Techniques for Gigabyte-Scale N-gram Based Information Retrieval on Personal Computers

Ethan Miller, Dan Shen, Junli Liu, Charles Nicholas, and Ting Chen
Computer Science and Electrical Engineering Department
University of Maryland Baltimore County
{elm,dshen,jliu,nicholas,tchen}@csee.umbc.edu

In this paper, we discuss the implementation techniques that allowed us to use n-gram based retrieval methods on a gigabyte corpus on commodity personal computer hardware. While such techniques have been used before in word-based systems, n-gram systems have different challenges primarily caused by the significantly larger number of unique terms in the corpus.

Our work shows that using appropriately tuned gamma compression, extensible hash tables and significant amounts of precalculation on the inverted index allows the indexing of a one gigabyte multilingual corpus in a commodity workstation with 256 MB of memory. Response time for full-document queries on this system is approximately 20 seconds for 1 KB documents while providing the same retrieval precision and recall as previous n-gram based systems. We also discuss the space-time tradeoffs we encountered in building a high performance n-gram based retrieval engine. Because of the larger term count - our corpus had nearly 1 million unique terms and over 700 million postings - we deliberately chose methods that reduced space at the cost of increasing retrieval time, primarily through on-the-fly calculations and decompression. We found that, perhaps somewhat counter-intuitively, compressed on-disk indices were actually faster than uncompressed indices because of the reduced time necessary to transfer information off the disk.

1 Introduction

Scientists, researchers, reporters and the rest of humanity all need to find documents relevant to their needs from a growing amount of textual information. For example, the World Wide Web currently has over 320 million indexable pages containing over 15 billion words [1], and is growing at an astonishing rate. As a result, information retrieval (IR) systems have become more and more important. However, traditional IR systems for text suffer from several drawbacks, including the inability to deal well with different languages, susceptibility to optical character recognition errors and other minor mistakes common on the WWW, and reliance on queries composed of relatively few keywords.

The TELLTALE information retrieval system [2] was developed to address these concerns. TELLTALE uses n-grams (sequences of *n* consecutive Unicode characters) rather than words as the index terms across which retrieval is done. By using statistical IR techniques, the TELLTALE system can index text in any language; the current version has been used unmodified for documents in English, French, Spanish, and Chinese. Additionally, n-grams provide resilience against minor errors in the text by allowing matches on portions of words rather than requiring the entire word to match. A third advantage of TELLTALE is that users need not learn query languages and query optimization methods. Instead, they can simply ask for “more documents like the one I’ve got now,” allowing for greater ease-of-use.

Previously, however, the TELLTALE system was unable to index large volumes of text. While traditional word-based IR systems have a number of tricks and tools at their disposal, many of these methods must be modified or discarded when building n-gram based systems. This paper describes our successful efforts to apply traditional techniques to an n-gram based IR system, showing how we adapted traditional IR techniques to n-grams. By using our techniques, we were able to construct an n-gram based IR engine that permitted full-document queries against a gigabyte of text while running on an inexpensive personal computer. These improvements represent a hundred-fold increase in corpus size over previous n-gram-based efforts. Moreover, the compression techniques we adapted from word-based IR systems reduced the size of the index file from seven times larger than the text corpus to approximately half the size of the original text, a fifteen-fold improvement.

2 Background

Our work builds on a large body of research in information retrieval covering both traditional word-based IR systems and systems based around n-grams. In this section, we discuss some of the most relevant previous

This paper appeared at the1999 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, June 1999, pages 1410–1416.

work. Of course, a full treatment of prior work in information retrieval would require a full book (if not more), and such texts exist [3,4].

2.1 Word-based IR systems

There are many information retrieval systems in existence, but space prevents us from mentioning more than our specific system. We will briefly discuss TELLTALE; the reader is referred to [3] and [4] for a more detailed treatment of information retrieval systems.

2.2 N-gram based IR using TELLTALE

TELLTALE [2,5] is a dynamic hypertext environment that provides full-text information retrieval from text corpora using a hypertext-style user interface. The most important difference between TELLTALE and the systems described in the previous sections is that TELLTALE is n-gram-based while the others are word-based. Because of its use of n-grams, TELLTALE has some unique features including language independence and garble tolerance.

