Trends in Retrieval System Performance

Justin Zobel Hugh E. Williams Sam Kimberley
Department of Computer Science, RMIT University
GPO Box 2476V, Melbourne 3001, Australia
{jz, hugh}@cs.rmit.edu.au, sam@mds.rmit.edu.au

Abstract

Computer technology is continually developing, with ongoing rapid improvements in processor speed and and disk capacity. At the same time, demands on retrieval systems are increasing, with, in applications such as World-Wide Web search engines, growth in data volumes outstripping gains in hardware performance. In this paper we experimentally explore the relationship between hardware and data volumes using a new framework designed for retrieval systems. We show that changes in performance depend entirely on the application: in some cases, even with large increases in data volume, the faster hardware allows improvements in response time; but in other cases, performance degrades far more than either raw hardware statistics or speed on processor-bound tasks would suggest. Overall, it appears that seek times rather than processor limitations are a crucial bottleneck and there is little likelihood of reductions in retrieval system response time without improvements in disk performance.

1 Introduction

The rate of improvement in computer technology is remarkable. Each year processors are faster, caches larger and more complex, and the capacity of disks and memory larger. These developments in hardware have allowed standard desktop workstations to be used, with little modification, for high-performance applications such as file systems, web serving, and—of particular interest to us—retrieval systems, including database systems and text retrieval engines.

A decade ago, database systems were largely confined to mainframe computers, and the idea of a text retrieval system on a personal computer was simply implausible: for example, indexing 800 Mb of text required 23 days on a minicomputer, while resolution of a query on the same machine required 4 seconds [7], a result so good that it merited publication. In contrast, a standard desktop machine could now index a gigabyte in under an hour and resolve a query in well under a second [14]. These improvements are due to advances in both algorithms and technology—as we show below, the hardware alone gives an improvement in query resolution time of about a factor of 5.

It is well-known that any improvement in hardware is quickly matched by new software requirements. In 1999, just as in 1991, approximately $2000 buys a desktop machine minimally configured for running the latest releases of popular operating systems and word processors [10]. While such trends have little impact on the performance of a well-implemented retrieval system, another related trend is of importance: the explosive growth in the volume of data available for storage and retrieval. The most conspicuous source of this data is the web, but it also arises in commercial applications such as data warehousing and the computerisation of paper-based processing, and in scientific projects such as the Human Genome project, satellite telescopy, and interplanetary missions. Advances in hardware have yielded substantial performance gains for retrieval systems. In combination with the growth in stored data, however, not only is the performance of retrieval systems affected in unpredictable ways, but choices of algorithms can change.

In this paper we experimentally investigate how retrieval system performance is affected by changes in hardware and in data volume. We had expected to observe that the net effect was approximately neutral—that is, that response times would not have changed. However, taking a series of basic Sun SPARC systems from over the last decade we show that increasing data volumes are, for some applications, greatly outpacing improvements in hardware performance, and, even when the data is constant, in some cases improvements in hardware performance have made little difference to response time. In other cases, assuming data growth roughly equals processor improvements and with the low improvement in factors such as disk access rates, there are slight overall gains. We use these experiments to identify current bottlenecks, that is, the aspects of system performance where improvements to algorithms are most

required. Our experiments also demonstrate that retrieval systems need their own benchmarks for hardware.

We first examine trends in hardware performance and data volumes, then consider in abstract the likely impact of these trends on text retrieval systems. We then describe our experimental testbed, and give results for a variety of kinds of retrieval: in-memory algorithms, queries where most costs scale linearly with increasing data size, and typical information retrieval queries, where some significant costs are independent of data volume.

2 Trends in hardware and data

Hardware performance continues to improve. Processor speeds are doubling every 18 months or so. The cost of memory continues to fall: the volume of memory installed in a typical machine doubles about every two years. The rise in disk capacity is perhaps the most dramatic of these improvements, with, for example, the size of a typical disk in a notebook computer growing by a factor of 100 between 1992 and 1999—that is, doubling in size every year, and well exceeding the historial rate of annual increase of 27% [3].

