Improving Efficiency of Indexing by Using a Hierarchical Merge Approach

Aiqun Du and Jamie Callan
Computer Science Department
University of Massachusetts
Amherst, MA 01003

Abstract

As electronic collections become larger and more numerous, there emerges the problem of scalability and efficiency with building very large databases for information retrieval system. The speed performance of the old merger which is part of the INQUERY indexing system degrades significantly when used in building big databases under main memory space limitation. For this project we found a better solution by using a hierarchical merge approach. Timing tests on new merger indicate that merge time can be reduced significantly. Hierarchical merge nicely solves the problem of scalability and greatly improves the efficiency with building very large databases. A description of this new method, major experimental results, and preliminary conclusions are presented in this report.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1998		**00-00-1998 to 00-00-1998**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Improving Efficiency of Indexing by Using a Hierarchical Merge Approach	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Massachusetts Amherst,Department of Computer Science,140 Governors Drive,Amherst,MA,01003**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**9**	

1 Introduction

The likelihood of encountering a very large text corpus in practice is encouraging the development of text retrieval techniques that can work with them. As time passes and electronic collections become larger and more numerous, systems capable of working with very large collections will be more and more in demand. On the other hand, computer main memory is relatively limited and constrained compared to the amount of data in a large collection. The requirement for larger memory while building big databases can sometimes make this resource a bottleneck for an information indexing system.

INQUERY is a state-of-the-art, widely used, full-text information retrieval system ([5] [4] [2]). The INQUERY document indexing system is described in detail in [3] [1]. The overall indexing process consists of two main operations: parsing and merging. The subsystem responsible for parsing is called the *Parser*. It creates partial inverted lists by scanning, lexically analyzing, and inverting documents. A partial inverted list contains document entries for a subset of the documents in the collection. It must be combined with other partial inverted lists for the same term to create a final inverted list for the document collection. The Parser buffers partial inverted lists in main memory and flushes them to intermediate files when the buffer is full. The subsystem responsible for merging is called the *Merger*. After all of the documents have been parsed, the Merger combines the intermediate files to produce the final inverted lists for the collection.

The aim of this project is to solve the efficiency and scalability problem of the merger of INQUERY indexing system. The speed performance of old merger (`merge_btl`) degrades significantly when used in building big databases under tight memory space limitation. For example, for a 20GB collection containing 50 intermediate files, each of which has 1000 blocks of partial inverted lists, given a 100MB buffer space, it will take merge_btl about 2 weeks to run. This is unacceptable for a practical IR system.

We found a better solution by using a hierarchical merge approach. Timing tests on new merger indicates that merge time can be reduced significantly. Hierarchical merge nicely solves the problem of scalability and greatly improves the efficiency with building very large databases for information retrieval systems. A description of this new method, major experimental results, and preliminary conclusions are presented in this report.

2 Hierarchical Merge

2.1 Old merger – merge_btl

An intermediate file produced by the parser will contain one or more blocks of partial inverted lists, where each block corresponds to a batch of documents. The partial inverted lists within a block are complete inverted lists for the documents indexed during the corresponding batch. To build final inverted lists for the entire document collection, the partial inverted lists from all of the blocks must be merged [1].

Merge_btl is performed in main memory by allocating an X bytes merge buffer and dividing it evenly among all of the intermediate (imd) file blocks. If there are Y imd file

blocks, the merge buffer can be filled using Y disk reads. Each disk read will do a data transfer of X/Y bytes. The merge buffer provides an interface to the imd file blocks for the merger. Once the merge buffer has been primed from the imd file blocks, it starts to merge the partial inverted lists within all the blocks. This process iterates until all the partial inverted lists have been read in and merged.

The problem with merge_btl is that if there are many imd blocks to merge and not much memory space available, it can lead to very small buffer size allocated for each block. For example, for a 20GB collection containing 50 intermediate files, each of which has 1000 blocks, there are 50,000 imd blocks in total. If they are merged in a batch, given a 100MB buffer space, each imd block has only 2KB buffer. This will cause the process to do excessive I/O on a machine with a medium amount of memory such as 32MB. Because a read in a typical UNIX file system causes 8KB to be read from the disk, but we only have a 2KB buffer, the remaining 6KB is discarded. Thus too many I/Os will extend the merge time significantly.