An n-gram [6] is a character sequence of length *n* extracted from a document; typically, *n* is fixed for a particular corpus of documents and the queries made against that corpus. To generate the n-gram vector for a document, a window *n* characters in length is moved through the text, sliding forward one character at a time. At each position of the window, the sequence of characters in the window is recorded. For example, the first four 5-grams in the phrase " character sequences…" are " char", "chara", "harac" and "aract". In some schemes, the window may be slid more than one character after each n-gram is recorded.

Most information retrieval systems are word-based because there are several advantages for word-based systems over n-gram based systems. First, the number of unique words is smaller than unique n-grams (for $n>3$) in the same text corpus, as shown in Figure 1. As a result, the index for an n-gram-based system will be much larger than that of a word-based system. Second, stemming techniques can be used in word-based systems but not in n-gram-based systems. However, n-grams provide similar functionality by moving a window one character at a time, enclosing the stem alone at some time.

Third, n-gram based systems don't explicitly remove stop words, though the importance of common n-grams is reduced by standard statistical techniques such as centroid subtraction or the use of TF/IDF weights. Additionally, our n-gram system doesn't make use of synonyms, though there is no reason why it couldn't be modified to do so.

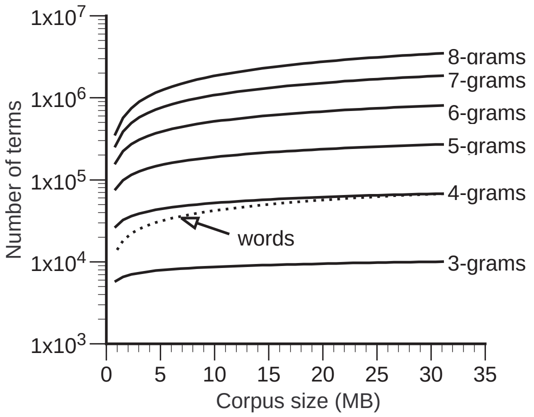

Figure 1. Number of unique terms (words and n-grams) in corpora of varying sizes.

At the same time, there are several advantages for using n-grams. First, a system using n-grams can be more garble tolerant. If a document is scanned using OCR (Optical Character Recognition), there may be some misread characters. For example, suppose "ozone" is scanned as "a zone". An OCR system may not recognize this error, but an n-gram based retrieval engine will still find the n-gram "zone ". From this we can see that using a system based on n-gram technology can provide garble tolerance.

Second, a system can achieve language independence by using n-grams. In most word-based information retrieval systems, there is a language dependency. For example, in some Asian languages, different words are not separated by spaces, so a sentence is composed of many consecutive characters. Grammar knowledge is needed to separate those characters into words, which is a very difficult task to perform. By using n-grams, the system does not need to separate characters into words.

3 Techniques for scalability

Because TELLTALE is n-gram based and the number of n-grams in a document is much larger than the number of words in the same document, the index for TELLTALE is much larger than that of a word-based system. Thus, building a large-scale n-gram based retrieval system is a technical challenge.

All of the performance figures reported in this paper were measured by running on a Silicon Graphics Origin200 with two 180 MHz MIPS R10000 processors, 256 MB of main memory, and 32 KB each of instruction and data cache. While this may seem an impressive machine, it is currently possible to purchase a more powerful computer for under $5000 from commodity PC vendors. Thus, our techniques are applicable to those who can't afford large-scale computers as well as to those who can, and we are currently porting our code to run on Linux-based PCs.

3.1 Textual data used in experiments

To allow practical comparison of various algorithms and techniques, we performed our experiments on real-world collections of data obtained from TIPSTER [7], a DARPA (Defense Advanced Research Projects Agency)-funded program of research and development in information retrieval and extraction. The TREC [8] (Text REtrieval Conference) is part of the TIPSTER Text Program, and provides a very large text data collection. Three types of text data from TREC are used in this paper: a selection of computer magazines and journals published by Ziff-Davis (ZIFF), the Associated Press newswire (AP), and the Wall Street Journal (WSJ). Here we use ZIFF1 to represent the collection from 1989's ZIFF, ZIFF2 to represent the text data from 1988, AP1 to represent the text data from 1989's AP, AP2 to represent the data from 1988 and WSJ to represent the data from 1989's Wall Street Journal.