Other improvements in hardware have, however, been more modest. Disk latency and seek times have only improved slowly, estimated at about 8% a year up to 1994 [3] and, arguably, more slowly since then: for cheaper drives, the rapidly increasing density of data storage has forced manufacturers to slow disks down, lest the rate of data flow off disk exceed bus capacity. The rate of data flow off disk can be further increased through RAID arrays and striping, which however only highlights the delays imposed by latency and seek times. Busses have become faster and wider, but have not quite kept pace with disk and processor developments. Likewise, memory access times have dropped, but not as rapidly as processor speeds have grown: the number of cycles spent waiting for a byte to return from memory has gradually increased, motivating the use of large in-processor caches in even modest machines. Overall,

> technology changes unevenly ... [Over the last 30 years] memory costs decreased by 10,000 as densities increased more than a million times. For disks, advances in magnetic storage have reduced cost per byte by a factor of a million, yet disk access [speed] has probably changed by less than a factor of 10. [5]

Quantifying the trends for data volume is not straightforward. While CPU speed lends itself to easy analysis, by for example plotting clock rate against release date, collection size is more problematic. Arguably the best-known large collection, the documents that comprise the Web, is growing extremely rapidly [8]. A related collection, Internet news items, is growing more slowly, but the number and volume of mailing lists (which supplement Internet news and greatly exceed news in volume) is unknown. The volume of text data held by newspapers grows only linearly—a typical newspaper might add 100 Mb a year—but the volume of image and video data held by news organisations is dramatically increasing. Current legislation, another application of text retrieval, changes little in size from year to year, but the volume of court records continues to grow (and in Australia is now stored as digital audio recordings as well as text transcripts). The data generated by the Human Genome project has roughly doubled in volume every 12 to 15 months since the early 1990s, and over the next few years is projected to grow much more rapidly as new gene sequencing technology becomes available. It is likely that this growth will continue or accelerate over the next five years [4].

Data growth is as much due to the appearance of new applications as to growth in old ones. The World-Wide Web barely existed five years ago, while data warehouses are another technology that has led to a leap in data volumes; only the uses of the technology will dictate how rapidly data volumes will continue to grow. Business data collection, for example through the widespread use of credit and fly-buy cards, is another new area. Many businesses are in the process of introducing computer technology for applications that were previously paper-based or manual, such as document production [12]. In other cases, the increasing availability of mass storage technology is changing the kinds of data that is gathered. In stock markets, for example, it was until recently customary to record only the closing value of each stock; some markets now record stock values at intervals of a few minutes, or even at each purchase, and the number of purchases has greatly increased in the last decade.

Overall, it is clear that much more data is being held online than was the case a few years ago, and the volumes are continuing to grow. It would be brave to predict that this growth will soon cease: that would be tantamount to predicting that no new applications will appear. After all, it is not many years since Internet news was a significant proportion of net traffic; today it is insignificant. Quantifying the growth in data volumes would require averaging over a selection of collections, an exercise requiring significant effort and with unclear outcomes. It does appear, however, that in some applications it is the availability of storage that is helping drive the volume of data collected. From another perspective, the volume of data stored cannot exceed storage device capacity; in most applications there is little incentive to store data unless it will be readily available.

In this paper we make the approximation that collection size grows with disk capacity, that is, we take disk capac-

ity as an indicator of typical collection size. While this assumption may only hold for only some collections, there are certainly collections for which this approximation is conservative, and we believe that it provides a reasonable basis for experimentation.

3 Text retrieval systems

Conventional database systems are used to resolve queries, expressed in a formal language such as SQL, against a repository of structured records. An answer might be a single record, a subset of records or an aggregate across a subset, or a restructuring or analysis of the whole repository. Some costs, such as key-based access, are approximately constant; others are linear in collection size; others, in particular some kinds of join, are quadratic.

Information or text retrieval systems are used to resolve informal queries, typically a list of words, by locating the documents in a text repository that have the highest estimated statistical likelihood of being perceived as relevant to the query [11, 14]. An answer to such a ranked query is always a list of documents. Typically the list is of fixed length, that is, is independent of collection size. In such information retrieval systems, query evaluation proceeds as follows [2]. First each of the query terms is found in a lexicon, requiring approximately one disk access per term. (Even if the vocabulary is organised as a B-tree, the high branching factor and system disk caching ensure that all but the leaves are permanently held in memory.)

