In Search of Reliable Retrieval Experiments

William Webber and Alistair Moffat

Department of Computer Science and Software Engineering
The University of Melbourne
Victoria, Australia 3010

{*wew, alistair*}*@cs.mu.oz.au*

Abstract *There are several ways in which an "improved" technique for solving some computational problem can be defended: by mathematical argument; by simulation; and by experimental validation. Each of these has risks. In this paper we describe some of the issues that arose during an experimental validation of architectures for distributed text query evaluation, and the approaches that were taken to resolve them. In particular, collections and clusters must be scaled in a way that maximizes comparability between different data sizes; query sets must be appropriate to the target collection; and hardware issues such as file placement on disk must also be considered. Our intention is to report on our experience in a practical sense, and thereby assist others to avoid the same problems.*

1 Introduction

There are several ways in which an "improved" technique for solving some computational problem can be defended: by mathematical argument; by simulation; and by experimental validation. Each of these has risks. For example, a mathematical analysis might be erroneous, or might apply only for impossibly large problem domains, or might be predicated on a model of computation that does not reflect actual computer hardware. Similarly, a simulation might be flawed because it fails to account for some aspect of the real-world behavior that is being modeled.

In this paper we describe some of the issues that arose during an experimental validation of architectures for distributed text query evaluation, and the approaches that were taken to resolve them. The issues discussed are "real", in the sense that each of them turned out to be a major impediment to accurate measurement in a set of experiments that we were running, but had not been anticipated at the time the experiments were initially planned. In particular, we found that collections and clusters must be scaled in a way that maximizes comparability between different data sizes; query sets must be appropriate to the target collection; and hardware issues such as file placement on disk must also be considered.

Proceedings of the 10th Australasian Document Computing Symposium, Sydney, Australia, December 12, 2005.

Our intention is to report on our experience in a practical "warts and all" sense. In our paper describing the new query distribution technique (now being reviewed [Moffat et al., 2005]), we simply stated how the experiments had been run, as if that *modus operandi* had always been the intention. The reality is somewhat different, and our experimental validation took more than a year longer than originally planned, and required a complete rethink of both the software we were testing and also what it was we were planning to measure. We hope that in admitting to our experiences we can inform others planning experimental validations, and thereby assist them to avoid the same problems. Readers who benefit from this commentary might also be interested in the work of Zobel et al. [1996].

2 The challenge

Two standard architectures for distributed text query evaluation are described in the literature. In *document partitioning*, each node in the cluster indexes a different subset of the collection's documents. The central receptionist distributes each query to all of the nodes; and each node evaluates the query against its local index, returning the results to the receptionist. The receptionist merges these results and returns them to the user. In *term partitioning*, each node handles a subset of the index's vocabulary. Queries are evaluated centrally by the receptionist, using index information supplied by the relevant nodes. Previous experimental investigations [Tomasic and García-Molina, 1993, Jeong and Omiecinski, 1995, Ribeiro-Neto and Barbosa, 1998, Cahoon et al., 2000, Badue et al., 2001] had been inconclusive, and we had been engaged in debate as to which method provided superior performance. A new evaluation strategy – denoted *pipelining* – emerged out of that debate, and late in 2003 we set in motion plans to test all three strategies. The pipelined mechanism again makes use of a term-partitioned index, but the evaluation state is shipped between cluster nodes, and each node holding information about a query participates in the evaluation of that query.

A key goal of the experiments was that the testing should be under conditions approximating that of a large-scale, real-world search engine, in accordance

with the position we had already argued for previously [Moffat and Zobel, 2004]. The results should not only probe the potential of our new architecture; they should also conclusively demonstrate the relative merits of document and term partitioning, and settle the arguments we had engaged in.

The uni-processing `Zettair` text retrieval engine (available from `http://www.seg.rmit.edu.au`) was used as a basis for the experiments, with code added to implement each of the distributed architectures. We wanted to explore scalability in two directions: as the number of nodes in the cluster increased; and as the size of the collection grew. Cluster scalability would be investigated by performing runs on one, two, four, and eight machines, and comparing query throughput rates. Similarly, collection scalability would be provided through the use of standard TREC collections of different sizes, `GOV`, `wt100g`, and `GOV2`, being respectively 18 GB, 100 GB, and 426 GB in size. Using these three data sets also had the (as it turned out, specious) attraction of testing the systems on different types of collection. To emphasize the practical nature of the experiments, we chose a convenient real-world query log, the public `Excite97` log. Finally, to eliminate startup effects, the first twenty thousand queries were taken from the log, but timings were taken against the second group of ten thousand rather than the whole set.

