corpus and XTM can stream this reduced dataset to perform the required search. Further research and development on this problem is under way and we plan to use the lessons learned from the million query track to enhance XTM.

Acknowledgment

The authors would like to thank the organizers of the TREC 2007 million query track and the NIST assessors and the other track participants for judging the documents.

References

[1] Benjamin C. Brodie, Roger D. Chamberlain, Berkley Shands, and Jason White, "Dynamic reconfigurable computing," *Proceedings of 9th Military and Aerospace Programmable Logic Devices International Conference*, September 2006.

[2] Burton H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM"*, vol. 13, No. 7, pp. 422-426, 1970.

[3] Richard M. Karp and Michael O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development*, vol. 31, No. 2, pp. 249-260, March 1987.

[4] Mark Franklin, Roger Chamberlain, Michael Henrichs, Berkley Shands, and Jason White, "An architecture for fast processing of large unstructured data sets," *Proceedings of 22nd IEEE International Conference on Computer Design*, October 2004, pp. 280-287.

[5] Benjamin C. Brodie, David E. Taylor, and Ron K. Cytron, "A scalable architecture for high-throughput regular-expression pattern matching," *Proceedings of The* 33rd *International Symposium on Computer Architecture*, vol. 34, No. 2, pp. 191-202, May 2006.