[Stone, 1987] Stone, H. S. (1987). Parallel querying of large databases: A case study. *IEEE Computer*, 20(10):11–21.

[Tomasic and Garcia-Molina, 1993a] Tomasic, A. and Garcia-Molina, H. (1993a). Caching and database scaling in distributed shared-nothing information retrieval systems. In Buneman, P. and Jajodia, S., editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 129–138, New York. ACM Press. Conference held in Washington, D.C. Also published as SIGMOD Record Volume 22, Issue 2, June 1993.

[Tomasic and Garcia-Molina, 1993b] Tomasic, A. and Garcia-Molina, H. (1993b). Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In Carey, M. J. and Valduriez, P., editors, *Proceedings of the Second International Conference On Parallel and Distributed Information Systems*, pages 8–17, Los Alamitos, CA. IEEE Computer Society Press. Conference took place in San Diego.

[Turtle and Croft, 1992] Turtle, H. R. and Croft, W. B. (1992). Uncertainty in information retrieval systems. In Motro, A. and Smets, P., editors, *Proceedings of the Workshop on Uncertainty Management in Information Systems*, pages 111–137. Workshop held in Mallorca, Spain.

of about 3.8 MB (800,000 postings uncompressed), on the order of 1.0% of the index, can improve throughput by about 171% for the prefetch strategy. For the other strategies, improvements are smaller. Although not reported here, we also experimented with various cache policies. For example, in one case, lists above a given threshold were not cached, even if they fit in the cache, on the presumption that they would flush out too many useful lists. However, we observed no significant improvement with this caching variation.

In our study we assume that a single combined index was constructed for all fields (see Section 3). Although we have not yet experimented with partitioned indexes, our current results indicate that performance will *not* improve by partitioning. Partitioned indexes force *subject* field searches to read more, but shorter lists, which would only exacerbate the disk seek problem. Some systems actually have both combined *and* partitioned indexes. This does not seem attractive either. For some queries, lists read will be shorter, but the same number of disk seeks need to be performed. As long as seeks remain the critical resource, the gains should not be significant. (This assumes that the system is dedicated to query processing. If the hosts are used for other tasks, then reducing the size of lists to intersect may still be desirable.) This conjecture needs to be verified.

Acknowledgments: Thanks to Howard Marantz and Norman Roth who gathered the raw trace and posting counts from FOLIO. Luis Gravano, Ben Kao and Masahiko Saito provided several useful suggestions.

References

[Burkowski, 1990] Burkowski, F. J. (1990). Retrieval performance of a distributed text database utilizing a parallel processor document server. In Agrawal, R. and Bell, D., editors, *Proceedings of the Second International Symposium on Databases in Parallel and Distributed Systems*, pages 71–79, Los Alamitos, CA. IEEE Computer Society Press. Conference held in Dublin, Ireland.

[Chervenak, 1990] Chervenak, A. L. (1990). Performance measurements of the first RAID prototype. Technical Report UCB/UCD 90/574, University of California, Berkeley, CA.

[Cringean et al., 1990] Cringean, J. K., England, R., Mason, G. A., and Willett, P. (1990). Parallel text searching in serial files using a processor farm. In Vidick, J., editor, *Proceedings of the 13th International Conference on Research and Development in Information Retrieval*, pages 429–453, New York, NY. ACM Press. Conference held in Brussels, Belgium.

[DeFazio and Hull, 1991] DeFazio, S. and Hull, J. (1991). Toward servicing textual database transactions on symmetric shared memory multiprocessors. In *Proceedings of the International Workshop on High Performance Transaction Systems*, Asilomar.

[Emrath, 1983] Emrath, P. A. (1983). *Page Indexing for Textual Information Retrieval Systems.* PhD thesis, University of Illinois at Urbana-Champaign.

[Frakes and Baeza-Yates, 1992] Frakes, W. B. and Baeza-Yates, R. (1992). *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall, Englewood Cliffs, NJ.

[Gray and Reuter, 1993] Gray, J. and Reuter, A. (1993). *Transaction processing: concepts and techniques.* Morgan Kaufmann, San Mateo, CA.

[Jain, 1991] Jain, R. (1991). *The Art of Computer Systems Performance Analysis.* John Wiley and Sons, New York, NY.

[Jeong and Omiecinski, 1992] Jeong, B.-S. and Omiecinski, E. (1992). Inverted file partitioning schemes for a shared-everything multiprocessor. Technical Report GIT-CC-92/39, Georgia Institute of Technology, College of Computing, Atlanta, GA.

[Salton, 1989] Salton, G. (1989). *Automatic Text Processing.* Addison-Wesley, New York, NY.

[Stanfill, 1990] Stanfill, C. (1990). Partitioned posting files: A parallel inverted file structure for information retrieval. In Vidick, J. L., editor, *Proceedings of the 13th International Conference on Research and Development in Information Retrieval*, pages 413–428, New York, NY. ACM Press. Conference held in Brussels, Belgium.

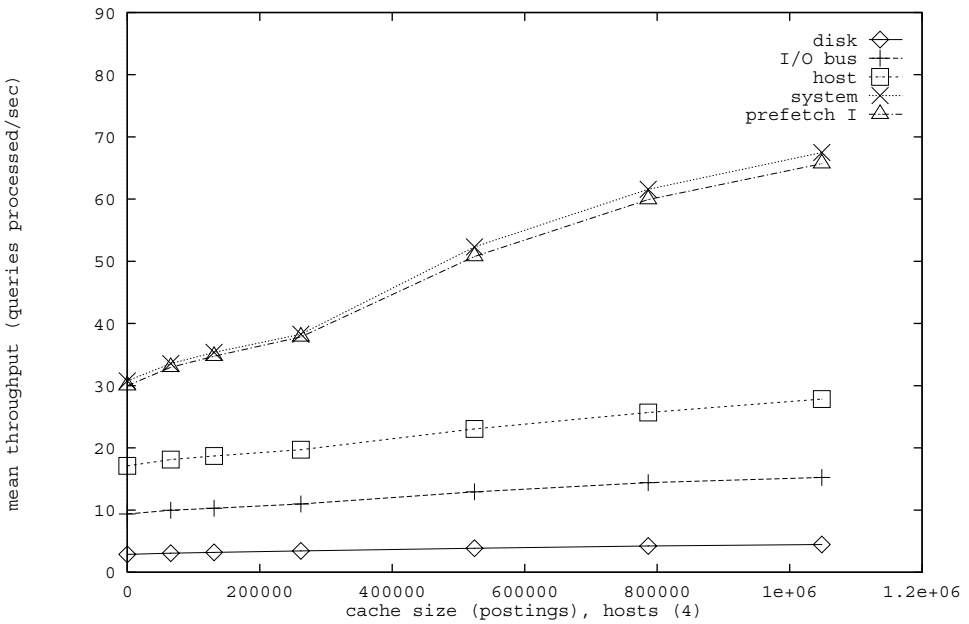

Figure 15: The impact of the cache size on throughput.

on throughput for the disk organization because it remains bottle necked on the disks.

6 Conclusion

We have studied an information retrieval system for bibliographic entries or abstracts. Using queries from the INSPEC database on the FOLIO system at Stanford University, we analyzed strategies for distributing indexes across a set of processors and for performing queries in parallel.

Our main result is that inverted lists referenced by queries in such systems tend to be relatively short and it does *not* pay off to split them across hosts, much less across I/O subsystems or disks. Either system index organization, or the system index organization with the prefetch I optimization, performs best over wide ranges of parameter values. Prefetch I is especially good as the database size scales up. However, the system organization does utilize the LAN or processor interconnect more heavily, so it would not be appropriate for systems with slow networks.

Our conclusion is different from that of our earlier work [Tomasic and Garcia-Molina, 1993b] where a full-text information retrieval system was analyzed. In that case, inverted lists are much longer, and breaking them up (e.g., striping them) does pay off. In particular, the host organization was superior in that scenario.

In our experiments, we explored the "mainframes vs. workstations" issue. That is, we took a specific index distributed over a fixed number of disks and I/O buses. Then we considered whether it would be best to connect all these resources to a single fast processor, or to connect them to n processors each of $(1/n)^{th}$ the speed. The mainframe does achieve moderately higher throughput, but the gains have to be evaluated in light of the higher mainframe cost. In other words, one has to take the throughput rates we report here, and divide them by the dollar cost of each configuration, to obtain a query/sec/dollar measure, as is done in transaction processing systems [Gray and Reuter, 1993].

We also explored the impact of truncation queries by running the same experiments with a trace with the truncation queries removed. We found only a slight performance improvement and therefore have not included results on these experiments. Our conclusion is that two factors contributed to this result. First, truncation keywords are a small fraction of all the keywords which appear. Second, the data structure used to model truncation queries is very efficient, since the performance impact is restricted to simply reading longer inverted lists (about twice as long on average).