With this setup in place, we ran our experiments, got interesting results that made the pipelining strategy look good (and simultaneously exposed the term-partitioning approach as being hopeless), wrote everything up, submitted a paper, and got rejected. In retrospect, most of the reasons given by the referees were appropriate, but as always in such a situation, we felt somewhat disheartened.

Other tasks then intervened; when we returned to the investigation a couple of months later, it was with new hardware, a new version of the `Zettair` engine, and with added instrumentation in the software to allow more data to be collected. These changes incorporated, the experiments were re-run, the presentation revised to include the new data, and the paper re-submitted (to a different venue).

Two weeks later a casual corridor conversation led to the quite shocking realization that there was a major problem with the experiments (described in more detail below), and we withdrew the second submission before – we hope – too much editorial and reviewing effort had been invested in it.

Determined to make our third attempt the last, we re-thought and redesigned the experiments, and checked all of the outputs carefully. But there were several more iterations of design needed, of both software and experiment, before we recently submitted (again) the paper describing the pipelined approach to distributed retrieval. In all, this saga took nearly two years from conception to completion, involved a surprisingly steep learning curve, and taught us several hard lessons about designing and running experiments on distributed systems. Our purpose in writing this paper is to describe the path that was followed, the ways in which flaws in the initial experiments were uncovered, and then the ways they were eventually (we believe) rectified.

3 Homogeneous data

The first issue for reflection was the decision to use three different collections to explore the manner in which the distribution architectures scaled with collection size. At face value, use of three collections of different sizes allows both exploration of scale effects, and also exploration of different types of data. In particular, `GOV` and `GOV2` are both derived from US government web-sites, albeit a couple of years apart (the former in 2002, the latter in 2004). On the other hand, `wt100g` is quite different. It was crawled from the general web in 1997, and was (at least by intent) restricted to HTML pages, whereas the government crawls include many long PDF files. That is, as well as being different in size, the three collections had different subject matter and document length.

Viewing these additional differences as a chance to kill two investigative birds with one experimental stone was misguided. A key tenet of experimental investigations is to know which attributes are being varied, and which attributes should be fixed, perhaps temporarily, or perhaps permanently. An obvious corollary is to then ensure that in any given experiment only one parameter is being varied, thereby "keeping it simple, stupid". By simultaneously mixing data types and collection sizes in our first set of experiments, we were unable to distinguish between alternative possible effects, and were led to erroneous conclusions.

In the subsequent experiments the two smaller collections were dropped. Instead, fractional collections were created by extracting slices out of `GOV2`. This had the benefit of also allowing for sub-collections to be sized in the exact ratios required, important when investigating the effect of (for instance) doubling both the number of nodes and the size of the collection.

Sub-collection extraction does, however, need to be undertaken with care. It would be wrong, for instance, to create a half collection by simply taking the first half of the documents in the `GOV2` repository. This would ignore the way that web crawls proceed, starting from top-level seed documents, and proceeding to deeper, more obscure ones, and would not have created a sub-collection that was homogeneous with respect to the main one. This is particularly the case with `GOV2`, where all the PDFs are stored at the end. Instead, to make a $1/n$th sub-collection we selected every nth file of the $27{,}204$ files making up the `GOV2` collection.

We have chosen to present the main themes of this paper as a sequence of "morals". We begin with this simple one, blindingly obvious, but, nevertheless, one we lost sight of:

Moral: When testing a system, only vary the things that need to vary. Fix everything else.

4 Appropriate query set

Our initial experiments used the Excite97 query log [Jansen et al., 1998, Spink et al., 2001], which had the benefits of being publicly available and widely known. More importantly, it was attractive because it was "real".

However, we subsequently found that the Excite97 log was a poor fit with the GOV2 collection which we used for the main experiments, for two reasons: first, it was collected in 1997, whereas GOV2 was crawled in 2004; and secondly, it is from a whole-of-web search engine, whereas the GOV2 collection is confined to US government web pages and documents. The mismatch means that many of the queries refer to information and resources that government web sites are unlikely to provide. The five most popular queries in Excite97 are "sex", "yahoo", "chat", "playboy", and "porn"; the most popular multi-word queries, "princess diana" and "chat rooms".