	ZIFF1	**ZIFF2**	**AP1**	**AP2**	**WSJ**
Documents	75,029	56,903	83,719	78,789	12,046
Unique (thousands)	562.5	498.7	500.0	478.5	268.8
Total (millions)	185.1	134.1	202.6	186.8	31.9
Size (MB)	257	180	260	240	40

Table 1. Statistics for the document collections.

Each corpus is composed of tens or hundreds of individual files averaging one megabyte in length and containing one or more documents. Individual documents within a file are separated by SGML (Standard Generalized Mark-Up Language) tags. The overall characteristics of the corpora on which we ran experiments are summarized in Table 1. Note that the figures for unique and total n-grams are calculated for n = 5; we used this value for n in all of our experiments. We chose n = 5 because this value of n gives reasonable retrieval performance (precision and recall) for English language documents without requiring too much memory.

3.2 Data structures

The primary data structures in TELLTALE are in-memory data structures similar to those used in traditional word-based IR systems. The three hash tables, one for n-grams, one for document information, and another for file information, represent all of the information gathered from the raw text scanned into TELLTALE. While all three data structures are crucial to our system, the n-gram hash table and postings lists consume by far the largest fraction of memory, as Figure 2 shows, and are thus the best candidates for compression. However, we also "compressed" information in the document and file hash tables to reduce overall space requirements.

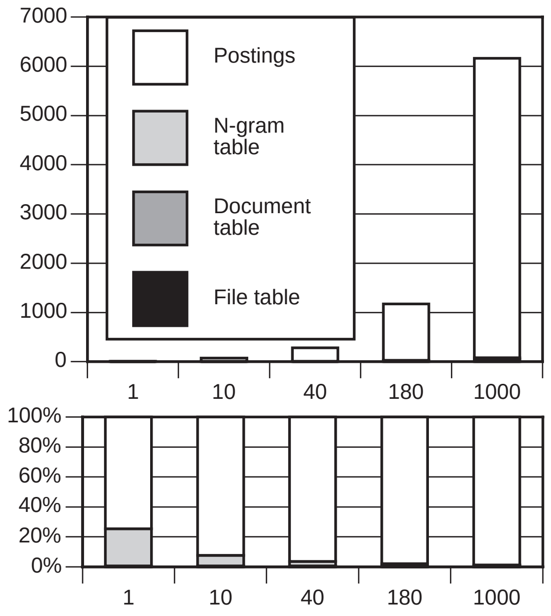

Figure 2. Space consumed by different data structures, assuming no compression.

The file hash table provides a link between documents and the files that contain them. While it would be possible to fold this information into the document hash table, storing it separately results in a large memory savings at little cost because file names are long. For example, a corpus with 500,000 documents of 2 KB apiece might pack an average of 500 documents into each 1 MB file. If file names average 60 bytes in length, the file table requires 60 + 4 = 64 bytes of data and 4 bytes of overhead per file for a total of just 64 KB of storage. On the other hand, storing a file name with each document requires over 3 MB. Thus, large corpora benefit greatly from the savings provided by a separate file table. Additionally, this structure works well for systems that don't use traditional file systems. For example, a system might optimize performance for access via the WWW by consolidating references to a single URL together; pointing all documents from a particular URL to one place would make retrieval and caching simpler.

The n-gram hash table is the central data structure in TELLTALE and, when the postings are included, the one that requires the most memory. This data structure is the one that is hardest to optimize for n-grams rather than documents because of the far greater number of both unique n-grams in the corpus and unique n-grams in a document. For example, a typical 1 MB file from the WSJ corpus has around 300 documents with over 500,000 postings — one posting for each different n-gram in a document. When terms are words, however, the same file has around 60,000 word postings, a reduction of more than a factor of 8. It is this difference that makes it more difficult to build IR systems using n-grams rather than words as terms.