Our caching results indicate that a relatively small cache can improve performance significantly. For our INSPEC database that has an index size of 377MB (129 million postings compressed) a cache

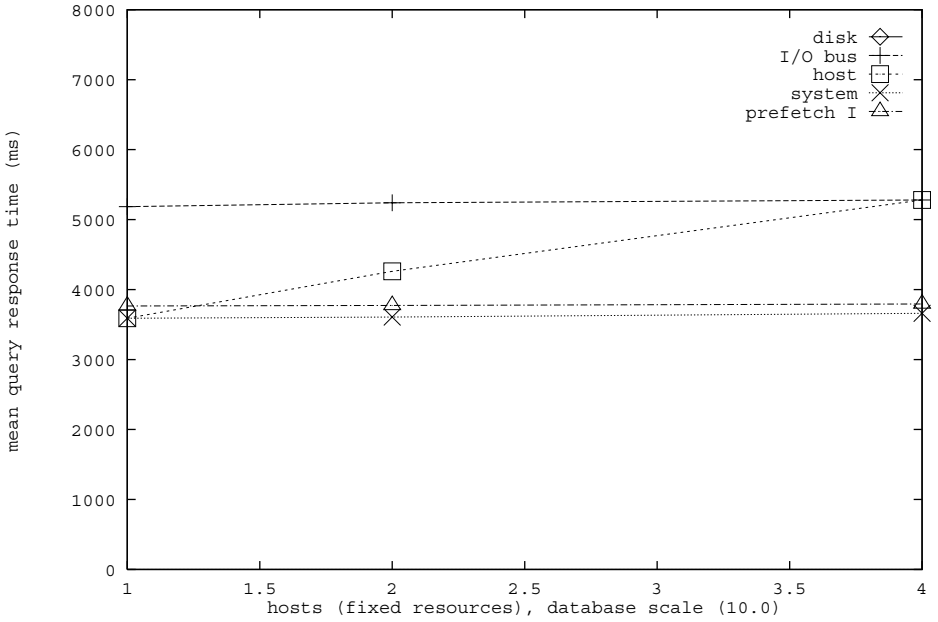

Figure 13: The mainframe vs. workstation trade-off.

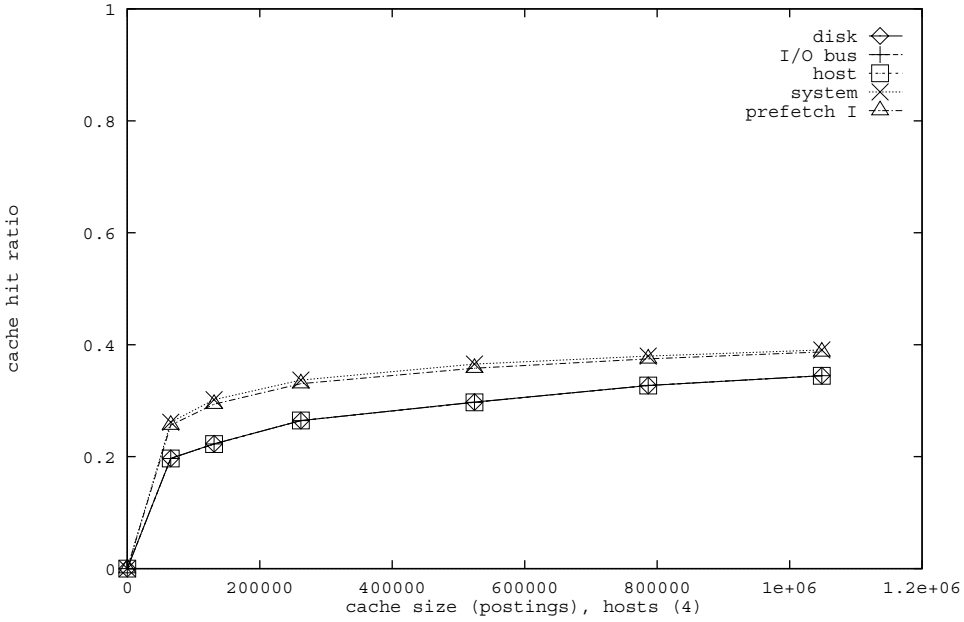

Figure 14: The improvement in the cache hit rate as the cache grows in size.

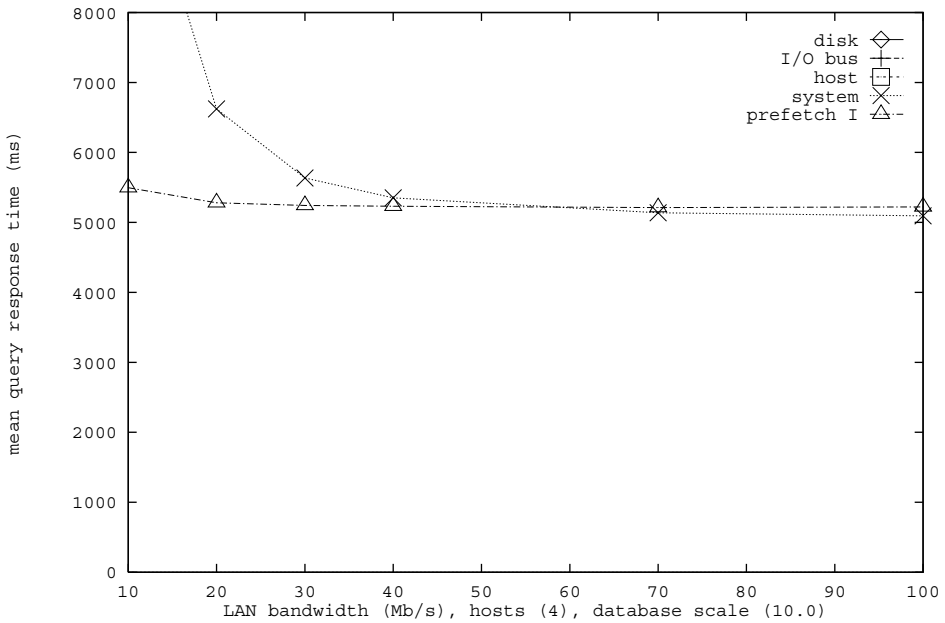

Figure 12: The sensitivity of mean query response time to LAN bandwidth.

Hosts	I/OBusesPerHost	Multiprogramming	CPU Speed	Database Scale
1	4	4	20	10.0
2	2	2	10	10.0
4	1	1	5	10.0

Table 10: Enumeration of variable values for fixed resources.

difference between a single host and multiple hosts is that the fast CPU has been replaced by several slower CPUs interconnected by a LAN. Figure 13 shows that the best index organization (system) has a response time of 3.59 sec., 3.61 sec., and 3.66 sec. for 1, 2 and 4 hosts respectively, indicating about a tenth of a second loss in response time when split among multiple hosts. Throughput for the system index organization is 1.12 queries/sec., 1.11 queries/sec., and 1.10 queries/sec. for 1, 2 and 4 hosts respectively. Thus a minimal performance loss is incurred by using the multiple host organization. The results indicates that a "mainframe" is slightly more effective, but the small improvement has to be evaluated in light of the potentially higher mainframe cost (compared to workstations and a LAN).

Finally, we turn to the issue of caching and address two simple questions. How rapidly does the cache hit rate rise as the size of the cache increases? What is the effect of the rising cache hit rate on performance? Figure 14 shows the increase in the cache hit ratio as the size of the cache increases for a four host system (database scale 1.0). Since the total cache size is the same regardless of the index organization, it is surprising that the cache hit rates vary depending on the organization. However, the behavior of the caches under the various organizations is quite different. For the system and prefetch I index organizations an inverted list is cached in only one place in the system. Thus, there are effectively *Hosts* number of independent caches. Also, suppose a list slightly larger in size than the cache is read from disk. In the system and prefetch I organization, the list does not fit in the cache and thus the caches would remain unchanged. In the disk, I/O bus, and host organizations, however, all four caches would hold a list of quarter the size, requiring some other lists to be removed from the cache. The figure shows that for the base configuration even a small cache has a good hit rate – achieving almost 40% where the maximum possible cache hit rate is about 65.7% (see Table 3). The cache hit rates for the disk, I/O bus, and host index organizations are the same since they access exactly the same lists on each host and so the cache contents are changed in the same way over the course of the simulation.