From an efficiency point of view, the semantic relevance of the queries to the indexed collection is not terribly important. On the other hand, it is important that the queries have the right statistical properties. In particular, "inappropriate" queries can be processed at quite different throughput rates to "appropriate" ones, especially if "inappropriate" means "without many answers in the collection". Individual query terms in an inappropriate log may be much less frequent in the collection than those in an appropriate log, and there may be many fewer matching answers.

Fortunately, the wt10g TREC corpus does match the Excite97 query log, and can be used to gauge term statistics in the web as a whole. The wt10g collection is a 10 GB subset extracted from the 100 GB wt100g collection, with attention paid to ensuring coherence and document quality. The wt100g collection was crawled in 1997, and the documents were taken from the web as whole, not restricted to a particular set of domains. In terms of both date and domain wt10g is thus a good match for Excite97.

Term	Collection	
	wt10g	GOV2
"sex"	1.87	1.39
"free"	13.59	6.70
"nude"	0.20	0.01
"pictures"	2.54	0.60
"pics"	0.23	0.02

Table 1: Collection frequency f_t as a percentage of the number of documents in the collection, for the TREC collections wt10g *and* GOV2*, and the five most frequent terms in the* Excite97 *query log.*

Table 1 shows the five most frequent query terms in the Excite97 log and compares their occurrence frequencies in wt10g and GOV2. In all cases, the terms are less common in GOV2 than in wt10g, ranging from two-thirds to a twentieth of the frequency. This discrepancy means that, proportional to collection size, the Excite97 log should execute faster against the GOV2 collection than against the wt10g collection. To test this hypothesis, we extracted a slice of the GOV2 collection with the same number of documents as wt10g, and ran 10,000 Excite97 queries against it. The query stream took 16% longer against wt10g than against the GOV2 slice, confirming that the first set of experiments in which we applied the "real" Excite97 queries to the "real" GOV2 collection were probably biased.

In one sense, 16% is not that great a difference, and it could be argued that using the same queries in all runs is sufficient to guarantee fairness. However, Table 1 points to another characteristic of collection inappropriateness that was particularly relevant to our distributed experiments. Consider the notion of *workload*, the amount of work a term imposes upon the system during processing of a query set. Workload is the product of the term's frequency in the query set and its frequency in the collection, the latter measured concretely as the length (in bytes) of the term's inverted list in the index. That is, if $l(t)$ is the length of the term's inverted list in the index, and $f_q(t)$ is the number of times that term occurs in the query set, then the term's workload is given by $w(t) = l(t) \times f_q(t)$, and its proportional workload by

$$\frac{w(t)}{\sum_{t \in T_Q} w(t)}$$

where T_Q is the vocabulary of the query stream. Since both collections and query sets have a skewed term frequency distribution, the workload of a query set's vocabulary should also be skewed.

The term "free" is the second most common term in the query log; occurs in almost a seventh of wt10g documents; and around half as frequently in GOV2. But for the 10,000 Excite97 queries being used in the experiments, "free" is the most workload intensive term, and generates 6.2% of the total workload for the wt10g collection, compared to 3.4% for GOV2.

The importance of workload skew in retrieval experiments should not be underestimated. Two of the three methods we were considering in our experiments involved partitioning the index between nodes by terms. Higher term workload skew means less evenness on average between partitions, which means in turn poorer distribution of processing workload between nodes. We eventually discovered that poor balancing of workload between nodes was the biggest problem in the pipelined system [Moffat et al., 2005].

Table 2 examines the issue of workload balancing in a term-partitioned system (such as pipelining). To create the table, the same sequence of 10,000 queries was extracted from the Excite97 log. The query term vo-

Collection	Query set	4	8	16
wt10g	Excite97	1.209	1.455	1.943
GOV2	Excite97	1.171	1.356	1.684
GOV2	synq	1.205	1.452	1.941

Table 2: The ratio of maximum partition workload to the mean partition workload for different numbers of partitions, using $10{,}000$ *random vocabulary partitionings, and* $10{,}000$ *queries.*

cabulary defined by those $10{,}000$ queries was then randomly partitioned across four, eight, and sixteen processors. The total workload at each node during a run of the $10{,}000$ queries was measured, first against wt10g, then against GOV2 (the third row is explained later). Finally, from this data the ratio of the most heavily loaded node to the mean of all nodes was computed. The figures given in Table 2 are the averages of these ratios over a set of $10{,}000$ different random term partitionings. For example, with $k = 4$ processors, on average a random vocabulary partitioning of wt10g resulted in one of the processors having an assigned workload (in a term-distributed sense) that was nearly 21% greater than the average across the four processors.