In this version of TELLTALE, each posting consists of a normalized frequency for n-gram *k* in document *i*, a pointer to document *i*, and a pointer to the next posting for n-gram *k*. Thus, a naive implementation requires 12 bytes per posting on a machine that supports 32-bit floating point numbers and 32-bit pointers. It is the space required for postings that consumes the lion's share of memory for corpora of almost any size. In typical documents, the number of unique 5-grams is about 65%-75% of the total number of 5-grams, so a 4 KB document will have 2500 - 3000 unique 5-grams, resulting in $12 \times 3000 = 36000$ bytes of storage. On the other hand, the word count for such a document will total perhaps 800 words with perhaps 400 different words — a reduction of an order of magnitude.

To obtain good performance on the cosine similarity using these data structures, we broke the similarity formula down as shown in Figure 3. Note that, in the final equation, all of the terms with the exception of the first term in the numerator can be precalculated. The remaining term is non-zero only when a term appears in both the query and a document in the corpus. Thus, we can precompute all of the "constant" expressions in the formula for each document, and only need compute the sum of the term frequencies on the fly. Because there are relatively few n-grams in common between any pair of documents, this calculation can be done quickly once the precomputation time has been invested.

$$SIM_C(d_i, d_j) = \frac{\sum_k (f_{ik} f_{jk}) - \sum_k (f_{ik} a_k) - \sum_k (f_{jk} a_k) + \sum_k {a_k}^2}{\sqrt{\sum_k d_{ik}^2} \sqrt{\sum_k d_{jk}^2}}$$

Figure 3. Expanded similarity formula.

As Figure 2 shows, by far the largest consumer of space in an uncompressed index is the postings list. Even for a small corpus of 1 MB, the postings list consumes 75% of the space. For larger corpora, the relative contribution of all other data structures shrinks further. By the time the corpus has reached 1 GB, the postings list consumes over 6 GB of memory, while all other data structures combined use less than 100 MB, or 1/60th the space. This is hardly unexpected — the number of unique n-grams in a corpus grows slowly after the corpus reaches a certain size because the number of unique n-grams in English (or any other language) grows rapidly for the first few megabytes of text but considerably more slowly for additional text. Moreover, some combinations (such as "zzyqv") are unlikely to occur in any documents in a corpus (though this sentence shows that *any* n-gram is possible in any document in a given language...). In essence, the first few documents define the "vocabulary," but later documents add few new n-grams to it. However, the number of postings grows linearly in the number of documents, and consumes far more space than document information or file information, both of which also grow linearly.

Because the postings list was clearly the largest impediment to scaling TELLTALE to handle gigabytes, we spent most of our effort optimizing its usage. The following sections describe our efforts.

3.3 Compression

While we could store uncompressed postings lists on disk, this approach suffered from two drawbacks. First, the resulting index file is huge — often six times the size of the original corpus. Second, large postings lists take longer to read than smaller, compressed, postings lists. Since I/O is very slow compared to CPU instructions, compressing the posting list results in two benefits: lower disk storage utilization and faster similarity computations due to reductions in the time needed to read a bucket. We found that the reduction in I/O transfer time more than outweighed the time needed to decompress the postings lists.

3.3.1 Strategy

The original index file contained postings lists composed a pair of numbers for each document in which the n-gram occurred: an integer identifying the document containing the n-gram and the normalized frequency for the n-gram in the document. This strategy required 8 bytes for each posting, 4 bytes each for the document number and a floating point number for the frequency.

We adopted techniques used in word-based IR systems to compress our postings lists. First, we converted all of the n-gram frequencies into integers by storing the actual count of n-grams in a document rather than the normalized frequency. Since we already stored the number of n-grams in a document, it was a simple calculation to regenerate normalized frequency from the n-gram count and size of the document. Second, we sorted the postings for a particular n-gram by document number so we could store the difference between an individual posting's document number and that of the previous posting. These gaps are smaller in magnitude than the document numbers themselves, and have the additional desirable property that they are smallest for large postings lists with many entries.

These two strategies greatly reduced the size of the index file. Moreover, compression was more effective for n-grams than for words because the distribution of term frequencies is more skewed for n-grams than for words. For example, Figures 4 and 5 show the distribution of term frequencies for 5-grams in the combined 1 GB corpus and the distribution of integers describing the “gap” between document numbers in postings.