Figure 15 shows that the effect of caching is to free the I/O subsystem resources to speed up query processing for the system and prefetch I index organizations. The cache does not have a dramatic effect

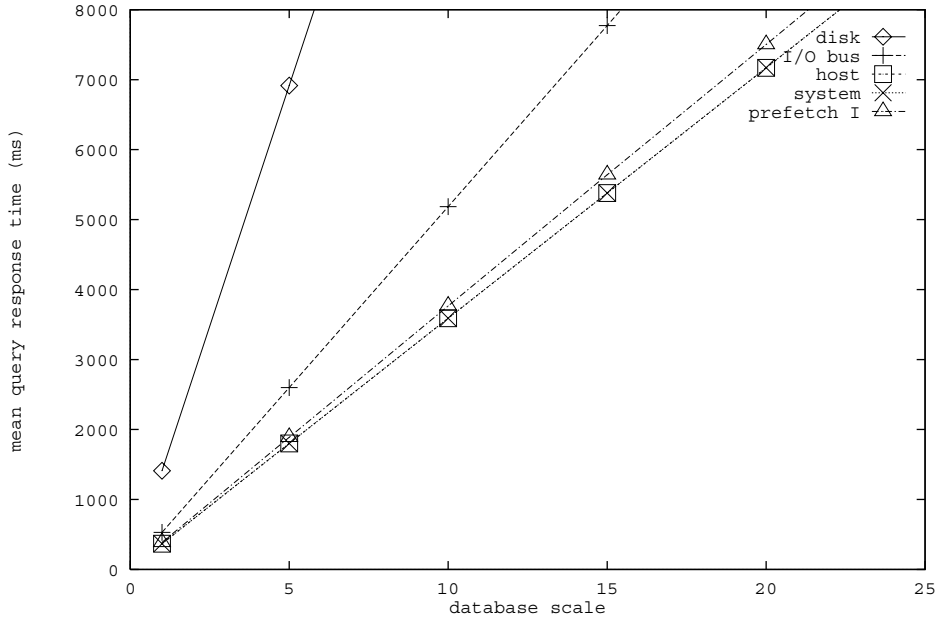

Figure 10: Scaling the database up to a 4 second response time for the best index organization.

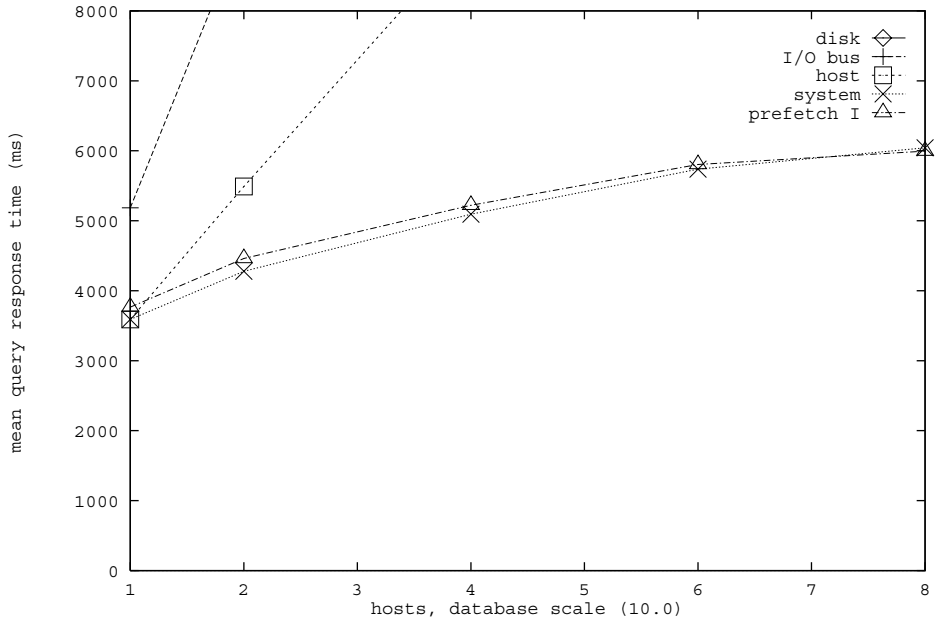

Figure 11: Increasing the number of hosts with a scaled database.

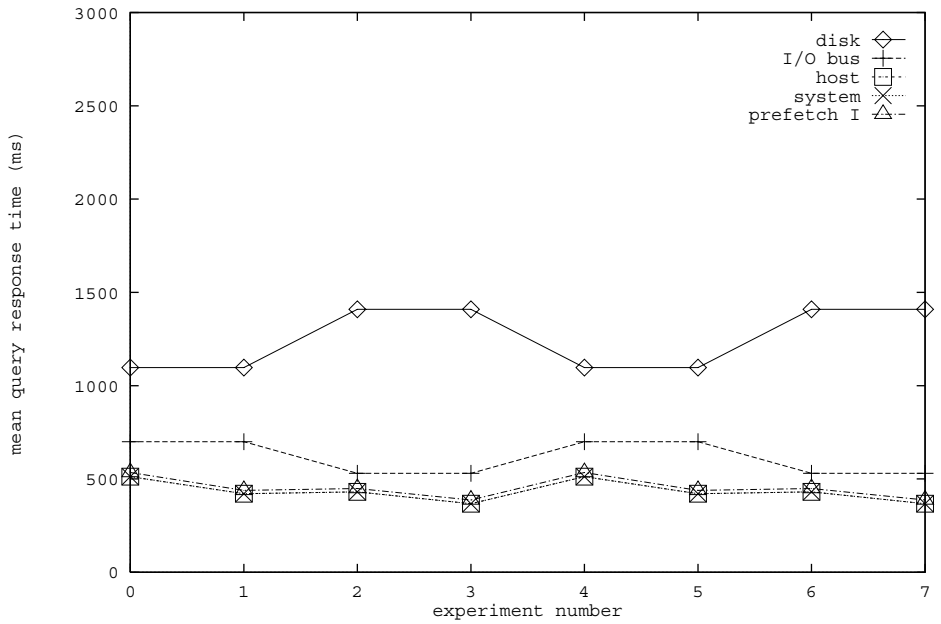

Figure 9: A 2^k factor experiment of three variables.

of 512, $DisksPerI/OBus$ of 4 and $I/OBusesPerHost$ of 4 as the value for these variables in the base configuration. (Note that this the worst configuration for the disk index organization.)

To study database scaling, we first maximize the size of the database which can be effectively processed with the base configuration. We choose a 4 second mean response time as the limit for an effective information retrieval system. We scale the database on the base configuration (as described in Section 3) until the best response time increases to the threshold of 4 seconds. This graph is shown in Figure 10. From this graph we choose the value of 10.0 for the maximum scaling of the database for a single host.

We now wish to observe the effectiveness of the system as the number of hosts is increased. Increasing the number of hosts also increases the total number of I/O buses, disks, and queries (since the number of queries in the entire system is determined by $Multiprogram \cdot Hosts$). In Figure 11 the increase in response time is shown as the number of hosts is expanded.[3] The increase in response time is due to two factors. First, the total load of the system is increasing in proportion to the number of hosts. Second, as the number of hosts increases the traffic across the LAN increases. We see this effect appear at 8 hosts where the prefetch I index organization slightly outperforms the system index organization. Thus, the prefetch I organization scales well as the number of hosts increases.

However, the performance of the system organization depends strongly on the speed of the LAN. Figure 12 shows the impact of this variable on mean query response time. The left hand side of the graph represents a Ethernet type network at maximum bandwidth and the right hand side of the graph represents an FDDI network. The graph shows that the system response time is highly sensitive to the LAN bandwidth and that a sufficiently fast network eliminates the disadvantages of the system organization. The prefetch I organization performs more poorly at a high bandwidth with four hosts due to the sequential nature of the two phased approach. Note that the prefetch I organization is relatively flat in this graph. Thus, even with a very fast network, this organization would be preferable if the network cannot be utilized to its maximum capacity. In addition, any of a number of other variables can make prefetching attractive e.g., the number of hosts, or a larger database.

Given that an index organization does well as the number of hosts increases, we can compare the base configuration of a single host to configurations with more hosts but the same total resources in terms of CPU speed, number of disks and I/O buses. This is essentially the trade-off between buying a single large mainframe processor or several slower workstation size processors. Table 10 shows a enumeration of configurations which explore this trade-off under a fixed total system load. The main

[3] In subsequent graphs, we attempt to keep the axis scales fixed so that various graphs can be easily compared.