In order to have a clear picture of how the current system behaves, experiments were done with merge_btl: run it for different block buffer sizes and record the wall clock time and number of disk I/Os, as shown in Figures 1 and 2. These tests were done on mildura (DEC Mips with 32MB RAM), with 0.5MB TREC collections (including `ap.dat` and `ziff.dat`), containing 517 imd file blocks, using 0.5MB buffer space. Note that under these conditions the OS might cache I/O, so the timing results might still be too fast.

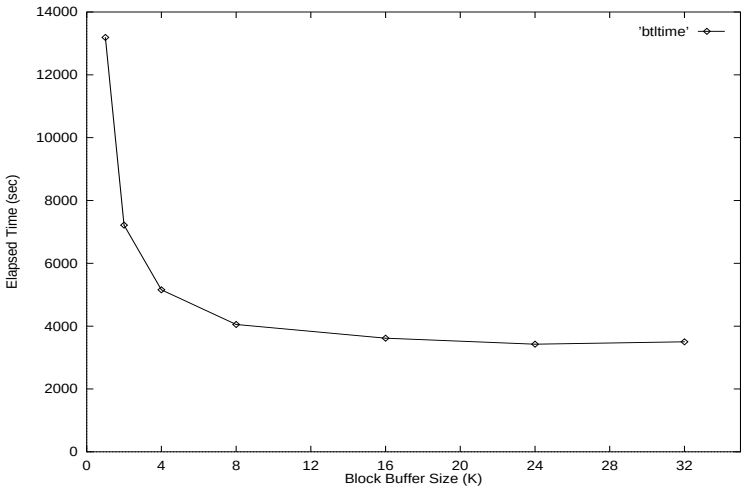

Figure 1: merge_btl: merge time vs. block buffer size

These results indicate that execution time decreases sharply when the block buffer size increases from 1KB to 8KB, and levels off after 8KB, suggesting that 8KB is a critical point. Since the UNIX file system typically transfers 8KB block per read, providing a smaller-than-8KB buffer leads to part of the data be discarded and has to be re-read in later, which is a waste of I/O processing time. Comparing the curves of execution time and I/O, we can see that the more I/Os, the more running time. I/O constitutes a significant part of total merge time. So we should make the merger perform as little I/O as possible, and also never allocate less than 8KB buffer per block.

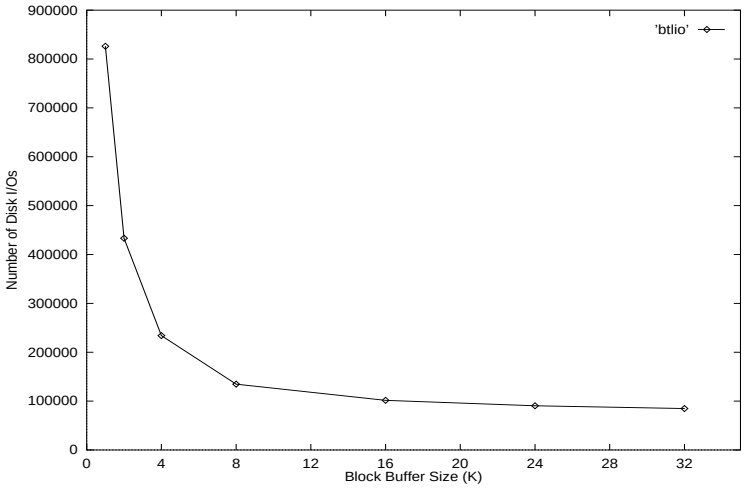

Figure 2: merge_btl: number of disk I/O vs. block buffer size

2.2 New merger

The new approach is a hierarchical merge of intermediate files. There are two possible types of implementation for this appoach:

- Take n intermediate files as a batch and merge them into one file. This step iterates until the one last batch for which the system does a simple call to merge_btl. The question is how to decide n?
- First "tidy" each intermediate file, that is, merge all the blocks in one file so that each file is one block. And then merge these files hierarchically. We need to find out if, when each file has just one block, there is some optimization that wouldn't be possible otherwise?