For each term, it is then necessary to fetch its inverted list, an array containing the identifiers of the documents in which the term occurs and, optionally, information such as the frequency with which the term occurs in each document. List length is roughly linear in collection size. For efficiency each list should be stored contiguously on disk, to allow fetching with a single disk access. Underlying file structuring with blocks may cause some lists to be split across cylinders, increasing access costs, but given that track capacity is currently of the order of a megabyte—thousands of times greater than a typical list in an efficient representation—this is not likely to be a significant factor in practice. Thus the cost of fetching a list is a sublinear factor, seek time (which increases slowly as collection size grows), plus a linear factor, transfer time. In modern machines efficiency can be improved by storing lists compressed, which reduces transfer time and average seek time (more lists fit into each cylinder) and effectively increases the capacity of system caches, but increases CPU costs; the net effect on current machines is roughly neutral, so that storage capacity is increased without impact on overall response time.

The information in the lists is then used to identify the most likely documents, an operation that requires memory and time linear in collection size. The top documents are then fetched, an operation that increases slowly with collection size due to increases in seek costs. The documents can also be stored compressed, which in this case—because of the high efficiency of text compression algorithms—leads to a net saving in response time.

Thus for ranked queries, the number of seeks is about constant. As collection size grows longer inverted lists must be fetched and decoded, a cost that should be roughly balanced by increasing throughput and processing power. For another mode of retrieval, Boolean querying, the trend costs are as for ranking, except that the number of answers is proportional to collection size, necessitating more seeks. From abstract considerations alone, then, the impact on response time of faster processor and bus and of larger disks and collections is not at all obvious. We explore these relationships in the experiments described below.

4 Experimental design

To measure performance on hardware over the last decade we have developed a new testing framework for retrieval systems and chosen a series of Sun machines as hardware platforms. Each machine was worth within a few thousand dollars of $30,000 at time of purchase. The machines, which we call Alpha, Beta, Gamma, Delta, and Epsilon, were purchased in 1990, 1993, 1995, 1997, and (a better configured machine) 1999 respectively. Their clock speeds and model names are shown in Table 1. Other differences between the machines include, not just bus speed, but bus design, which has been considerably improved between Alpha and Epsilon. Alpha runs the SunOS operating system; the others run recent versions of Solaris. The capacities of individual disks that came with these machines are shown in Table 1. The disk drives are all typical of their era, and vary in seek and latency times only a little; the newer disks are less than two times faster than the old. Experiments on Epsilon used a striped RAID array.

The data we use is drawn from the TREC text retrieval experiment sponsored by NIST [6]. This data is distributed as a series of CD-ROMs, and consists of text documents such as newspaper articles, journal articles, and patent applications. We have used disks 2 and 4. For each machine we have extracted from the TREC data a collection that is exactly one-tenth the size of the typical disk, that is, 55.2 Mb, 176.3 Mb, 375.4 Mb, 437.3 Mb, and 1,757.8 Mb respectively. The queries are numbers 251 to 300. Each original TREC query consists of a title of a few words, a single-sentence description that expands on the title, and a detailed narrative. From the queries we automatically constructed three sets of ranked queries, that is, three sets of lists of words: *title* queries that are similar to the kinds of queries presented to Internet search engines, of around 3 words each; *descriptive* queries, of around 12 words each;

Table 1. *Characteristics of machines used in experiments.*

	Alpha	Beta	Gamma	Delta	Epsilon
Year of purchase	1990	1993	1995	1997	1999
Clock speed (MHz)	40	50	150	168	336
Model	SUNW 4/75	SuperSPARC	HyperSPARC	UltraSPARC	UltraSPARC II
Per-disk capacity (Mb)	552	1763	3754	4373	17,578

and *full* queries, of around 40 words each.

The retrieval system used is the publicly-available MG prototype text database system, which incorporates compression for indexes and text [1, 9, 14], developed by staff at RMIT and the University of Melbourne. MG is designed to test algorithms for efficient text retrieval in different environments. Before running each query set, all system caches were flushed, so that the times reported below include retrieval from disk. MG release 1.2[1] compiles on all the machines without modification, and thus provides an excellent benchmark for cross-system comparison.