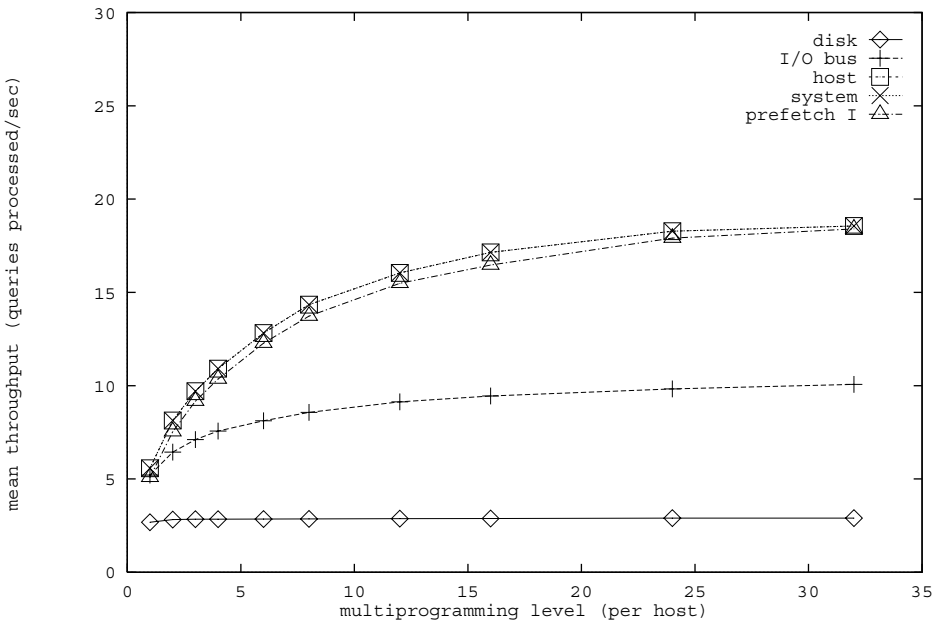

Figure 8: The effect of the multiprogramming level on throughput.

Experiment	BlockSize	DisksPerI/OBus	I/OBusesPerHost
0	512	2	2
1	512	2	4
2	512	4	2
3	512	4	4
4	16K	2	2
5	16K	2	4
6	16K	4	2
7	16K	4	4

Table 9: Enumeration of variable values.

to determine the configuration with the best response time. Table 9 lists the enumeration of the values of the variables.

The result of this experiment is graphed in Figure 9. The data points for each index organization have been connected by lines to aid the reader in understanding the graph. (Note that the a line connecting two points may represent the changing of the values of several variables.) We see that the left-hand half of the graph (values 0-3) is the same shape as the right-hand half (values 5-7). This means that $BlockSize$ has little effect on the response time, since it is the only variable to change value when comparing the halves of the graph. Next, examining each *sequential pair* of values (0,1), (2,3) etc. shows no change in the response time for the disk and I/O bus index organizations. For each pair the only variable to change is $I/OBusesPerHost$ which changes from 2 to 4. Thus, adding I/O buses (and implicitly, disks) does not improve the performance of these two index organizations. However, the response time for host, system, and prefetch organizations improve when more I/O buses are added because the total resources of the system are increased. Finally, consider the transition from the configuration in Experiment 1 to the configuration in Experiment 2. Here, the total number of disks is fixed at 8 but the arrangement of the disks changes because the number of I/O buses goes from 4 to 2. We see that disk index organization response time increases due to contention for the I/O bus (disk transfers on the same I/O bus are processed serially by the I/O bus). I/O bus index organization response time decreases because there are fewer inverted lists to read per keyword (since there are fewer I/O buses).

From this graph we pick the best combination of these variables for response time, namely $BlockSize$

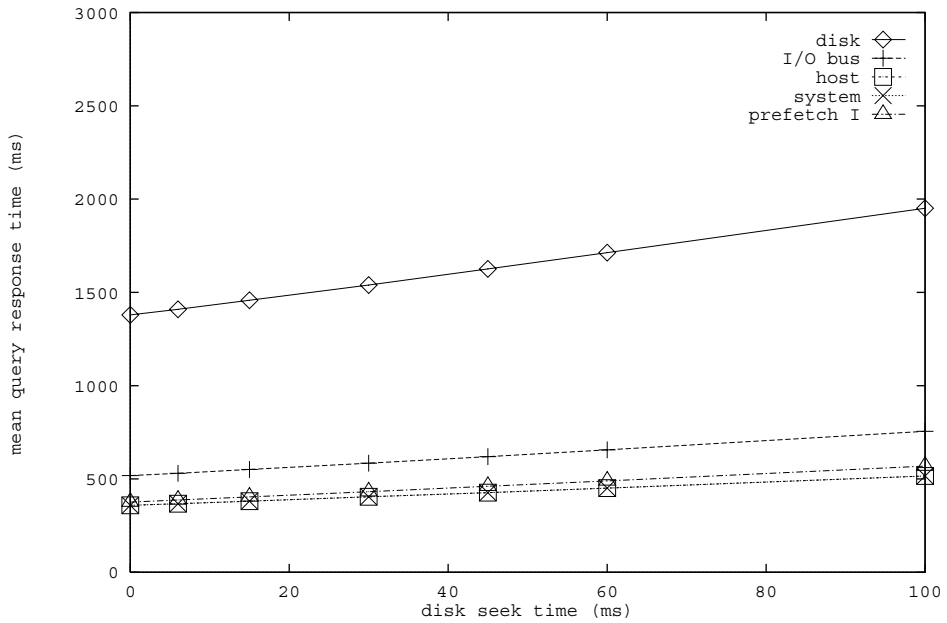

Figure 7: The sensitivity of response time to disk seek time.

the right could compare to optical disks. The graph shows that the disk index organization is most sensitive (i.e., has the largest slope) to the change in seek time, followed by the I/O bus, host, system and prefetch index organizations, respectively. This ordering of the index organizations is in decreasing number of inverted lists reads done by each organization. For a given query, the disk index organization does the largest number of reads, followed by the I/O bus index organization, etc. The increase in the number of reads leads to a higher disk utilization and increased queuing delays at every disk. We conclude that the seek time of the disk dominates the cost of accessing an inverted list as opposed to the bandwidth limitations of the I/O subsystem. The host and system index organizations perform identically in the base configuration because there is only 1 host in the base configuration. The prefetch I index organization performs slightly worse than the host and system index organization because the prefetch of an inverted list is performed sequentially with respect to the processing of the remainder of the query. This slightly decreases the amount of parallelism in the processing of the query. (Recall that prefetch I index organizations was designed to reduce LAN traffic, which is not an issue in a one host configuration.)

Figure 8 shows the effect of the rise in the multiprogramming level on the mean throughput of queries processed. The graphs shows that the disk and I/O bus index organizations are relatively insensitive to the change in the multiprogramming level. Other collected data shows that these two organizations are bottle necked in the I/O subsystem. As the multiprogramming level rises, the same number of queries can be processed per second, but each query takes longer and longer. The host, system and prefetch I index organizations continue to improve across the range of the multiprogramming level in the graph because the resources are more evenly balanced. For a multiprogramming level of 32, the response times for the disk, I/O bus, host, system and prefetch I index organizations are 11.01 sec., 3.17 sec., 1.73 sec., 1.73 sec. and 1.74 sec., respectively. Thus good response times are still available on a heavily loaded system.

Intuitively, experiments which vary the value of one variable in a configuration examine the change in a function along a single dimension. In some cases it is necessary to change the value of multiple variables in a systematic fashion in a 2^k *factor experiment* [Jain, 1991]. For three variables, this can intuitively be viewed as examining the values of a function at the corners of a three-dimensional cube (each axis of the cube corresponds to a variable).

To determine a reasonable base configuration, some of the values of the variables are provided by existing hardware, but other variables such as $DisksPerI/OBus$ are less easily determined. We conducted a 2^k factor experiment for $k = 3$ on the variables $BlockSize$, $I/OBusesPerHost$ and $DisksPerI/OBus$

Parameter	Value	Description
$CPUSpeed$	20	Relative speed in MIPS
$Multiprogram$	4	Multiprogramming *per Host*
$QueryInstr$	500000	Query start up CPU cost
$SubqueryInstr$	100000	Subquery start up CPU cost
$SubqueryLength$	1024	Base size of subquery message
$FetchInstr$	10000	Disk fetch start up CPU cost
$InterInstr$	40	Intersection CPU cost per byte of a decompressed inverted list
$Decompress$	40	Decompression CPU cost per byte of inverted list on disk

Table 7: Base case parameter values and definitions.

Parameter	Value	Description
$EntrySize$	40	Bits to represent an inverted list entry on disk (uncompressed)
$Compress$	0.5	Compression Ratio
$CacheSize$	0.0	Inverted list cache (in postings)
$ConcatInstr$	5	Concatenation CPU cost per byte of an answer set
$AnswerEntry$	4	Bytes to represent an entry in an answer set
$Documents$	1357034	Number of documents
$DatabaseScale$	1.0	Database scale factor

Table 8: Base case parameter values and definitions.

bytes ($postings \cdot EntrySize$). Then the number of instructions for this part of the subquery processing is 4,500 ($37.5 \cdot Decompress + 75 \cdot InterInstr$).