In a term-partitioned system the most heavily loaded node is likely to become a bottleneck. Table 2 demonstrates, first, that the problem of workload imbalance grows as the collection is split into more parts; and second, that the imbalance grows more quickly when the query set is appropriate for the collection than when it is not. Use of an inappropriate query stream in our first set of experiments led us to erroneous conclusions about the scalability of the pipelining regime we were testing.

Moral: Mixing real-world data with inappropriate real-world queries may yield artificial outcomes.

5 Synthetic queries

Because GOV2 was the only large collection available, we had no choice but to persist in using it. In the absence of a matching query set – the log from a US government search engine would have been very useful – we decided to generate a synthetic query set that was statistically "appropriate" to GOV2.

Randomly generating queries is a fraught exercise. Consider, for example, a method that uniformly selects terms from the available vocabulary. Such a method would produce only incidental skew in the query term frequency distribution, and result in relatively balanced workload across the partitions. In particular, the median frequency of a term in the collection is small compared to the median collection-frequency of terms in typical query sets.

Synthetic query generation is nothing new in distributed information retrieval research. However, previous methods have been based on one of two unsatisfactory frequency distribution models. The first model is exactly the uniform distribution considered in the previous paragraph [Badue et al., 2001, Tomasic and García-Molina, 1993, Jeong and Omiecinski, 1995]. The second model incorporates skew; however, the skew is usually based upon the frequency of terms in the *collection*, either by directly following that distribution [Cahoon et al., 2000], or by ranking according to collection frequency then fitting to a Zipf distribution [Jeong and Omiecinski, 1995]. However, this is also unsatisfactory, since the correlation between term frequency in natural query logs and term frequency in document collections is relatively weak [Baeza-Yates, 2005]. More generally, there is a problem with attempting to fit query term frequency distributions to models like Zipf's law, even if the ranking and coefficient are based upon natural query logs. It is well understood that, though such power-law models may fit the bulk of the distribution quite well, they often fail to fit its extremes [Baeza-Yates, 2005].

Other work on distributed text query evaluation has used queries created by hand as part of standard query sets used for retrieval effectiveness tasks, such as those published by TREC [Ribeiro-Neto and Barbosa, 1998, Badue et al., 2001]. However, such query sets are very short, typically being composed of only 50 queries – hardly enough to allow the system to even get warmed up. Also, having been hand-crafted by people experienced in information retrieval, they tend to employ a more discriminating (and therefore lower-frequency) vocabulary than do natural query logs.

After considering these alternatives, we set four desiderata, in decreasing order of importance: (1) the query term frequency distribution should be appropriate for the collection; (2) the collection term frequency distribution should be appropriate; (3) the query length distribution should be appropriate; and (4) the query term co-occurrence distribution should be appropriate. Query coherence (or meaning) was not an important requirement, and because we were working with bag-of-word queries (and not phrase queries), it mattered little if the generated queries were nonsensical to human readers.

To meet the four requirements, a query translation method was developed based on term frequency. Real-world queries from the Excite97 log were translated on a term-by-term basis, by finding target terms in the target collection with similar frequencies to those of the source terms in the wt10g collection.

Supposing that C' is a target collection for which queries are required, the details are as follows. The required inputs are an existing query set Q and a source collection C for which Q is appropriate. As before, T_Q is the set of terms occurring in Q; and $|C|$ and $|C'|$ are defined to be the number of documents in C and C'. Similarly, $T_{C'}$ is taken to be the set of terms occurring in C'. For each $t \in T_Q$, a translation term $t' \in T_{C'}$ is picked such that $f_t/|C| \approx f_{t'}/|C'|$, where $f_{t'}$ is the term frequency in C'. This simple process maintains

"spice sex"	"contra vhs"
"cartoon art"	"proposition claims"
"star trek"	"especially eliminated"

Figure 1: Sample query translations, converting Excite97 *queries applied to* wt10g *(on the left) to synthetic equivalents applied to* GOV2*. For example, the term "*sex*", which occurs in* 1.87% *of* wt10g *documents but only* 1.39% *of* GOV2 *documents, is translated to "*vhs*", which occurs in* 1.78% *of* GOV2 *documents.*

identical query length and query term frequency distributions between Q and Q'.