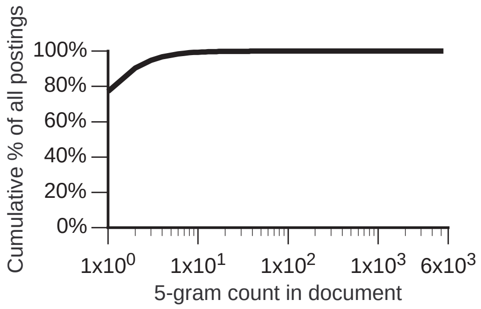

Figure 4. Distribution of 5-gram frequencies in postings.

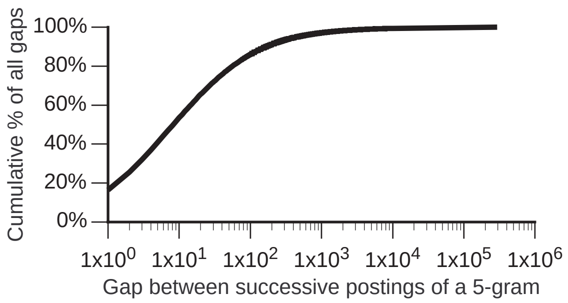

Figure 5. Distribution of gaps in postings.

The opportunities for compression were greatest for n-gram counts — the count of a particular n-gram in a single document is usually a very small number. In the corpora we studied, 97.77% of all postings had counts of 5 or less, while over 77% had a count of exactly 1. We also looked at the distribution of document serial number gaps, shown in Figure 5. This graph shows that most serial number gaps are also relatively small. However, the curve falls off far more slowly than that for posting counts. Half of the gaps were 8 or smaller, and 92.6% were 255 or less. This distribution means that over 92% of all gap values could each be stored in a single byte rather than the 4 bytes required by the default representation.

Based on these findings, we implemented two different compression schemes in TELLTALE. The first was simple to implement and provided reasonable compression for both n-gram counts and document serial number gaps. However, it was considerably below optimal; the `gzip` utility was able to compress the indices by a factor of two. We then switched to gamma compression, yielding files that were approximately the same size as the result of using `gzip` on the original files that used the first compression scheme.

3.3.2 Simple compression algorithm

Based on the statistics discussed in Section 3.3.1, we first considered a simple compression algorithm that saves space for small numbers. We used a single byte to represent numbers from 0 to 2^7-1, two bytes for numbers 2^7 to 2^{14}-1, and four bytes for numbers in the range 2^{14} to 2^{30}-1.

We got good compression results from this scheme. We generated a large on-disk file containing all of the documents in the ZIFF1, ZIFF2, AP1, and AP2 corpora using this compression. The combined corpus has 960 MB of raw text, including 294,440 documents and 889,125 unique n-grams, resulting in an on-disk file requiring 1.085 GB of storage. This provided better than a factor of four compression relative to the uncompressed index file. Additionally, query performance improved greatly.

This simple compression algorithm showed that we can process n-gram queries against 1 GB of text data quite well. However, we did not achieve as much compression as we could. We noticed that `gzip` was able to compress our on-disk files by a factor of 2, suggesting that we could devise a compression scheme that approached, or even surpassed, this level of compression. Doing so would reduce the size of index files and improve performance by reducing the amount of data that must be read for each query, but might increase the CPU time necessary to decompress postings lists.

3.3.3 Gamma compression

Our initial experiments showed that compression was very effective at reducing resource requirements, but that we could achieve additional gains by using gamma compression for our postings lists.

Gamma compression represents an integer *x* as two parts: a unary code for an integer *m* followed by a *k*-bit binary value *y*. The value for *k* is determined by taking the *m*th element of a vector of integers that is constant across all compressed values (i.e., constant for a particular compression scheme). For a vector $\langle k_0, k_1, \ldots, k_n \rangle$, the value of a representation *my* can be calculated as $\left(\sum_{i=0}^{m-1} 2^{k_i}\right) + y + 1$. While other codes such as delta compression do exist, they are more complex and not as efficient for very small numbers such as those found in n-gram frequency counts. Since integers in this system are not big and the gamma code is easy to implement, we picked gamma coding to compress the posting list.