As another cross-system benchmark, we used an in-memory retrieval task: search for strings in a large hash table. The hash table was first populated with strings, then the strings were reordered and searched for in the table in turn. Only the searching component of the task was timed. Two collections of strings were used: a lexicon of 104,406 distinct words (drawn from 925 Kb of text) and a lexicon of 764,439 distinct words (drawn from 7234 Kb of text). This task shows the relative efficiency of the systems when memory accesses and processor cycles are the main costs, that is, disk is not involved. By comparison with the results with MG, the relative cost of having to use disk is illustrated. We used another experiment to measure disk throughput, by fetching a 176 Mb file after flushing system caches, a task whose cost will typically be dominated by bus and disk reading (but not seeking) parameters.

In all experiments and machines there was ample memory for the software to run without paging. Memory sizes are increasing, and more memory means more caching of inverted lists and answer documents; while larger collections mean that there is more data to be cached. Experiments that factor the effect of memory, to be realistic, must consider numbers of users and rates at which similar queries or queries with similar answers re-enter the system—parameters that are guesses at best. However, given that disk volumes are growing at least as rapidly as memory volumes, we believe that omitting memory as a factor does not unnecessarily penalise the newer machines. We therefore do not consider memory in our experiments.

[1] MG is available at `http://www.cs.mu.oz.au/mg/`

5 Results

We first used the in-memory lexicon searching task on collections of constant size to compare the systems. Results are shown in Table 2. Epsilon is about 14 times faster than Alpha and 3 times faster than Gamma. The difference between Alpha and Beta, which are of similar CPU speed, is partly attributable to Beta's faster bus. Delta is about 7 times faster than Alpha, both in Table 2 and in Table 3 below. Given that Delta's data set is larger than Alpha's by a similar factor—almost exactly 8 times—these two machines provide an interesting contrast in the experiments described below. The disk-testing throughput task shows that throughput from disk increases only slightly more slowly. Standard SPEC benchmarks suggest that Epsilon is 23 to 40 times faster than Alpha,[2] a marked contrast to all results we report and strong evidence for the need for a framework such as ours for measuring retrieval experiments.

Somewhat different trends to our in-memory lexicon searching task can be observed for tasks that involve disk seeks. Table 3 shows elapsed and CPU time when each machine is used to resolve ranked queries against the Alpha data set and fetch the 50 most highly ranked documents. In CPU time, Epsilon is for this task around 9 times faster than Alpha and not quite twice as fast as Gamma. These results show that the CPU is increasingly idle, with disk costs dominating elapsed time for the newer machines; this idle time offers an increasingly wide window for the use of CPU for compression and caching to improve response time, as we have explored in other work [13].

Similar results are observed in Table 4, but the contrast with Table 2 is even more marked: there is almost no improvement in elapsed time between Gamma and Epsilon, which is only 5 times faster than Alpha. This result was perhaps the most surprising one observed: *even on the same data set* speed improvements over the last few years have been small. However, Epsilon would show somewhat better relative performance on a larger data set, in which seek

[2] We did not determine SPEC benchmarks ourselves, but took them from a variety of websites including

```
http://performance.netlib.org/performance/
       html/spec.html
http://www.spec.org/osg/
http://www.netspace.net.au/∼kmk/spec.html
```

Table 2. *Total elapsed time for the "in-memory lexicon searching task", and the "throughput task", to fetch a large file (seconds).*

	Alpha	Beta	Gamma	Delta	Epsilon
Search small lexicon	2.10	0.73	0.41	0.31	0.15
Search large lexicon	18.97	7.61	4.13	2.45	1.40
Fetch file	126.5	67.9	35.3	19.7	12.9

costs would be less significant. (The number of seeks for the results in Tables 3 and 4 is shown in the first column of Tables 5 and 7.)

Table 5 shows the costs of a typical use of ranking, where a query is resolved against a collection and the top 50 answers are returned. As collection size increases the number of seeks grows very slightly and is not a significant factor in relative costs. The two major differences across the collection sizes are that the larger collections have longer inverted lists to fetch and decode, and, less significantly, more memory is required to represent document scores during query evaluation. The other aspects of the task are constant. The net effect is more or less neutral. Alpha is slower than the other machines, but otherwise there is no strong trend except for a slight decline in CPU time as a proportion of total cost. For this task, collection growth and hardware improvements are in balance, but note that the number of seeks has not grown significantly.

