Guidelines for Presentation and Comparison of Indexing Techniques

Justin Zobel
Dept. of Computer Science, RMIT, GPO Box 2476V, Melbourne 3001, Australia
Email: jz@cs.rmit.edu.au

Alistair Moffat
Dept. of Computer Science, The University of Melbourne, Parkville 3052, Australia
Email: alistair@cs.mu.oz.au

Kotagiri Ramamohanarao
Dept. of Computer Science, The University of Melbourne, Parkville 3052, Australia
Email: rao@cs.mu.oz.au

Abstract

Descriptions of new indexing techniques are a common outcome of database research, but these descriptions are sometimes marred by poor methodology and a lack of comparison to other schemes. In this paper we describe a framework for presentation and comparison of indexing schemes that we believe sets a minimum standard for development and dissemination of research results in this area.

1 Introduction

Papers describing new indexing techniques are a regular feature of database journals and conferences. As referees of indexing papers we have, for a variety of reasons, found many difficult to evaluate. We were therefore motivated to construct a clear framework for development, presentation, and comparison of indexing schemes, to help guide future work in the area.

There are several specific areas of failing that we have observed in papers submitted to us for evaluation. For a technique to be of interest the reader must learn how it compares to other leading techniques; but such comparison is often lacking. Where comparisons are made they are rarely adequate: important criteria are frequently disregarded, and some comparisons are biased in favour of the new method. Another failing is the use of simplifying assumptions, often to allow tractable analysis, that are unrealistic and distort the results.

Lack of suitable comparison is perhaps the most serious of these failings. We outline the criteria on which we believe comparison should be made, and discuss the four principal methods by which indexing techniques can be compared with regard to these criteria: direct argument, mathematical modelling, simulation, and experimentation.

The methodology we suggest was in part driven by our desire to undertake a formal comparison of signature files and inverted files for text indexing.[1] In that work we applied the guidelines described here to one particular problem; but felt the guidelines themselves to be interesting enough to warrant separate description.

Criteria by which indexing techniques should be compared are discussed in Section 2. It is our hope that authors and referees of indexing papers will use this section as a checklist—and that any omission of evaluation criteria be justified. The four methodologies for comparison are described in Section 3. Some of the pitfalls of comparison are discussed in Section 4. Conclusions are presented in Section 5.

2 Criteria for comparison

An index is a data structure that identifies the locations at which indexed values occur. In the context of a database, an index identifies which records contain which values. Each kind of index is associated with query evaluation algorithms that access this information, and update algorithms that maintain it. When the utility of an index is being evaluated it is not just the data structure that is being considered, but the structure in conjunction with the necessary algorithms.

There are many criteria by which indexing techniques can be compared. We need at a minimum

[1] J. Zobel, A. Moffat, and K. Ramamohanarao, "Inverted files versus signature files for text indexing", Technical Report TR-95-5, Collaborative Information Technology Research Institute, Melbourne, Australia, 1995.

to consider overall speed, space requirements, CPU time, memory requirements, measures of disk traffic such as numbers of seeks and volume of data transferred, and ease of index construction. In a dynamic system we should also consider index maintenance in the presence of addition, modification, and deletion of records; and implications for concurrency, transactions, and recoverability. Also of interest for both static and dynamic databases are applicability, extensibility, and scaleability. All of these considerations will be in the context of assumptions made about the properties of the data and queries.

Assumptions To make a contribution to the study of indexing, it is not sufficient to simply describe a new indexing technique. It is also necessary to provide a demonstration of the value of the method, and place it in the context of other established methods. This demonstration will be based on several constraints and assumptions: the class of data, the class of queries, characteristics of the application—whether updates must be online, for example—and characteristics of the supporting hardware. For example, it is remarkable how many papers on indexing do not include a description of the class of queries to be supported.