The size of the inverted list cache in postings is determined by the variable $CacheSize$ which is measured in number of postings. The policy for the cache is least-recently-used. When an inverted list read is a cache miss, it is read from disk and the number of postings in the list is checked to determine if it will fit in the cache. If the inverted list is smaller or equal in size to cache, the cache removes (in a least recently used fashion) enough inverted lists to make room for the new list. The new list is then inserted in the cache. If the list is larger than the cache, the list is not placed in cache and no other lists are flushed. Both of these cases are cache misses. When an inverted list is a cache hit, it is moved to the end of the list of the least recently used inverted lists. (Note that a possible improvement would be to also cache the intermediate and final results from the intersection computations.) For truncation keywords, a cache hit occurs only if exactly the same truncation keyword is used.

The number of bytes needed to represent a document in an answer is given in $AnswerEntry$. The instructions needed to concatenate the answers from the subqueries is given by $ConcatInstr$. The number of documents in the database is given by the variable $Documents$ and is equal to the number of abstracts in the INSPEC database. Finally, the variable $DatabaseScale$ permits scaling of the database as described in Section 3.

5 Results

In this section we present selected results of a set of experiments performed by the simulation. Space limitations prevent us from showing all the results. In conducting these experiments sensitivity analysis of all the variables in Section 4 were performed. An *experiment* is the execution of the simulation for the entire trace with a given configuration.

Figure 7 shows the mean response time of queries under the various index organizations as disk seek time increases (the other simulation parameters for this configuration are given in Tables 5–8). The seek values on the left of the graph (around 10 ms) represent a typical magnetic disk. Values on

Parameter	Value	Description
$Hosts$	1	Hosts
$I/OBusesPerHost$	4	Controllers and I/O Buses per Host
$DisksPerI/OBus$	4	Disks for each I/O bus

Table 5: Hardware configuration parameter variables, values and definitions.

Parameter	Value	Description
$DiskBandwidth$	10.4	Mbits/sec bandwidth per disk
$DiskBuff$	32768	Size of a disk buffer in bytes
$BlockSize$	512	Bytes per disk block
$SeekTime$	15.0	Disk seek time in ms
$TrackToTrack$	4.0	Cost to seek one track in ms
$I/OBusOverhead$	0.0	I/O bus transfer in ms
$I/OBusBandwidth$	24.0	Mbits/sec bandwidth I/O bus
$LANOverhead$	0.1	LAN transfer in ms
$LANBandwidth$	100.0	Mbits/sec bandwidth LAN

Table 6: Hardware parameter values and definitions.

Typically an experiment systematically varies one or more of the values to determine the effect of the variables. A *configuration* is the total collection of variable-value pairs used in an experiment. The *base configuration* is the collection of variable-value pairs given in the tables in this section.

Table 6 shows the base configuration variables for the hardware. The values for this table were taken from [Chervenak, 1990]. The disks and I/O buses are simulated as follows. Requests for a disk read arrive from the CPU (after determining that they are cache misses). Each read has a specified length in bytes. The reads are queued at the disk in a first-come-first-served (FCFS) manner. Each request is first serviced by the disk by waiting an initial $SeekTime$ milliseconds. The disk loads its track buffer at $DiskBandwidth$ speed. When it finishes, the disk requests access to its I/O bus (only one disk at a time may occupy the I/O bus). When the I/O bus grants access both the I/O bus and disk are occupied for the transfer at $I/OBusBandwidth$ speed. If multiple tracks must be loaded then the initial seek time is extended by the needed track-to-track seeks (variable $TrackToTrack$).

The LAN handles the transmission of subquery and answer messages. Messages are serviced in a FCFS manner (except for messages that have the same source and destination - these are immediately returned to the host, simulating software loop-back). Each subquery has a length determined by $SubqueryLength$ and each answer has a length determined by $AnswerEntry$ times the number of postings in the answer. The service time for each message is $LANOverhead$ plus the time taken to transmit the message at the given $LANBandwidth$.

Table 7 shows the parameters which affect the CPU and the time taken to process a query. The overall speed of a CPU is determined by the parameter $CPUSpeed$. Varying this value proportionately varies the rate at which instructions are executed. The number of instructions needed to execute various stages of the matching process are listed in the table. Note that the multiprogramming level of the system is on a *per host* basis.

Finally, Table 8 lists the variables used to determine the size of the inverted lists. The variable $EntrySize$ determines the number of bits needed to record a posting in an inverted list. The variable $Compress$ determines the reduction in bytes in the inverted list due to compression.

To illustrate the use of the variables in Tables 7 and 8, consider a subquery which intersects two inverted lists with 5 and 10 postings, respectively. The initial subquery CPU processing would be 120,000 instructions ($SubqueryInstr + 2 \cdot FetchInstr$) since each inverted list read is charged a start-up cost of $FetchInstr$ instructions. The length of one list is 100 bits ($postings \cdot EntrySize \cdot Compress$). The length of the other list is 200 bits. The disk read length for both lists is 512 bytes (rounded up due to $BlockSize$). After the disk data is fetched, only the bits in the actual lists are used for subsequent computations. The number of instructions to process the intersection combines the costs of decompression and intersecting the lists. The size of the uncompressed inverted lists is 600 bits or 75

for A and B are fetched from disk; however, only the postings of the appropriate type (title for A, author for B) are used. The number of A postings with *title* designation is given in the trace, call it $n(A)$; the number of B *author* postings is $n(B)$. The expected size of the intersection of these lists is thus $n(A)n(B)/D$, where D is the total number of documents in the database. This assumes that each word is equally likely to appear in any of the D documents and that the words occur independently. (If an additional C word were in the query, the expected size would be $n(A)n(B)n(C)/D^2$.) (Emrath [Emrath, 1983] reports some measurements which support this model.)

For the disk, I/O bus and host index organizations, we believe the model is very accurate. For the system index organizations, query terms are clearly not independent. However, there are several mitigating factors in comparing our model with any particular actual subquery answer. For single word subqueries, our model exactly matches an actual subquery. For two word subqueries, there are three cases. In the case that both subqueries are frequent words, the result is near D in both the model and the actual result. In the case that both subqueries are infrequent words, the result is small in both the model and the actual result, and thus difference between the model and the actual result has a small impact on our results. In the case that one word is frequent and the other infrequent, our model underestimates the result size if the two query words are highly correlated. To compensate for this effect, we use the result size (for the overall query) in the trace as a lower bound to the expected number of documents in a subquery. This adjustment is reasonable since the size of the final answer to the match is bounded from above by the minimum of the sizes of the subquery answers in the system organization. For multiword subqueries, as the number of keywords in the query increases, the expected number of answers approaches zero. Our adjustment also compensates for this bias in our model.

Finally, we note one case in which we take license with the data in the previous section. Since the date and time of each query issued is reported in the trace, it is possible to simulate the exact sequence of query arrivals in the system in an open system type of queuing model. We have chosen a closed queuing system model that permits studying the effects of varying the multiprogramming level (the number of simultaneous queries in the system). This is accomplished by starting a number of queries at the beginning of the trace equal to the multiprogramming level and then starting the next query in the trace whenever a query finishes in the system. This method maintains a constant number of queries in the system.

The effect of this is to concurrently simulate queries that are *from the same user* and thus could not have been requested simultaneously. This situation introduces a race condition for two queries which access the same inverted list. If the queries are executed sequentially, the second query will always be a cache hit. But if the queries are executed concurrently, both queries may simultaneously check the cache and both may miss. We believe that the negative impact of this loss in realism is minor and is outweighed by the advantages of studying the effect of the multiprogramming level on response time and throughput (cf. Section 5). Our cache hit results are slightly pessimistic due to the race condition.

To study the effects of scaling the database, the parameter *DatabaseScale* was added to the simulation. This variable linearly scales the number of postings for each inverted list, the number of answers to a query and the number of documents in the database. While this model of scaling is primitive, since it does not account for the appearance of new words as documents are added, it does conservatively model the impact of adding documents. A word of caution is in order with respect to the scaling the number of answers. If a query matches 10 documents, a user will simply read all 10 documents to determine which ones are of interest. Faced with a query with a result size of 1000, a user would probably issue a modified query to produce a smaller answer. We believe that as the database grows in size (say, linearly) the mean result size grows more slowly as users continue to construct queries with manageable result sizes. However, we do not incorporate this into the database scaling model.

4 Simulation

In this section the hardware simulation is described, together with the parameters that specify the resources consumed by each stage of query execution. An example hardware organization is shown in Figure 2. Every hardware organization consists of a local area network (LAN) connecting several hosts together. Each host has a CPU and memory, a number of I/O buses, and a number of disks. Every host has the same number of I/O buses and every I/O bus has the same number of disks. Each host also has a cache. Table 5 lists the variables that determine the hardware organization. The "Value" column in the table shows the "base case" value of each variable used in the experiments described in Section 5.