Results for a similar task are shown in Table 6, where only a single document is fetched for each query. This eliminates part of the processing that is constant in costs, so that costs that scale with collection size are more dominant, as indeed they will be with each new generation of machines. In this application, Epsilon is no faster than Beta, Gamma, and Delta, although it is a more highly configured machine. In CPU time, Epsilon is slower than all of its predecessors.

(These results incidentally confirm the design decision in MG to store inverted lists contiguously. Had lists been broken into small blocks, as is sometimes recommended in the file systems literature, the fetch cost—which is significant even with contiguous lists—would have been much greater.)

Conjunctive Boolean querying presents a quite different test environment, because most of the costs scale with collection size. Results for Boolean querying are shown in Table 7. (No results are given for the full queries because few documents are answers to conjunctive queries of 30 terms.) Alpha and Beta are the fastest machines, and Epsilon is the slowest. Delta has less than three times Beta's data, has three times the clock speed, but responds more slowly. In the presence of scaling data, machines are becoming slower.

For the descriptive queries, there are often no answers in the smaller collections, allowing query processing to terminate early, in some cases before all query terms are considered. Part (a) of Table 7 includes these queries, for many of which the evaluation time is negligible, in particular on the smaller collections; on the other hand, even when, in the larger collections, these queries do have answers, they are not numerous. Part (b) shows results for only the queries with answers in the Alpha collection, which is a subset of the others. Considering clock speeds and data volumes alone, one would expect Epsilon to be two or three times slower in response time than Alpha. However, it is up to 19 times slower. Interestingly, the bottleneck for this task is increasing CPU costs—due to the need to decompress lists and answers—which is the converse of what we had expected to observe. Epsilon is remarkably slow, requiring up to 60 times as much CPU time as Alpha.

The Boolean retrieval task is similar to common non-key queries to conventional database systems, and therefore such systems too could be expected to become slower as data volume grows, even with newer hardware.

Some of the results tabulated above are plotted together in Figure 1. Together they show that the trend costs can vary considerably depending on the precise mix of use of disk and CPU; for example, on the Alpha collection, ranked querying is consistently faster from machine to machine, whereas Boolean querying is not. Several years of hardware development have had little effect on Boolean response time. Ranking is slightly faster with each new generation of machine and data set, while Boolean querying is much slower. The graph also shows how well systems cope with scaling data. The Epsilon collection is around 32 times as large as the Alpha collection. For the ranked task, response time increases by a factor of only 2.5 when resolving descriptive queries against the larger collection; for the Boolean task, the factor is over 80.

Table 3. *Average per-query elapsed and CPU time for the "static collection ranking task", to resolve ranked queries with 50 answers against the Alpha 55.2 Mb data collection (seconds).*

		Alpha	Beta	Gamma	Delta	Epsilon
Title	CPU	1.25	0.64	0.25	0.20	0.15
queries	Elapsed	3.91	2.44	1.52	1.00	0.84
Descriptive	CPU	1.40	0.70	0.27	0.21	0.16
queries	Elapsed	4.45	2.41	1.68	1.16	0.94
Full	CPU	1.82	0.89	0.34	0.27	0.21
queries	Elapsed	5.81	3.51	2.28	1.46	1.16

Table 4. *Average per-query elapsed and CPU time for the "static collection Boolean task", to resolve Boolean queries against the Alpha 55.2 Mb data collection (seconds).*

		Alpha	Beta	Gamma	Delta	Epsilon
Title	CPU	1.93	1.06	0.38	0.34	0.21
queries	Elapsed	4.23	2.66	1.47	0.99	0.71
Descriptive	CPU	0.25	0.11	0.06	0.04	0.03
queries	Elapsed	1.11	0.53	0.28	0.29	0.25

6 Conclusions

We have explored how the performance of a text retrieval system varies with changing hardware and data volume. Even with constant-sized data, new hardware does not improve on old as much as hardware specifications would suggest, with for example a 1999 Sun machine no faster than a 1995 machine for Boolean retrieval. With growing data volume, changes in performance are unpredictable, with small improvements in some cases and massive declines in others.