To additionally preserve workload characteristics and term co-occurrence rates, the translation process was performed one query at a time. For each $q \in Q$, there were some terms $t_1 \dots t_i$ that already had translations $t'_1, \dots, t'_i$, and others $t_{i+1} \dots t_x$ that did not. To find a binding for t_{i+1}, a sequence of possible matching terms with the right collection frequency was explored. If the conjunctive Boolean query $\bigwedge_{j=1}^{x} t_j$ had no matches in C, then any suitable set of bindings was assigned to $t_{i+1} \dots t_x$. On the other hand, if $\bigwedge_{j=1}^{x} t_j$ had a non-empty answer set in C, then at least three and as many as seven possible bindings for t'_{i+1} were explored, and the one with the most answers to $\bigwedge_{j=1}^{i+1} t'_j$ in C' chosen. This continued until either all terms in the query had mappings, or until no non-empty target query was uncovered after seven attempts to find a term binding, at which point the remaining terms were assigned any mapping of the right collection frequency.

The synthetic query set Q' generated by this process is referred to as synq. Figure 1 shows three of the queries in synq and their original forms, and illustrates the fact that the translated queries generally do not make semantic sense. However, in conjunction with GOV2 they do match the source query set as it applies to wt10g in all of the statistical aspects that we are concerned with for our experiments. The last row of Table 2 measures the term-partitioned workload skew for synq on GOV2, and exhibits the same pattern of values as does the row that measures skew for the Excite97 queries on wt10g.

6 Careful scaling

Once the data and queries had been fixed, the next issue we grappled with was how exactly to structure experiments so that we correctly isolated scale as a factor – one of the hypotheses we sought to test was that pipelining was "more scalable" than document distribution.

In a uni-processing environment, scalability is established by working with increasing amounts of data, and measuring query throughput (or some related attribute such as average elapsed time). A strategy can claim to "scale well" if, when normalized by the data volume, the throughput rate stays steady or decreases. For example, if a throughput of X queries per second is possible on G gigabytes of data, then a scalable system should deliver $X/10$ queries per second (or more) on $10G$ GB of data.

In a distributed experiment, it is tempting to apply scale incorrectly, and to seek to verify that if one machine can attain X queries per second on G GB of data, then k machines can attain kX queries per second on G GB. In fact, the correct experiment is to apply k machines to kG GB and establish that the rate of X queries per second can be maintained in the face of data growth. At face value these are the same experiment, but there is a subtle difference between the two. When the number of processors changes, and the volume of data is held fixed, other effects can intrude. For example, the k machines also have k times as much memory, meaning that a greater fraction of the index can be in memory.

Indeed, if k machines cannot process kG GB at X queries per second, and one machine can process kG GB at X/k queries per second, then distribution is a failure, since a k-way mirrored system is superior.

In our final set of experiments with GOV2 we were careful to work with homogeneous fractions of the collection, and measured query throughput rates using one processor and $1/8$th of the collection, two processors and $1/4$ of the collection, four processors and $1/2$ of the collection, and eight processors with all of the collection, so as to correctly identify the effects of scale.

Moral: When testing that a process is scalable, be sure that you know what you are actually measuring.

The scaling strategy described in the previous section led to another uniformity issue to do with the mechanics of the query engine. The Zettair system uses a dynamic thresholding scheme that limits the number of accumulators used during processing, in order to bound the amount of query-time memory needed [Moffat and Zobel, 1996, Lester et al., 2005]. Limiting memory use is important if throughput is to be maximized, since many concurrent query threads are likely to be active at any given time. In preliminary experiments with a monolithic system we had found that a limit of $L = 100{,}000$ accumulators was sufficient with GOV2 to achieve a high level of retrieval effectiveness (measured using average precision over a fixed query set, relative to an unrestricted run).

Initially, we applied this limit to each processing node. However, this was unfair to the document-partitioned system. It was forced to maintain kL accumulators system-wide during its (parallelized) processing of each query, compared to L for (serialized) pipelining. That is, we were placing the baseline system at an unfair disadvantage. To be fair to the document-partitioned architecture, the per-node accumulator limit should be set at L/k, which then raises the question as to how retrieval effectiveness behaves. Unfortunately, by this stage we were using synthetic queries, and were unable to assess retrieval effectiveness. Instead, we quantified the extent to

which the rankings varied from each other using a dissimilarity metric. Additional experiments then justified the correctness of the scaled-accumulator approach; and a k-processor document-partitioned split of GOV2 with a $100{,}000/k$ per-node accumulator limit yielded almost exactly the same rankings as a monolithic system with a $100{,}000$ limit.