Correspondingly two versions of new merger were developed. Two basic guidelines govern our implementation:

- Buffer size should never be smaller than 8KB per block. An upper limit N is set on the number of blocks in a merge batch to guarantee that each block has at least 8KB buffer. If there are more blocks in a single file than would allow 8KB per block, it will merge the first N blocks, write the merged block out, then the next N blocks, write it out, ..., and the output is one imd file with a smaller number of larger blocks. N is calculated as the quotient of buffer space size divided by the total number of imd blocks.
- New merger only does I/O in multiples of 8KB, because in UNIX, each I/O reads 8KB. For example, if it could read 8,200 bytes, have it read only 8,192 bytes.

To test whether the buffer size for each block should be 8KB or bigger (multiples of 8KB), we implemented newmerge_10, newmerge_20, newmerge_25, newmerge_100, etc., which merge at most 10 blocks, 20 blocks, 25 blocks, 100 blocks, etc., in each batch. This introduced

another batch limit number M. The actual upper limit of number of blocks to merge in each batch is determinted by choosing the minimum between N and M.

New merger uses more disk space than merge_btl, to store temporary imd files generated during the intermediate steps of hierarchical merge. These temporary imd files are automatically removed after successful completion of the merge.

3 Experiments

3.1 Platform and Collections

All the experiments described in this section were run with login disabled on adelaide, which was a DEC 3000/600 running OSF/1 V3.0. The system was configured with 64MB of main memory and 256MB of swap memory. All of the data files and executables were stored on the local disk, and a 64MB "chill file" was read before each merge run to purge the operating system file buffers, to guarantee that no data was cached by the file system across runs. If the machine is chilled in this way, we can get nearly the same time when we run the same experiment twice in a row [1]. All times reported were measured with the GNU `time` command.

The experiments with new merger were done with the 1GB TREC volume 1 and 2GB TREC volume 2. Table 1 shows the number of imd file blocks, with various imd block sizes, for the 15 TREC collections.

collection	*8M*	*5M*	*2M*	*1M*	*0.5M*
ap	16	25	67	137	299
ap2	15	23	47	127	276
doe	13	19	40	104	222
fr2	7	11	22	58	123
fr_a	6	8	17	44	94
fr_b	4	6	11	30	63
wsj87	8	12	25	67	144
wsj88	7	11	22	58	125
wsj89	3	4	7	20	42
wsj89b	1	1	1	1	2
wsj90	5	7	14	37	79
wsj91	8	13	26	70	151
wsj92	2	3	6	16	35
ziff	12	19	39	102	218
ziff2	9	14	28	74	157

Table 1: Number of imd file blocks with various imd block sizes.

3.2 Results and Analysis

Timing test results for the 2GB collection are shown in Figures 3 to 6. Different methods of new merger, including tidy, newmerge_100, newmerge_25, newmerge_20, and newmerge_10, were tested given various sizes of buffer space, and on 8MB, 5MB, 2MB, and 1MB imd file block sizes, respectively. We also run newmerge_* with different imd block sizes in order to test small n (the number of intermediate files in a merge batch).