In contrast to conclusions that could be drawn from reports of hardware improvements—for example, that compression algorithms are of declining interest because of the abundance of disk—it is clear that in the context of retrieval systems the study of algorithms and efficiency remains important with increasing machine capacity. For retrieval systems, the number of disk seeks required to resolve a query is clearly a crucial bottleneck. Doubling clock speed and doubling data size is not cost neutral, unless the number of seeks is held constant. The cost of accessing disk must be a focus in development of new query evaluation algorithms. Indeed, comparing machines by processor time—even machines of similar architecture—is not helpful, despite being common practice (see for example Post [10]).

In addition to growth in data volumes, the numbers of users are increasing, as more households and businesses conduct business and recreation online. Internet search engines must not only index more pages, but must service more users. We have not explored the impact of growth in user numbers, but it is a further factor to consider in the light of our results. It is not surprising that the collections indexed by the Internet search engines are increasingly out-of-date, or that drastic heuristics must be employed to ensure rapid response to queries. We can only conclude that improvements in hardware will not meet the demands being made of retrieval systems, and that the emphasis on making improvements to processors may be misplaced.

Acknowledgements

We thank Michael Fuller and Neil Sharman. This work was supported by the Australian Research Council.

References

[1] T. Bell, A. Moffat, I. Witten, and J. Zobel. The MG retrieval system: Compressing for space and speed. *Communications of the ACM*, 38(4):41–42, Apr. 1995.

[2] E. Bertino, B. Ooi, R. Sacks-Davis, K.-L. Tan, J. Zobel, B. Shidlovsky, and B. Catania. *Indexing Techniques for Advanced Database Systems*. Kluwer Academic Press, Boston, Massachusetts, 1997.

[3] P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson. RAID: High-performance, reliable secondary storage. *Computing Surveys*, 26(2):145–185, 1994.

[4] F. Collins, A. Patrinos, E. Jordan, A. Chakravarti, R. Gesteland, and L. Walters. New goals for the U.S. human genome project: 1998–2003. *Science*, 282(5389):682–689, 1998.

[5] M. Flynn. Computer engineering 30 years after the IBM model 91. *IEEE Computer*, 31(4):27–31, 1998.

[6] D. Harman. Overview of the second text retrieval conference (TREC-2). *Information Processing & Management*, 31(3):271–289, 1995.

Table 5. *Average per-query CPU time (seconds), elapsed time (seconds), and number of seeks for the "fifty-answer dynamic collection ranking task", to resolve ranked queries with 50 answers against data collections proportional to machine disk size.*

		Alpha	Beta	Gamma	Delta	Epsilon
Title	CPU	1.25	0.78	0.38	0.31	0.42
queries	Elapsed	3.91	2.89	2.20	1.48	1.86
	Seeks	103.6	109.2	113.0	112.8	124.8
Descriptive	CPU	1.40	0.92	0.49	0.42	0.61
queries	Elapsed	4.45	3.29	2.58	1.75	2.20
	Seeks	114.6	120.6	123.9	124.6	135.8
Full	CPU	1.82	1.30	0.83	0.72	1.20
queries	Elapsed	5.81	4.50	3.60	2.52	3.42
	Seeks	157.9	164.8	167.4	168.6	180.0

Table 6. *Average per-query CPU time (seconds), elapsed time (seconds), and number of seeks for the "one-answer dynamic collection ranking task", to resolve ranked queries with one answer against data collections proportional to machine disk size.*

		Alpha	Beta	Gamma	Delta	Epsilon
Title	CPU	0.13	0.16	0.13	0.11	0.20
queries	Elapsed	0.53	0.44	0.18	0.25	0.38
	Seeks	8.5	8.6	8.6	8.7	9.6
Descriptive	CPU	0.24	0.29	0.25	0.21	0.40
queries	Elapsed	1.06	0.85	0.60	0.51	0.77
	Seeks	19.6	19.8	19.9	19.8	20.7
Full	CPU	0.59	0.67	0.56	0.71	0.97
queries	Elapsed	2.39	2.05	1.74	1.29	1.94
	Seeks	62.8	62.9	63.0	63.0	64.0

[7] D. Harman and G. Candela. Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *Journal of the American Society for Information Science*, 41(8):581–589, 1990.