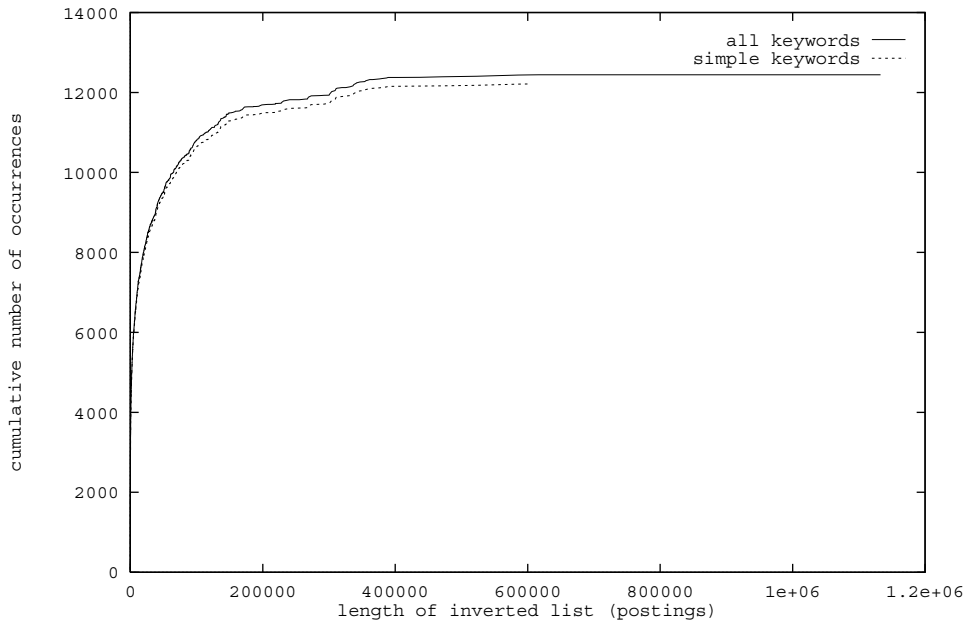

Figure 6: The cumulative distribution of occurrences of inverted indexes of a given length which appear in queries. Simple keywords do not use truncations.

"prefetch I" that operates as follows: we divide the processing for a query into two phases. In the first phase, the home site sends a subquery to the host holding the shortest inverted list for the query. This host broadcasts the shortest list on the LAN to all other hosts. In the second phase, the remaining inverted lists are retrieved (and intersected if more than one list resides at the same host), except that before results are sent to the home host, they are intersected with the first list broadcast. This significantly reduces the data volume on the LAN by reducing the mean subquery answer size. (In [Tomasic and Garcia-Molina, 1993b] two other prefetch variations are studied. For our current study, we evaluated all three variations; Prefetch I was the variation with the best performance (as in [Tomasic and Garcia-Molina, 1993b]), so to economize on space, we only describe the winning variation and its performance.)

To simulate the processing of a query, we consider five stages. The first stage covers the initial CPU processing for parsing the query and generating subqueries. Second, the subqueries are queued at the LAN for transmission to other hosts. Third, the process blocks, waiting for the subqueries to complete. When all the answers are returned the process wakes and simulates another CPU processing stage for the intersecting of the inverted lists. Finally, the process terminates, indicating that the matching for the query is complete. (If the prefetch algorithm is used, several additional stages are added to account for the two phases.)

A subquery goes through five stages also. First, initial start-up CPU processing is simulated. Second, the cache is checked for the words which appear in the query. For cache misses, reads are issued to the disks for the inverted lists. The process blocks, waiting for the disk reads to be returned through the I/O bus subsystem. When all the reads have returned, the subquery process wakes and simulates the intersecting of the inverted lists into an answer by a CPU processing stage. The answer is then queued at the LAN and the subquery terminates.

From our trace data, we can determine how many inverted lists have to be fetched to answer a given query, and how large the lists are. However, our simulation also requires the sizes of the intermediate results, and we estimate them by calculating the expect number of answers as follows. In the case of the disk, I/O bus and host index organizations, we make the assumption that the answers are distributed in equal proportion across all hosts. Thus, to compute the size of the subquery answer we simply divide the result size reported in the trace by the number of hosts. For the system organization, however, each subquery generally contains a subset of the keywords in the query. The following example illustrates how the expected answer size is calculated. Say the subquery is *find title A author B*. The full lists

Description	Words	Postings	Mean	Median
Abstract	487247	74477422	152.9	1
Author	311632	4110892	13.2	2
Classification	2962	4211136	1421.7	634
Conference	11934	7246145	607.2	41
Free Term	319831	28110201	87.8	1
ISBN et. al.	12637	2445828	193.5	41
Author Org.	59546	8228475	138.2	2
Document Org.	2505	1145724	457.4	61
Report	7508	7833	1.0	1
Thesaurus	3695	11382655	3080.6	708
Publication	18411	6794557	369.0	4
Title	171537	10292321	60.0	1
Total	1409445	158453189	112.4	n/a

Table 4: The inverted indexes and associated statistics. The mean and median columns apply to the number of postings per word.

The distribution of the lengths of the inverted lists which appear in the trace characterizes the work required to process the queries. Figure 6 shows the cumulative distribution of list lengths. The steep slope on the left-hand side of the figure shows that almost all of the inverted lists are less than 100,000 postings in length. The flat slope on the right-hand side of the figure shows that there are a few long inverted lists. These observations are confirmed by the following statistics: There are 12,503 inverted lists referenced in the trace. The mean length of an inverted list is a little more than 43,000 postings. The median inverted list has 6139 postings. From the figure we see that the mean length is much larger because of a few very long inverted lists.

3 Query Processing

Information retrieval systems may partition their indexes by field designator or may build a combined index. In the partitioned case, all occurrences of a word in a *title* field are listed in one index, all *author* occurrences in another, and so on. In the combined case, a single index is built, and for each entry in an inverted list, a type annotation indicates the field where the word was used. The main advantage of the combined index is that a query such as *find subject art* can be answered by fetching the single inverted list for *art*, as opposed to fetching the lists for *art* in the *title*, *author*, etc. indexes. (Recall that a *subject* search is shorthand for a combined search, see Section 2.) On the other hand, a query *find title art* can be processed faster with partitioned indexes, since only the relevant postings need to be processed. Since 37.2% of our queries involve *subject* searches, here we assume a single combined index. (In Section 6 we return to this issue.)

The use of a combined index allows a very simple model of truncation queries. We assume that the posting lists are allocated sequentially and in alphabetical order. Thus, the matching inverted lists can be read with a single disks access and the number of postings to read is the sum of the postings of the individual matching words. For the keyword *yak#*, the inverted lists for keywords *yak, yakube*, etc. cost only a single disk access. (Note that FOLIO does not allow truncations of the form #yak.) The assumption of this data structure reflects a realizable data structure and is the "best case" for truncation processing with inverted lists. Furthermore, we assume the system organization can be tuned so that words with the same alphabetical prefix reside on the same host. Our simulation results demonstrate that (with a proper data structure) truncation processing can be done with a negligible impact on performance.

We model continuation queries through a dummy index called *user*. To illustrate, suppose user 5 issues the query *find subject A* and its returns 100 results. Then the continuation query *and author B* would be simulated as the query *find user 5 and author B* where *user 5* is an inverted list with 100 postings.

As discussed in the introduction, four physical index organizations are considered. We found in previous work [Tomasic and Garcia-Molina, 1993b] that the LAN may be the bottleneck for the system index organization. To ameliorate this problem we adopt one query processing optimization named

Description	Value	
Total Keywords	12444	
Number of *Subject* Field Keywords	4630	
Percent *Subject* Field of All Fields	37.2	
Number of Truncation Keywords	230	
Unique Keywords	4263	
Maximum Cache Hit Percent	65.7	
Description	**Mean**	**Median**
Keywords per Query	1.79	2
Result Size per Query	736.1	22
Matches per Keyword	25213.1	2344
Matches per Truncation Keyword	33894.9	6387
Postings per Keyword	43248.5	6139
Postings per Truncation Keyword	77897.5	35176
Words Matched per Truncation Keyword	73.3	23

Table 3: Statistical properties of the simulation trace.

```
Count:          5  Key: CITRENBAUM
Count:         43  Key: CITRIN
Count:          5  Key: CITRINI
Count:          1  Key: CITRINOVITCH
Count:          4  Key: CITROEN
```

Figure 4: The number of postings for a sample set of words which appear in the *author* portion of the documents.

documents. The number of bytes per document is roughly 1,800. The total database size can be (very roughly) estimated at 2.3 Gigabytes.

For the matching of queries, the total of lengths of the inverted lists (i.e. total number of postings) read is important in determining the amount of work done in the matching process. As mentioned earlier, we scanned the actual INSPEC inverted lists recording the number of postings and their field designations. For example, Figure 4 shows a sample of the file containing the number of postings for words with the *abstract* field designation. Table 4 lists for each field designation some statistics on the postings.