The second way in which accumulator limits might be scaled is with the size of the collection. When the collection size halves, the target L might also be halved. Again, it came down to the effectiveness results; and in this case, they were more equivocal. The answer rankings (to depth $r = 1{,}000$) did change if the accumulator limit was decreased in proportion to the size of the collection; however, the difference was slight (by the normalized dissimilarity measure, roughly 6% for the smallest collection). This relatively small decrease in effectiveness was acceptable, given the need for consistency in the experiments.

Moral: If in doubt, make choices in a manner that least disadvantages the attribute of the baseline system against which you most wish to compare.

7 Disk placement

As the various inputs to the experiments were refined, variability in the timing results became apparent. The largest index was 16 GB, and the cluster machines each had 1 GB of main memory. Overall performance was thus dependent on the performance of disk reads, and the variability we were observing seemed in some way connected with disk attributes.

Disk performance is primarily affected by two factors: the degree to which the stored data is fragmented into groups of blocks or "extents"; and the physical location on disk of those extents. Fragmentation causes delays as the disk head seeks from one extent to another; the greater the distance between the fragments, the greater the delay. And disk platters rotate at a constant speed, so a disk head takes the same amount of time to complete a rotation at the rim of the platter as it does at its spindle; but more blocks are held per circumference at the rim than at the spindle, due to the greater radial area. It turned out that on the disks we were using, the read-speed ratio between rim-near and spindle-near blocks was around $7{:}4$.

Our initial experiments had been done on machines with a single 220 GB partition for experimental data that was shared with other active users. In particular, it was not practicable to keep it empty of data for the many months of our experiments. The large partition size made our experiments subject to wide variability in disk read speeds between runs; and the presence of other data on disk made file fragmentation more likely. As a result, our initial results suffered from significant I/O-related variability. For example, the system running the full GOV2 index on a single machine could spend anywhere between 5% and 25% of its time waiting for disk. This 20% difference in effective processing capacity led in turn to a similar difference in system throughput, masking the effects we were trying to measure. The I/O variance is especially problematic in a situation such as ours, where the measurements of necessity involve different files of different sizes.

Our first approach to solving this issue was to pre-allocate "storage areas" by creating a number of 4 GB files filled with null bytes on every machine, then reusing these files for each experiment by writing the index data into their existing blocks. In UNIX terms, this means opening the files for writing without setting the `O_TRUNC` flag. (An `rcp` replacement that behaved in this way was developed.) Each time a given index was loaded onto a server, it was then guaranteed to be in the same location. It was not a sufficient solution in itself, however, and did not guarantee that indexes of different sizes received equivalent placement. As an extreme, imagine a file where the first 2 GB were contiguous and rim-wards, but the remaining 2 GB were fragmented across small extents close to the spindle: a 2 GB index and a 4 GB index written into such a storage space would have very different read performance. In addition, file pre-allocation also does not guarantee that the storage spaces on different machines are equivalent.

Storage pre-allocation and reuse, therefore, is not in itself sufficient; that storage space must also be similarly located on different systems, and substantially contiguous in each case. Actually getting equivalent file allocations across different machines, particularly on very large, partially-full partitions, then became a hit-and-miss process of iteratively creating files, finding ones with similar-looking block locations and reasonable contiguity, and then verifying their equivalence by timing tests – a rather tedious and time-consuming process.

Moral: Make sure that the baseline system and the system being compared against it are given equal access to resources.

Having battled to create appropriate storage spaces, we decided that such a process could hardly represent either experimental best-practice or a practical example to others, and in the end we adopted another solution.

The final method we used to control disk location variability was to re-partition all of the disks, and on each machine create a partition just a little larger than the largest index. Constraining the size of the experimental partition meant that the range of possible file locations, and therefore disk read speeds, was similarly constrained. We placed this partition in the fastest section of available disk space, which is also the section of disk with the most blocks per circumference, and hence the smallest speed difference between contiguous blocks. With a dedicated experimental partition, it was then possible to always copy indexes into an initially empty partition.

Moral: If performance is to be repeatable, disk partitions should be kept as small as possible, and should be empty at the start of each run.