[8] S. Lawrence and C. Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998.

[9] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, Oct. 1996.

[10] G. Post. How often should a firm buy new PCs? *Communications of the ACM*, 42(5):17–21, 1999.

[11] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Reading, MA, 1989.

[12] R. Wilkinson, T. Arnold-Moore, M. Fuller, R. Sacks-Davis, J. Thom, and J. Zobel. *Document Computing: Technologies for Managing Electronic Document Collections*. Kluwer Academic Press, Boston, Massachusetts, 1998.

[13] H. Williams and J. Zobel. Compressing integers for fast file access. *Computer Journal*, 42(3):193–201, 1999.

[14] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, California, second edition, 1999.

Table 7. *Average per-query number of answers, CPU time (seconds), elapsed time (seconds), and number of seeks for the "dynamic collection Boolean task", to resolve Boolean queries against data collections proportional to machine disk size.*

		Alpha	Beta	Gamma	Delta	Epsilon
Title queries	Answers	47.8	155.0	424.4	512.1	2009.8
	CPU	1.25	2.23	2.67	3.10	14.85
	Elapsed	2.85	5.63	8.40	7.46	22.93
	Seeks	59.0	178.9	469.1	563.2	2158.7
Descriptive queries	Answers	0.3	1.0	8.3	11.9	66.0
	CPU	0.08	0.08	0.15	0.18	3.73
	Elapsed	0.60	0.48	0.58	0.56	4.93
	Seeks	13.7	15.3	25.0	29.4	101.3

(a) Using all 50 queries

Title queries	Answers	74.7	242.0	662.3	799.3	3135.4
	CPU	1.93	3.46	4.51	4.82	22.69
	Elapsed	4.23	8.63	12.97	11.53	35.07
	Seeks	88.4	275.1	727.3	874.1	3359.3
Descriptive queries	Answers	4.0	11.3	94.8	135.8	640.3
	CPU	0.25	0.33	1.10	1.44	15.18
	Elapsed	1.11	1.13	3.50	3.46	21.53
	Seeks	19.3	31.0	127.3	173.3	766.8

(b) Using only those queries that have answers in the Alpha collection

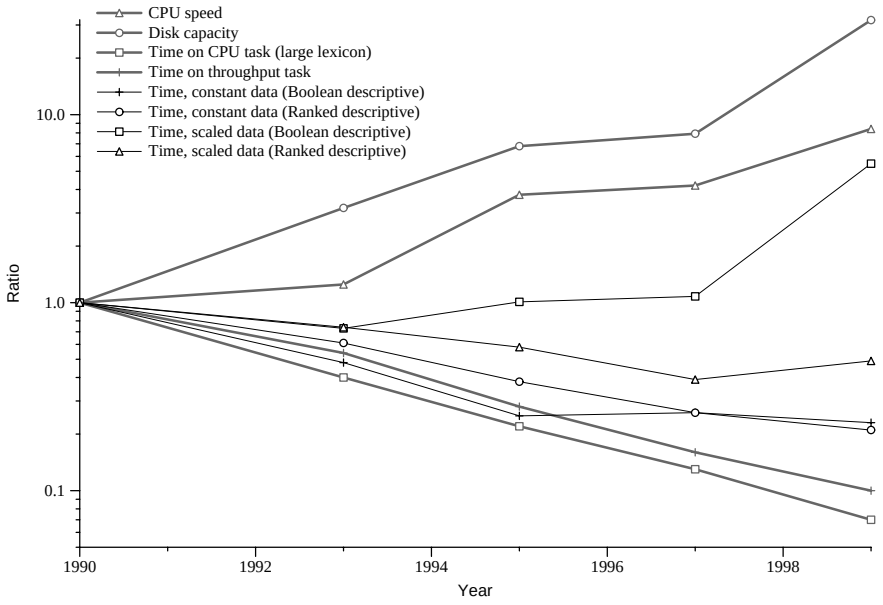

Figure 1. *Relative data size, CPU speed, and elapsed times across machines Alpha (1990) to Epsilon. All numbers are relative to Alpha.*