Readers will judge the success of a new technique according to its performance on the basis of the stated assumptions—if the assumptions are perceived to be flawed, the demonstration will not be regarded as valid. It is therefore necessary to establish a convincing basis for the demonstration. Assumptions should not only be claimed to be reasonable, they should be argued for, and, where possible, demonstrated as being reasonable. They should not be selected or designed to favour the indexing technique being demonstrated. If the assumptions are questionable, then so too will the results be questionable.

For example, there needs to be a clear argument that the class of test queries is in some way representative. Although real data is often available, sources of real queries are less common—and may simply not exist for new applications—so of necessity test queries are often artificial. But the onus is on the author to persuade the reader that test queries are realistic.

Similarly, assumptions about hardware should correspond to current technology or likely future improvements. The performance of the hardware should be related to well-known benchmarks, to allow comparison with familiar systems and to convey the impression that the technique will be of value on probable hardware—rather than, as we saw argued in one case, on a machine with limited memory but massive arrays of parallel disks.

Applicability Different indexing schemes support different classes of queries; contrast for example inverted files with quad trees. The functionality of an index should always be considered in any comparison.

No indexing scheme is all-powerful. For even simple databases (for example, relations of numerical data) there are classes of query that are difficult to support via an index, such as queries to fetch records where one attribute value is a function of another attribute value. In these cases the only option is to scan the database, and the index is irrelevant.

Extensibility The usefulness of an indexing technique will be limited by the range of query types it can support, and by the degree to which it can be extended to support further query types. An indexing scheme that allows further forms of query, or which can be modified to provide additional functionality, is of more value than an indexing scheme that is otherwise equivalent in power, or possibly even better in some respects, but cannot be so extended.

Scaleability Given that the volume of disk space available for a given cost is rapidly increasing, and that databases are growing correspondingly, indexing techniques need to be able to scale up. To examine scaling it is probably most helpful to consider both average costs as well as best and worst cases. An asymptotic analysis may be interesting—readers should not only learn what might happen with twice as much data as experimented with, but should learn what might happen with twenty or one hundred times as much data.

Scaling can change relative performance of components of an algorithm, particularly algorithms that utilise disk. Having larger datasets can reduce the chance of sequential seeks to the same block or cylinder; can increase data fetch costs, even relative to seek costs (because the former are typically linear in database size); and can even affect the proportion of records that are answers.

Query evaluation speed Perhaps the single crucial test of an indexing scheme is its ability to identify answers to queries in reasonable time. Index speed can only be discussed in the context of a class of queries—specifying a query class and a method of evaluation for those queries is how the performance of an index is measured.

Speed is not always an easy quantity to measure or estimate, since it depends on many parameters: CPU speed, disk capabilities, system load, buffer space available, and so on. Nonetheless some absolute indication of speed should be part of the description of any new indexing scheme, and percentage improvements need an absolute reference point. If possible speed should be described in terms of the performance of some commonly available hardware; and if not, some thumbnail sketch of the hardware in terms of clock speed, disk access time, and so on, should be given. Thus a query might be described as "typically evaluated, for our test data, in around one second on a lightly loaded Sun SPARC 10 Model 512", and preferably even more detail should be supplied.

Speeds that are estimated based upon a mathematical model of computation, or extrapolated from other experiments, should be clearly identified as such; the reader should be able to know immediately whether presented results are the result of actual experiments or of some form of simulation.

When measuring speed in an experiment, tests should always have a "cold start"—that is, execute as if previous tests have not loaded crucial data into system caches. Iteration of experiments is essential to determining of average times, but buffers and caches should be cleared between runs.

Disk space Measuring the disk space consumed by an indexing scheme is straightforward, but it is important to be careful about what is included and what is excluded. For example, the cost of address tables (to convert record identifiers to record addresses) should usually be included when describing schemes in which they are required, because there are schemes that do not require address tables; and items held in memory during query processing should also be counted if they must be stored on disk when the database is not active.