Even with all this preparation, we found that XFS, the filesystem we were using, had some peccadilloes that surprised. The first was that even on an empty disk, it does not always write files contiguously, but sometimes leaves moderately sized gaps. (Block allocations under XFS filesystem can be retrieved with the `xfs_bmap(8)` utility.) We therefore automated our index installation tool to check the block allocations, and recopy the files if they contained a gap beyond a certain tolerance (we used 0.5 GB). This step could be omitted by experimenters less embittered about the whole issue than we were by this stage. As for most modern filesystems, there is no filesystem editor available for the XFS filesystem, nor is there any way to force particular block allocations for a given file.

A second peculiarity of XFS's behavior was that it regarded the first block of the partition to be adjacent to the last, and if it started a large file towards the end of a partition, would "contiguously" place the rest of the file at the start of the partition. Moreover, if files and directories were iteratively created and deleted, (which was our initial procedure with indexes), each new directory's files were placed closer to the spindle than the previous directories files, until the end of the partition was reached and the addresses wrapped around. The result was that, over a set of experiments, a sequence of more-or-less contiguous indexes was created, interspersed at regular intervals with a single maximally-gapped one. We discovered this behavior via puzzling cyclical breakdowns in performance over sets of runs.

To counterbalance these annoyances, XFS does have a more useful trait: if all the files in a directory are deleted, but not the directory itself, then any new files created in that directory are located starting at the same block offset as the old ones. We exploited this behavior by creating a couple of dozen directories on each node, and in each directory, creating a file holding a single byte. The directory with the file in the rim-closest block was retained; the remainder were deleted. Then, for all of the experiments, index files were copied into that directory, and accessed via symlinks.

Moral: You may need to become intimately acquainted with the behavior of your disk drive.

During the sequence of extensive timings we discovered one more problem that threatened our sanity – the disk on one of the nodes was 4.5% slower than the disks on the other seven. This was despite the fact that all nodes were purchased at the same time in a single order and with supposedly identical configurations. We contemplated deliberately placing the data partition on this disk in a more rim-ward location than on the other nodes, but in the end decided that life was too short, and instead avoided using this machine for all except the less disk-intensive $k = 8$-node runs.

Moral: There is always something else to go wrong.

8 Dynamic thresholding

There were also useful outputs from our tribulations.

The most tangible of these related to the horrible realization that led to the withdrawal of our second submission of the pipelining paper. The issue in question was algorithmic in nature, and concerned how `Zettair` was implemented. The solution to that problem is now the subject of a separate paper [Lester et al., 2005]; this section briefly summarizes the basis of that work.

Dynamic thresholding was mentioned in Section 3 as a technique for limiting memory usage. The `Zettair` system nominally uses the `continue` accumulator strategy [Moffat and Zobel, 1996]. Under this scheme, postings are permitted to create new accumulators until the limit is reached; thereafter postings may update the scores of existing accumulators, but not cause the creation of new ones. The original design of these algorithms was that the transition between the initial "OR" state and the final "AND" state be made at the end of processing an inverted list. However, the implementation in `Zettair` departed from this by making the transition as soon as the accumulator limit L was hit, even if this was in the middle of an inverted list – a seemingly innocent "interpretation" of the policy that had the benefit of making L a firm limit, rather than a target that might be grossly exceeded.

However, changing states in the middle of inverted lists has the undesirable effect of favoring documents that appear early in the collection, and has an adverse impact on retrieval effectiveness. More critical from our point of view was the consequent behavior of `Zettair` – it contained an optimization that aborted processing an inverted list when the largest document number in an accumulator had been surpassed within that list. In all of the systems, if the first term in a query was the one that triggered the state change, then the largest document number in the accumulator could only be part-way through the collection, and the `Zettair` optimization meant that only the first part of each other inverted list was dealt with.

Even worse, in the pipelined system, the variable recording "largest current document number in the accumulator set" was being passed from machine to machine as part of the query bundle without being properly initialized, and once the accumulator limit had been reached, the inverted lists for subsequent terms were being fetched from disk, but not processed in any way. That is, the system was reverting to the `quit` strategy of Moffat and Zobel, magnifying apparent throughput and giving pipelining an absolute advantage. It was this complete breakdown in experimental rigor that caused us to withdraw the second submission of our paper.

The chain of events involved in this error was uncovered only when, post submission, we thought to check again that the various schemes were identifying roughly the same set of documents, and found that

they weren't. The moral of this section is one that even beginning software engineers are taught:

> *Moral:* After making a change to a program, rerun all of the tests, not just the most recent one.