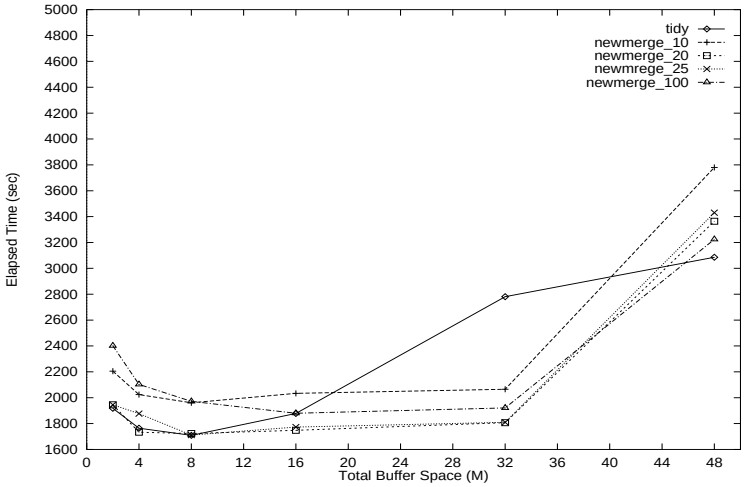

Figure 3: Test new merger with 8M imd file blocks

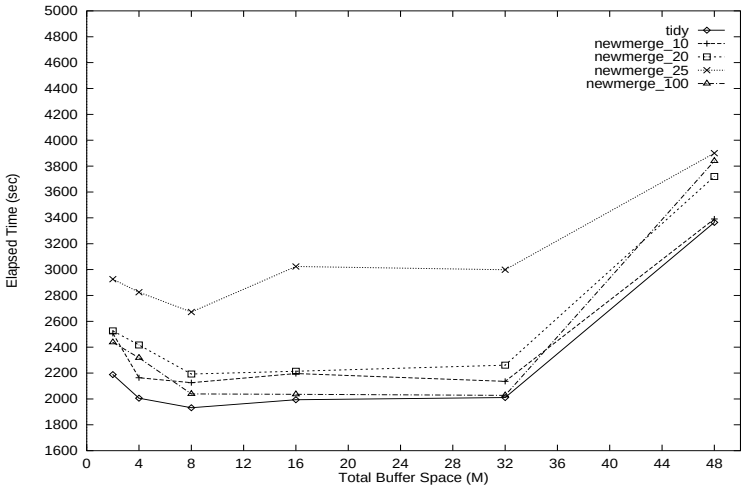

Figure 4: Test new merger with 2M imd file blocks

By analyzing the test results, we learned that there are three major factors that can affect the execution time of new merger:

1. Merge method

   - In most cases tested, tidy runs faster than newmerge_100, newmerge_25, newmerge_20, and newmerge_10. When using just one disk, tidy performs the best because it

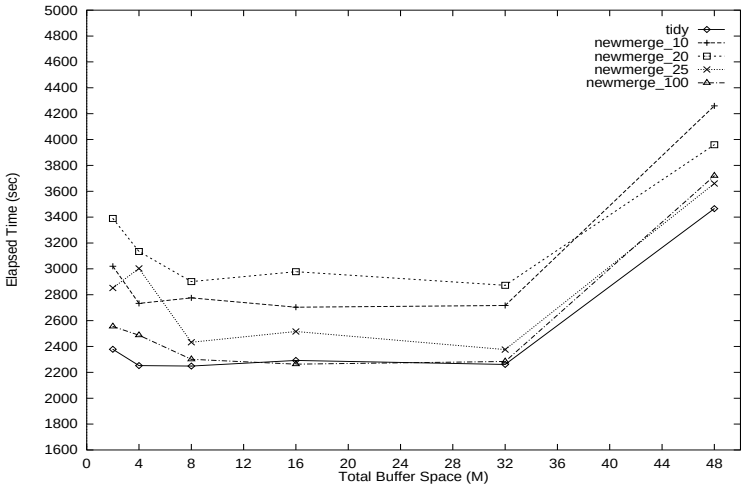

Figure 5: Test new merger with 1M imd file blocks

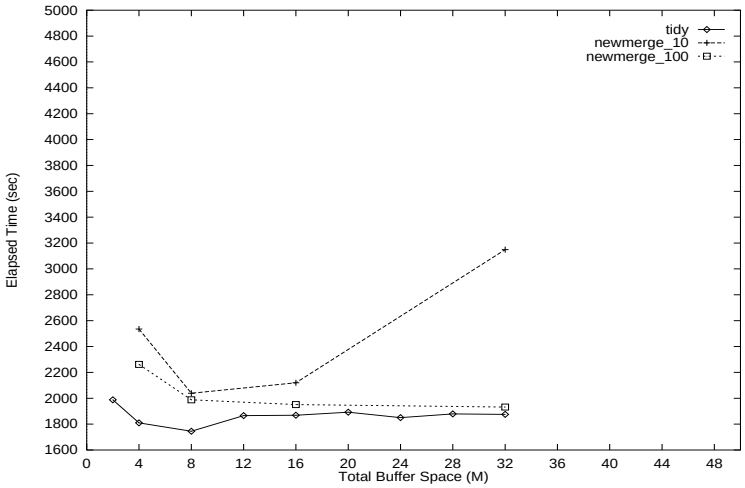

Figure 6: Test new merger with 5M imd file blocks

keeps the disk head in one region of the disk (faster seeks), and because the operating system I/O caching is better able to handle within-file accesses. However, if most of the imd files contain just one or two blocks each, tidy would be unnecessary and some other method, such as newmerge_100 should be chosen instead.