To drive our simulation, we combine the information from the trace and postings files into a single trace file that is easy to use. Figure 5 shows a sample of this final trace file. For example, the first line of the example shows query number 8, where user 68 issued a query which had 0 results. The query referred to the *author CITRIN* and the *subject BROMINE*. The value 239427 is a hashed (but unique) value of the keyword *CITRIN* and is used to determine the disk or disks which the inverted list or lists will reside for the various index organizations. The next number, 43, is the number of posting for *CITRIN* that have the author field designation. The final number, 47, is the total number of posting entries for *CITRIN*, i.e., the total number of documents where *CITRIN* appears, regardless of the field designation. The number of postings for a subject field designation is the total of the postings for the constituent parts. Note that this number is typically much higher than the number of documents that match, due to the duplication of matches in the various field designations. For this sample trace file, two cache hits would occur (one for *author CITRIN* and one for *subject EXAFS*) assuming that the cache is initially empty.

```
8 68 0 ( CITRIN author 239427 43 47 ) ( BROMINE subject 236928 1137 1138 )
9 68 6 ( CITRIN author 239427 43 47 ) ( EXAFS subject 248828 4225 4226 )
10 68 21 ( EXAFS subject 248828 4225 4226 ) ( CHLORINE subject 241651 3188 3194 )
```

Figure 5: Queries 8 through 10 of the trace input to the simulation.

```
68 04/25/93 10:45:02 CMD: fin a citrin and s exafs
68 04/25/93 10:45:04 SEA: CPU:262, Res:6, Find AUTHOR citrin and SUBJECT exafs
68 04/25/93 10:45:04 CMD: !DISPLAY 1
```

Figure 3: Some example data from the raw trace. The first column is a unique integer representing the user login.

Total Raw Trace Queries	8390	100.0 %
Discarded Queries	429	5.1 %
Nonexistent Terms	1001	11.9 %
Simulation Experiment Queries	6960	83.0 %

Table 2: Breakdown of raw trace and simulation trace.

in the first line. (The trace repeats the query here.) The third line reports that the result of the query (short descriptions of each document in this case) where shown to the user.

To drive the simulation, only a subset of the raw trace is considered. Queries that have terms with no associated inverted list (12.0%) (e.g. misspellings) are ignored since the FOLIO query parser catches these queries and rejects them. Thus, they have no impact on performance beyond a small CPU overhead for parsing. To be more precise, we assume a negligible performance cost for detecting query terms with no associated inverted list. Queries consisting of boolean AND operations on terms or truncation matching of terms are simulated. We also simulate queries which are continuations of previous queries or are subject queries (discussed later in this section). We do not consider 5.1% of the queries which are errors in the log, phrase queries, boolean OR queries, boolean NOT queries, or queries on chemical compounds. We believe that these queries have very little impact on the performance results presented here. The raw trace contained 8390 query commands. The remaining queries used to drive the simulation constitute 83.0% of the original queries (or 94.2% of the queries not caught as a zero result by the parser).

One important feature of FOLIO is the designation of the *subject* field in query matching. This field designation is a syntactic shorthand for matching the union of the field designations *abstract*, *conference*, *freeterm*, *document organization*, *thesaurus*, and *title* simultaneously. Thus, the query *find subject theory* is conceptually a shorthand for the query *find abstract theory or conference theory or freeterm theory or document organization theory or thesaurus theory or title theory*. The subject field designation constitutes 37.2% of all field designations. Subject queries are handled by our simulation (see Section 3).

Two other features features of FOLIO handled by our simulation are *truncation matching* and *continuation queries*. A truncation match is a keyword containing a "#" that matches zero or more characters. Thus, the keyword *yak#* matches *yak, yakube*, etc. The addition of truncations can introduce performance problems and we discuss a specific data structure to handle them in Section 3. A continuation query adds extra conditions to the current query by using the command *and* instead of the command *find*. For example, the query *find subject exafs* produces 1595 answers. This query can be refined with the continuation query *and author citrin* which produces the same 6 results as the query *find author citrin and subject exafs*. The simulation of this feature is discussed in Section 3.

Some statistics of the traces will be helpful in interpreting the results of the simulation. Table 3 summarizes some properties of the query traces. To our surprise, the mean number of keywords per query is less than two and the median is two. However, note that each use of the subject field designation is a shorthand for a query with multiple keywords. The mean size of the result of a query is large (over seven hundred) but the median result size is small at 22. We suspect that the queries with large results are immediately refined to produce smaller results.

Issuing multiple queries to refine an answer set is common in information retrieval systems. This query refinement behavior by a user provides an opportunity for the caching of inverted lists. Of all the keywords appearing in the traces, 65.7% of them are duplicate appearances. Thus, if we cached every single read of an inverted list in the system we would achieve a maximum cache hit ratio of 65.7% over the entire trace. (While this figure is not a high as those reported in the file system literature, Section 5 shows that caching does have a significant impact on mean throughput.)

For the database, total number of documents for the INSPEC database is reported at 1,357,034

4. What is the impact of caching inverted lists in main memory? Is there enough locality of reference between queries to make caching worthwhile?

5. What is a good data structure for truncation matching? What is the impact of truncation matching on performance?

Our evaluation is based on query traces from the FOLIO library information retrieval system at Stanford University, run against a detailed event-driven simulation of the hardware and query processing. The trace data is described in Section 2, while our processing model is described in Section 3. The simulation model is presented in Section 4. Our results are given in Section 5 and conclusions in Section 6.

1.1 Previous Work

A substantial amount of work has been done in the general area of Information Retrieval. For a good explanation of the current state of the art in terms of improved precision and recall, see [Salton, 1989, Turtle and Croft, 1992].

Not that much has published on distributing inverted lists and searching them in parallel. The four index organization we have described are from [Tomasic and Garcia-Molina, 1993b]. Reference [Stone, 1987] discusses some of the basic issues. Burkowski simulates a shared-nothing information retrieval system [Burkowski, 1990] to study the performance impact of the placement of documents and inverted indexes. Jeong and Omiecinski [Jeong and Omiecinski, 1992] independently study for a shared-everything architecture similar issues of the physical index design as in this article. Work has also been done on searching with a variety of other architectures, e.g., processor farms [Cringean et al., 1990], fine-grained parallelism [Stanfill, 1990], and shared-memory multiprocessors [DeFazio and Hull, 1991].

Here we study an abstracts database as opposed to a full-text system [Tomasic and Garcia-Molina, 1993b]. This article extends the preliminary results reported in [Tomasic and Garcia-Molina, 1993a]. In a full-text system, every single word occurrence is indexed. In an abstracts system, only the abstract is indexed. If we compare two systems with the same *number* of documents, the index in the full-text case will be much larger. Even if the volume of raw data is equal (for example, abstracts that are a tenth of the size of the full-text documents but there are 10 times as many abstracts), the inverted lists for the abstracts case will still be smaller. This is because repeated words are indexed in the full-text case only. For instance, if a word appears 10 times in a document, there will be 10 index entries (pointing to each occurrence) in the full-text case, and only one entry in the abstracts case. As we will see, the fact that inverted lists are shorter for abstracts dramatically changes the relative performance of the various organizations. (Emrath [Emrath, 1983] focuses on the performance trade-offs involved in partial or complete indexing.)

2 Data

Stanford University provides on-campus access to its information retrieval system FOLIO from terminals in libraries and from workstations via telnet sessions. FOLIO gives access to several databases; one of these is INSPEC, an abstracts database for technical documents in disciplines such as physics, electrical engineering, and computer science. A trace of all user commands for the INSPEC database were collected from 4/12/93 to 4/25/93. In addition, the number of postings of every word in the INSPEC database inverted index was also collected.

Each INSPEC abstract is divided into fields such as *title* and *author*. One of these fields is called *abstract*. To avoid confusion between the field and the complete record name, we will henceforth refer to the complete abstract as the *document*. In a query, a user specifies the field where each word should appear. This is called the *field designation*.

An example of the raw trace is shown in Figure 3. The first column is the user identification[2], the second and third columns are the date and time of the command, and the fourth column specifies the type of data. The first line of the figure shows that user 68 issued a query for *author Citrin* and the *subject exafs*. The user types *fin* as shorthand for *find*, *a* as a shorthand for the *author* field designation and *s* for *subject*. The second line of the raw trace shows that six documents are the result for the query

[2] To insure the privacy of the data, the user login identification has been replaced by a unique integer.