9 Parallelization

One area that we did get right was that of parallelized query processing. Much of the work in the area of text query processing efficiency has employed a serial processing model, with each query required to complete before the next one is commenced. (Indeed, this was a requirement of the 2005 TREC Terabyte efficiency track). In this model, the objective is to minimize average query response time.

Our investigation allowed the parallel processing of queries, and concentrated instead on minimizing the elapsed time to process a whole batch of queries. In a parallelized environment, average query response time and query throughput rate are not the same. Indeed, one generally maximizes throughput by allowing a high degree of parallelism, at the expense of average query response time. In a sense, the decision to support parallelism was an easy one to make – right from the planning stage, it was clear that pipelining would not come close to matching document-distribution in a serial-evaluation environment.

All three distribution methods were treated equally. The public `Zettair` software does not support parallelism, so modifications had to be added to enable multi-threading. Experiments were then run against each architecture with different numbers of simultaneously active queries, to find the number that maximized throughput, which turned out to be 32 simultaneous queries for all methods. Parallelism allowed both document-partitioning and pipelining to greatly increase throughput compared to serial mode evaluation; using eight nodes, the former increased throughput by over 150%, the latter by almost 600%. Even a monolithic `Zettair` system running on a (hyper-threaded) single-processor system gained a 70% increase in throughput from parallelization.

> *Moral:* Even on single-processor machines, it is unlikely that maximum performance can be attained without parallelization.

10 Conclusion

The initial experiments were promising for the new pipelined architecture, but as we refined our experimental design and techniques, the promise slowly evaporated. We did, however, develop a much richer understanding of how these experiments should be performed, and a much keener appreciation of the important issues in distributed text query evaluation. We now have a strong basis for further research into different techniques for index partitioning and workload balancing in distributed retrieval environments.

Acknowledgments This work was supported by the Australian Research Council. Justin Zobel (RMIT University) provided helpful input.

References

C. Badue, R. Baeza-Yates, B. Ribeiro-Neto, and N. Ziviani. Distributed query processing using partitioned inverted files. In G. Navarro, editor, *Proc. Symp. String Processing and Information Retrieval*, pages 10–20, Laguna de San Rafael, Chile, November 2001.

R. Baeza-Yates. Web usage mining in search engines. In A. Scime, editor, *Web Minng: Applications and Techniques*. Idea Group Publishing, 2005.

B. Cahoon, K. S. McKinley, and Z. Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Transactions on Information Systems*, 18(1): 1–43, January 2000.

B. P. Jansen, A. Spink, J. Bateman, and T. Saracevic. Real life information retrieval: A study of user queries on the web. *ACM SIGIR Forum*, 32(1):5–17, Spring 1998.

B.-S. Jeong and E. Omiecinski. Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(2):142–153, 1995.

N. Lester, A. Moffat, W. Webber, and J. Zobel. Space-limited ranked query evaluation using adaptive pruning. In *Proc. 6th Int. Conf. on Web Informations Systems*, pages 470–477, New York, November 2005. LNCS 3806, Springer.

A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates. A pipelined architecture for distributed text query evaluation. September 2005. Submitted.

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

A. Moffat and J. Zobel. What does it mean to "measure performance"? In X. Zhou, S. Su, M. P. Papazoglou, M. E. Owlowska, and K. Jeffrey, editors, *Proc. 5th Int. Conf. on Web Informations Systems*, pages 1–12, Brisbane, Australia, November 2004. LNCS 3306, Springer.

B. A. Ribeiro-Neto and R. R. Barbosa. Query performance for tightly coupled distributed digital libraries. In *Proc. 3rd ACM Conference on Digital Libraries*, pages 182–190, Pittsburgh, PA, June 1998. ACM Press, New York.

A. Spink, D. Wolfram, B. J. Jansen, and T. Saracevic. Searching the web: The public and their queries. *Journal of the American Society for Information Science*, 52(3):226–234, 2001.

A. Tomasic and H. García-Molina. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In M. J. Carey and P. Valduriez, editors, *Proc. 2nd International Conference On Parallel and Distributed Information Systems*, pages 8–17, Los Alamitos, CA, January 1993. IEEE Computer Society Press.

J. Zobel, A. Moffat, and K. Ramamohanarao. Guidelines for presentation and comparison of indexing techniques. *ACM SIGMOD Record*, 25(3):10–15, October 1996.