- Newmerge_100's performance is relatively close to tidy when buffer space is large (16MB–32MB), and it runs faster if given 16MB buffer space. Newmerge_25 and newmerge_20 can be as fast as tidy in the case of 8MB imd block size and 8MB buffer space. This is because for big imd block size like 8MB, newmerge_25 and newmerge_20 actually have a similar merge pattern to tidy. But their overall performance is much slower than tidy and they generally require more disk space than newmerge_100. Newmerge_10 tends to be slower and the most disk-consuming, as it needs to do more hierarchies of merge than other methods and therefore generate more temporary imd files.

- Buffer size for each block should be big, i.e., multiples of 8KB rather than just 8KB. This is another reason why the tidy approach works better.

2. Imd file block size

   - There appeared a trend that, the larger imd file block sizes, the less merge time. Among experimenmts with 8MB, 5MB, 2MB, and 1MB imd file block sizes, new merger with 8M imd blocks generally yields the quickest performance over all cases. Therefore, when run the parser (`inparse`) to parse the source documents, we choose to set its buffer size argument so as to generate imd file blocks of big size, for example around 8MB.

3. Merge buffer space

   - For a 2GB database, new merger generally runs fastest if given 8MB–16MB buffer space.
   - Paging begins at about 32MB, and is a problem at 48MB. That causes the timing curve rise for buffer sizes above 32MB. We should avoid specifying larger than 32MB buffer space running new merger on similar machines (e.g., machines with 64MB of memory).

4 Conclusion

A hierarchical merge approach solves the scalability problem for building very large databases, and significantly improves the speed of indexing for information retrieval systems. Compared to merge_btl, the new merger reduces the merge time by 5 times under a worst case buffer space limitation. Moreover, it solves the problem with merge_btl due to the OS constraint on the number of files that can be opened at the same time (maximum 1024 file descriptors). When the number of intermediate files exceeds the upper limit, merge_btl simply cannot be used since it tries to merge all the files simultaneously. Obviously this is not a problem with the hierarchical merge.

Experimental results show that, for building 2GB database, the optimum merge performance is tidy using 8MB buffer space, or newmerge_100 using 16MB buffer space, and with 8MB or 16MB imd file block size. This is recommended as the default method for merge tasks on similar system configurations.

The new merger has been tested on very large collections. The success of building the 20GB TREC VLC (Very Large Corpus) strongly demonstrated its good performance. It took 4 hours 40 minutes to merge the intermediate files of the VLC collections into a single index. There are 1,514 imd blocks in total in the 697 intermediate files. newmerge_100 was used with 16MB buffer space.

More experiments and deeper OS-related knowledge are needed to further explain some of the previous experimental results and to gain a better understanding of the inner behavior of merge system.

Acknowledgements

This material is based on work supported in part by the National Science Foundation, Library of Congress and Department of Commerce under cooperative agreement number EEC-9209623,and also supported in part by United States Patent and Trademark Office and Defense Advanced Research Projects Agency/ITO under ARPA order number D468, issued by ESC/AXS contract number F19628-95-C-0235. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsor(s).

References

[1] Eric W. Brown. *Execution Performance Issues in Full-Text Information Retrieval.* PhD thesis, University of Massachusetts at Amherst, October 1995.

[2] J. P. Callan, W. B. Croft, and J. Broglio. TREC and Tipster Experiments with INQUERY. *Information Processing & Management*, 31(3):327–343, 1995.

[3] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *Proceedings of the Third International Conference on Database and Expert Systems Applications*, pages 78–83, Valencia, Spain, 1992. Springer-Verlag.

[4] H. R. Turtle and W. B. Croft. Efficient probabilistic inference for text retrieval. In *RIAO 3 Conference Proceedings*, pages 644–661, Barcelona, Spain, April 1991.

[5] H. R. Turtle and W. B. Croft. Evaluation of an inference network-based retrieval model. *ACM transactions on Information Systems*, 9(3):187–222, 1991.