Even after selecting gamma compression, however, we had to choose the best vector to use to compress our integers. To do this, we ran several experiments against the data shown in Figures 4 and 5 to compute the amount of space that would be required using several different vectors. The results of some of our experiments are shown in Table 2. As this table shows, optimal compression for n-gram counts and document gaps were achieved with different vectors. To simplify implementation, we chose the first vector in Table 2 as our compression scheme, though future versions of TELLTALE may use different vectors to compress different value sets. Even with our choice of a single vector, however, we were within 5.5% of the space required by the optimal two-vector compression scheme. Moreover, the optimal scheme for a given corpus can only be discovered by experimentation such as that we performed on our 1 GB corpus. Since the optimal choice for any particular corpus may be different, we chose a simple scheme that performed well.

Vector	N-gram counts (MB)	Document gap (MB)	Total (MB)
<0,1,2,...,11,12, 14,16,18,20,28>	87.7	409.1	496.8
<0,0,1,2,...,7,8, 10,12,14,16,18,28>	83.2	435.0	518.2
<0,0,0,1,2,...,7,8, 10,12,14,16,18,28>	82.3	460.2	542.5
<0,2,4,6,8,10,12, 14,16,18,28>	96.6	388.1	484.7
<0,2,3,6,9,12,15, 18,21,28>	96.0	398.5	494.5

Table 2. Index sizes for gamma compression vectors.

Using our gamma compression scheme with the vector in the first line of Table 2, we generated a large index file covering ZIFF1, ZIFF2, AP1, and AP2. The raw text from these files was 960 MB, but the on-disk index consumed only 647 MB. While the table entry suggests that under 500 MB would be necessary, the table does not include additional data structures necessary to store document info and the n-gram headers; these structures make up the additional 150 MB. Since they are only read in at startup, however, we decided not to attempt to compress them. Our experiment gave a compression ration of 0.67, which is nearly as good as `gzip`.

Gamma compression also improves query performance by reducing the amount of data that must be read for a single query. After loading the in-memory information for 960 MB of text, TELLTALE can compute and sort about 300,000 documents' similarity result at the rate of 50 seconds per thousand characters in the query.

3.3.4 Handling gamma compressed postings lists in memory

With the help of gamma compression, the compressed postings list is small enough to be loaded into today's main memories if allocation is handled intelligently. Since about 1 GB of raw text data can be indexed in under 700 MB, we can handle 1 GB of text data in less than 750 MB of main memory, allowing for a small amount of overhead. Doing so improved the performance by eliminating disk I/O during a query, though the improvement was not as large as we had expected.

The major difficulty with handling the compressed postings lists in memory is coping with the many bucket capacities necessary — some postings lists will be only a few bytes long, while others may require many thousands of bytes. Additionally, these buckets must grow dynamically as new documents are scanned in. These requirements are best met using lists built from fixed size "chunks" of space connected in a linked list. The overhead for this scheme is relatively small — fixed size chunks that can hold 32 bytes require only 4 bytes of pointer overhead for an overhead of 12.5%. In addition, fixed size chunks waste some space because part of the last chunk is unused. On average, this will waste half of a chunk per n-gram, 16 bytes in our system. Thus, total overhead for a system that scanned in the 1 GB AP-ZIFF corpus would be 14 MB for unused chunk space and about 62.5 MB for pointers. In future versions of TELLTALE, we will attempt to reduce this overhead by allowing multiple chunk sizes for maximum efficiency.

Operation using in-memory compressed postings lists requires that the lists must be uncompressed before they are used in similarity calculations. While this technique uses less memory than uncompressed postings lists, it is somewhat slower because of the time needed to uncompress a postings list. A 200 MHz Pentium laptop is capable of decompressing two million integers a second; while this seems an impressive number, most similarity calculations must process ten million postings or more. Thus, decompression time contributes significantly to similarity calculation time.

By using in-memory gamma compression rather than uncompressed postings lists, TELLTALE can reduce its memory usage by a factor of four or more without discarding information. Note, however, that the original TELLTALE uses slightly less memory for small corpora; this occurs because the overhead for the gamma compression version of TELLTALE is slightly higher. However, this higher overhead is more than recovered as the amount of indexed text increases.