Index	Disk	Inverted Lists in *word: (Id, Field)* form
Disk	d 0	a: (0, A), (0, B); theory: (0, T);
		system: (0, A), (0, T), (0, B)
	d 1	a: (1, B); theory: (1, A), (1, T), (1, B)
	d 2	a: (2, B); hard: (3, T), (3, B)
	d 3	a: (3, B);
Host &	d 0	a: (0, A), (0, B), (1, B);
I/O bus		theory: (0, T), (1, A), (1, T), (1, B)
	d 1	system: (0, A), (0, T), (0, B)
	d 2	a: (2, B), (3, B)
	d 3	hard: (3, T), (3, B)
System	d 0	a: (0, A), (0, B), (1, B), (2, B), (3, B)
	d 1	system: (0, A), (0, T), (0, B)
	d 2	hard: (3, T), (3, B)
	d 3	theory: (0, T), (1, A), (1, T), (1, B)

Table 1: The four inverted index organizations for the words "a", "hard", "system", and "theory" in Figures 1 and 2. The field "A" is for author, "T' is for title, and "B" is for abstract. (Other lists are not shown.)

the hosts.) The subqueries are processed in the same manner as the other index organizations. When the answers to the subqueries are returned to the home host, another intersection is performed on the answers to produce the final answer.

To illustrate, consider the query *find abstract system and title theory* issued to the home host CPU 0. In the system index organization two subqueries are issued. One is the subquery *find abstract system* and is sent to the host CPU 0, the other is the subquery *find title theory* and is sent to host CPU 1. The subqueries are processed in parallel. CPU 0 generates the subquery answer (the list of matching abstracts) (0) and CPU 1 generates the answer (0, 1). Both answers are transmitted to the home host which constructs the intersection of the answers producing the answer (0) indicating that document 0 matched the query. In the host index organization two subqueries are also issued, each consisting of the query *find title theory and abstract system*. The subqueries are processed in parallel. Host CPU 0 transmits the answer (0) and host CPU 1 transmits the empty answer. The home host concatenates the two answers to produce final answer to the query.

Note that the abstracts themselves do not have to reside on the same disks as the inverted index. That is, for constructing the various index organizations, abstracts were *logically* assigned to disks or hosts, but they can be physically stored wherever we wish. Also note that the abstracts themselves are not accessed during query processing; to see an abstract the user must issue a separate display command. In this article we assume that abstracts are stored on separate disks (a common technique in practice) and we only study query processing, i.e., we ignore display commands.

In this article we address five basic types of questions:

1. What index organization yields best performance? In the system organization, typically a query only involves a subset of the hosts, leaving the irrelevant hosts free to process other queries. On the other hand, the other strategies allow more intra-query parallelism and may generate more uniform loads at the hosts. So which one leads to lower response times or higher throughputs? Also, there are various optimizations to the system organization, dealing with the order in which lists are fetched and intersected (see Section 3). How do these improve performance?

2. What are the critical hardware resources? What is the optimal arrangement for a given set of hardware resources? In particular, many current information retrieval systems run on large mainframes. Can we really improve performance by having instead a collection of less expensive machines, implementing distributed indexes?

3. How well do the algorithms and hardware scale as the database size grows? As mentioned earlier, current data collections are growing rapidly, and it is not clear what index organization scales best, or whether it is more important to add disk or processor or communication resources as the database grows.

id: 0 *author:* A. System *Title:* Theory of System *abstract:* A practical system, good for one year or one million dollars, whichever comes first.
id: 1 *author:* B. Theory *Title:* Theory of Theory *abstract:* A theory, might be useful, might not.
id: 2 *author:* C. Hardware *Title:* Hard Ware *abstract:* A very hard piece of ware.
id: 3 *author:* D. Student *Title:* Thesis *abstract:* A direct extension of my advisor's will.

Figure 1: A example set of four documents.

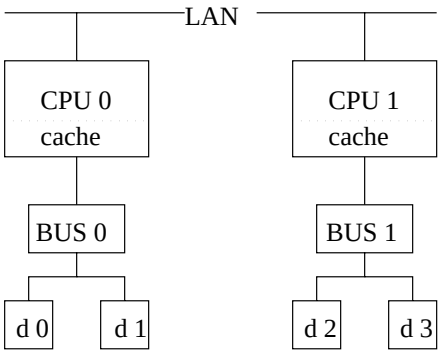

Figure 2: An example hardware configuration.

3. Host Organization. An inverted index is constructed for the abstracts assigned to the disks of each host. The inverted lists are spread across the disks of the host.

4. System Organization. In the previous organizations, for each word, there are multiple inverted lists, one at each disk, I/O bus, or host. In the system organization, a single inverted list is generated for each word. Each inverted list is allotted to one of the disks of the system.

To illustrate these organizations, consider the four documents in Figure 1. Each document contains four fields: author, title, abstract, and id. An example hardware organization is shown Figure 2; it has two hosts, labeled CPU 0 and CPU 1, each with a cache, one I/O bus and two disks. Table 1 shows the various inverted index organizations for the figures. (The Host and I/O Bus organization are equivalent in this example since each host has a single I/O bus.) Note that in this table, each entry in an inverted list is typed with the field name where the word appears ("A" for author, "T" for title, "B" for abstract).

For instance, the word *a* appears five times in the example set of documents. The word appears in each abstract and it appears in the author field for document 0. For the system organization, all the appearances of a word are in the same inverted list. In the table this inverted list in located on disk d 0. For a given keyword and field designation, all the corresponding entries in the inverted list are the *postings* for that keyword and field designation. Thus, this inverted index organization combines all the postings of a word for all the field designations into a single inverted list. In the other organizations, the *a* list is split. For example, in the Host organization, there is one *a* list covering abstracts in disks d0 and d1 (stored in d0 in this example), and another *a* list for the d2, d3 abstracts (stored in d2).

To answer a query originating on a host (the *home* host) in the disk, I/O bus, and host index organizations, a copy of the query is made for each host. This subquery is sent to each host which then *matches* the subquery against its inverted lists. Since queries consist of keyword-field pairs connected by boolean ANDs, matching is accomplished by constructing the intersection of the inverted lists. The result of the intersection, the *answer* to the subquery, is then transmitted to the home host. The home host concatenates all the subquery answers to produce the final answer.

To answer a query in the system index organization, a subquery is sent to each host relevant to the query. (A host is relevant to a query if the inverted list for at least one keyword in the query resides on

Performance Issues in Distributed Shared-Nothing Information Retrieval Systems *

Anthony Tomasic† and Hector Garcia-Molina
Stanford University Department of Computer Science

August 5, 1996

Abstract

Many information retrieval systems provides access to abstracts. For example Stanford University, through its FOLIO system, provides access to the INSPEC database of abstracts of the literature on physics, computer science, electrical engineering, etc. In this article this database is studied by using a trace-driven simulation. We focus on a physical index design which accommodates truncations, inverted index caching, and database scaling in a distributed shared-nothing system. All three issues are shown to have a strong effect on response time and throughput. Database scaling is explored in two ways. One way assumes an "optimal" configuration for a single host and then linearly scales the database by duplicating the host architecture as needed. The second way determines the optimal number of hosts given a fixed database size.

1 Introduction

Information retrieval (IR) systems, of the type found in libraries, provide indexed access to the abstracts of documents. Information vendors such as Dialog and BRS Search also provide access to such abstracts databases. The number of such databases is rapidly growing, as more and more information is stored digitally. At the same time, an increasing number of users have access to these databases through the networks. To handle the increased load, a distributed architecture can be used, dispersing the data and index structures across several computers and performing searches in parallel. This article studies the performance trade-offs in such a shared-nothing distributed information retrieval system. By shared-nothing, we mean that processors do not share memory or disks. Our work complements an earlier paper [Tomasic and Garcia-Molina, 1993b] where a full-text information retrieval system was studied (where an entire document is indexed, as opposed to just its abstract). Research in IR has given rise to many retrieval models, e.g. the boolean model, the vector space model, and probabilistic models. Since the user data used here is based on a boolean model of abstract-only databases, we consider only this class of systems in this article.

An abstracts database typically uses an *inverted index* to speed up query processing (see [Frakes and Baeza-Yates, 1992] for a survey of access methods for text).[1] For each word, an *inverted list* is constructed that gives all the abstracts in which the word appears. In a multiprocessor environment, the inverted lists can be distributed in various ways:

1. Disk Organization. The abstracts are logically partitioned by physical disk, that is, each disk is assigned a (hopefully equal) number of abstracts. An inverted index is then constructed for the abstracts of each disk.

2. I/O Bus Organization. Each I/O bus controls a subset of disks. An inverted index is constructed for all the abstracts of the disks on each I/O bus. Each inverted list is stored on a disk in the I/O bus group.

*This research was partially supported by the Defense Advanced Research Projects Agency of the Department of Defense under Contract No. DABT63-91-C-0025.

†Current address: INRIA Rocquencourt, 78153 Le Chesnay, France. e-mail: Anthony.Tomasic@inria.fr

[1] Other data structures, such as signature schemes and PATRICIA trees, can also be used for IR. We focus on an inverted index in this article.