CPU time CPU time can be traded against memory, disk space, and disk accesses, and so needs to be considered in conjunction with these properties. For example, memory space and disk traffic can be reduced if data is stored compressed, but CPU time may be increased because of the cost of decompression. In many applications CPU time is insignificant compared to other costs and can therefore be regarded as negligible, but it should not be ignored altogether: there needs to be some evidence that it is negligible. In some cases it may also be helpful to consider the asymptotic complexity of the processes involved.

Memory requirements Memory requirements are a highly fluid quantity, because they can often be traded directly against disk traffic. The most constructive approach is to indicate how memory can be used by an indexing scheme, and the impact of memory use on other factors.

At one extreme, an entire index can be held in memory. For current hardware, this suggestion is only slightly outrageous: a typical machine has 10 to 1,000 times more disk than memory, and for some indexing techniques the index may occupy only a few percent of the space required for the data. Assuming, however, that this is not the case, memory can be used for search structures and for buffers, both of which can allow big reductions in query evaluation time and update complexity.

Disk traffic Disk costs have two components, the time to fetch the first bit of requested data (seek time) and the time required to transmit the requested data (transfer rate). Transfer rates are more or less stable but seek times are highly variable, as they depend on whether the disk head is at the current track, and, if not, the distance to the requested data. It can therefore be convenient to consider two kinds of seeks, "random" accesses to an arbitrary block of a file, and sequential accesses to the next block of a file. There is also a third kind of access, refetching a block, in which case there is some likelihood that the block will be held in a system cache.

It is increasingly common for disk drives to incorporate optimisations such as reading and buffering whole tracks in response to each block read request; such optimisations make any kind of modelling or prediction approximate at best. The operating system is also a complicating factor, as it intervenes in the reading process in several ways: fetching header and index blocks, caching, swapping, and so on. Broad approximations, close to correct over a long run of accesses, will often be the only realistic way of describing disk performance.

Index construction We have seen many papers in which the index simply "is", without discussion of how it was created. But for a indexing scheme to be useful it must be possible for the index to be constructed in a reasonable amount of time, and so papers describing complex indexing methods should also describe and analyse a mechanism whereby the index can be built. Where possible, index construction costs should be described as a function of the size of the database. Scaleability is of concern during index construction as well as during query processing.

Temporary space requirements during index construction are a consideration that is easy to overlook. A space-economical index is not cheap if large amounts of working storage are required to create it.

Insertion, modification, and deletion When a database is updated—by insertion, deletion, or modification of records—the index must also be updated to reflect the change. Index update costs are often the major component of these operations. For example, in a text database insertion of a record might result in one or more disk accesses to the index for every term that appears in the record. Immediate update is not always required and ameliorations can often be used to reduce update costs. If such strategies are supposed, the assumption should be made clear, and the cost of immediate update also discussed.

Implications for concurrency, transactions and recoverability An index must be consistent with the indexed data. In a production system that manipulates dynamic data there will be intermediate inconsistencies during update, and there will be times at which the index itself is inconsistent. It is also possible in a dynamic system for several updates to be in progress simultaneously. How easy it is to recover from system failure, or even to maintain consistency during parallel access, is another measure of the usefulness of an indexing technique.

3 Comparison of indexing techniques

There are four principal ways of comparing algorithms such as indexing techniques: by direct argument, by mathematical modelling, by simulation, and by experiment. In this section we sketch the characteristics of each of these approaches.

Direct argument It is sometimes possible to construct a formal proof that an algorithm has a certain property, for example that it will always outperform another algorithm in a given respect. Such arguments can be powerful because they imply performance regardless of circumstance. To make such an argument it is necessary to have a clearly stated hypothesis, including a precisely defined model of computation.

This analytic approach has wide currency in the area of algorithm design and analysis, where asymptotic behaviour is of great interest; but is of lesser practicality for database systems, where it is usually unreasonable to ignore constant factors. Nevertheless, the possibility of a comparison using this approach should not be ignored.