The in-memory gamma compressed version also provides increased speed compared to the on-disk version. Comparisons with the original, uncompressed in-memory version are less relevant because the original version can only handle very small corpora. Thus, we focused our attention on the relative performance of the gamma compressed postings lists on disk and in memory. We ran queries against a collection containing the 257 MB of text in ZIFF1, which comprise 75,029 documents, 562,492 unique n-grams, and a total of 185,159,683 postings. We were limited to this size because the machine on which the queries were run, a two processor SGI Origin 200, had only 256 MB of physical memory, and the use of virtual memory would have resulted in unacceptably slow performance due to excessive paging. When the entire ZIFF1 corpus was loaded into memory, it used 210 MB of memory, leaving the rest for operating system use. As can be seen in Figure 6, in-memory gamma compression is twice as fast as on-disk gamma compression, though the difference is not as large as we had expected. This may be due to the high performance XFS file system used on the SGI server on which the experiments were run.

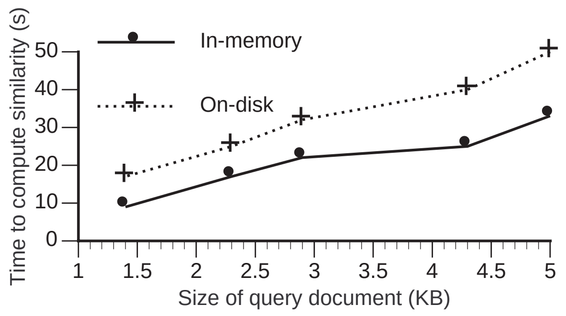

Figure 6. Performance comparison of in-memory and on-disk version with gamma compression.

As the preceding experiments have shown, gamma compression performs well for n-gram-based IR just as it does for word-based schemes. However, we had to adjust the gamma compression vectors to best compress the postings lists generated for n-grams because document gaps and, particularly, occurrence count distributions are different between words and n-grams. Using these techniques, we expanded TELLTALE's capability from around 10 MB to over 1 GB while maintaining good query performance.

4 Conclusions and future work

We greatly expanded TELLTALE's capacity and improved its performance, without compromising its ability to handle multilingual or slightly garbled documents. It is these advantages, combined with an ability to perform retrieval using full documents rather than relatively short queries, that make TELLTALE a useful tool. However, there is still much work to be done with it. Because TELLTALE is the first system using n-grams that can handle a gigabyte of text, we hope to be able to show that n-grams are equal to or better than words as indexing terms. We are also currently performing experiments in using TELLTALE to index collections in non-English languages ranging from European languages such as French and Spanish to ideogram-based languages such as Chinese. Our preliminary results are promising, but more investigation needs to be done.

Acknowledgments

The authors are grateful to the many people who contributed to this paper by giving us feedback and suggestions, including Claudia Pearce and Bill Rye at the Department of Defense and David Ebert at UMBC. Our research was supported in part by funding from the U.S. Department of Defense.

References

[1] Steve Lawrence and C. Lee Giles, "Searching the World Wide Web," *Science* **280** (3), 3 April 1998, pages 98 - 100.

[2] Claudia Pearce and Ethan Miller, "The TELLTALE Dynamic Hypertext Environment: Approaches to Scalability," in *Advances in Intelligent Hypertext*, J. Mayfield and C. Nicholas, eds. Lecture Notes in Computer Science, Springer-Verlag, October 1997, pages 109 - 130.

[3] Robert R. Korfhage, *Information Storage and Retrieval*, John Wiley & Sons, 1997.

[4] Ian H. Witten, Alistair Moffat, and Timothy C. Bell, *Managing Gigabytes*, Van Nostrand Reinhold, 1994.

[5] Claudia Pearce and Charles Nicholas, "TELLTALE: Experiments in a Dynamic Hypertext Environment for Degraded and Multilingual Data," *Journal of the American Society for Information Science*, April 1996, pages 263 - 275.

[6] Marc Damashek, "Gauging Similarity with n-grams: Language-Independent Categorization of Text," *Science* **267** , 10 February 1995, pages 843 - 848.

[7] Donna Harman, "The DARPA TIPSTER project," *ACM SIGIR Forum* **26** (2), Fall 1992, pages 26 - 28.

[8] The Text Retrieval Conference. Information available at `http://trec.nist.gov`.