Mathematical modelling A model is a mathematical description of a system, based on a small number of independent parameters. Given a description of database size, hardware performance, and the query class, a model should provide an estimate of likely query evaluation time, perhaps in the form of details such as CPU time and number of disk accesses. The model may also provide information such as approximate index size.

Modelling and simulation (described in the next section) both rely on estimation of system performance. For many indexing techniques there are a few simple parameters that can be used: CPU speed, seek time, and disk transfer rate. We suggest that these be estimated by tests on actual hardware, thus allowing at least ballpark comparisons with experimental results. Use of actual parameters will also allow the model to be verified by implementation. Such simple parameters are however an approximation, and researchers should be aware of their limitations. It is difficult, and probably unnecessary, to construct a model that is an exact description of performance.

An implementation of a model is *not* an experiment. Encoding a model in a program and, for example, using it to demonstrate variation in performance as a function of database size can be informative, and can confirm that the model has certain properties. But it does not confirm that the model is an accurate reflection of the proposed indexing scheme, nor does it provide any kind of experimental test.

Simulation A simulation is usually an implementation or partial implementation of an algorithm, complete enough to allow measurement of performance (thus approximating real performance) but easier to undertake than a full-scale experiment. A simulation is less convincing than an experiment, but implementation of at least the skeleton on the method being tested can give a good indication of likely performance in practice.

A simulation is conducted in more of a "white coats" environment than is an experiment. Extraneous factors can be controlled or eliminated, which is often not possible when testing a real system.

Experiment An experiment is an implementation tested with real, or at least realistic, data. Experiments should be designed to yield unambiguous results—with other explanations eliminated and external factors minimised. Ideally, an experiment should be conducted in the light of predictions made by a model: it should confirm (or otherwise) some expected behaviour.

Experiments should be reproducible, which means that not only should they be conducted rigorously but that their description should be sufficiently comprehensive that others can reproduce the conditions and verify the claimed results. Where possible, experiments should be based on benchmarks such as standard sets of data and queries; use of such benchmarks allows easy comparison with other work.

Choose your weapon In practical situations a combination of two or more of these methods might be warranted. For example, one might make a direct argument that the space required by one method is less than the space required by another, for example if it stores a subset of the data. For the same two systems number of disk accesses required by each might be calculated as the result of mathematical models, and the per-record CPU time required during query processing estimated by applying the relevant operations to every record in the database and then dividing by the number of records. Alternately, all of these factors might be measured during an experiment.

Where possible, the approaches should be used to support each other: if certain behaviour is predicted by a mathematical model, and an implementation is also described, then an experiment should be designed to verify that behaviour. Of course, the experiment might not confirm the model; and discrepancies should be accounted for rather than ignored.

4 How not to compare

A comparison between two indexing schemes, or indeed between any two methods of achieving the same ends, should above all be fair. In this section we examine some practices that do not yield fair comparison, drawn from our experience of refereeing and reading indexing papers. These practices are easy to fall into; we have several times ourselves drawn a conclusion about some behaviour or another, only to have some remark from a colleague or referee draw our attention to the unreasonableness of our claims. Some of the examples are drawn from work on inverted files and signature files, an area where we believe many unfair comparisons have been made.

Fool's paradise As discussed in Section 2, assumptions should be reasonable and realistic. For example, an indexing technique for text databases was tested by demonstrating it on conjunctive Boolean queries of ten to thirty terms, the implicit assumption being that such queries are likely. But if query terms appear randomly in 10% of the records—and few words other than stopwords are in this category—then six words provides a selection rate of 1 in 1,000,000. Actual queries with semantically related terms will have more matches, but even so the supposition that conjunctive Boolean queries can have as many as thirty terms seems, at best, dubious. The assumptions about the query set can be improved by stating explicitly that the indexing technique is only suitable for queries involving dozens of terms, in which case the readers will judge for themselves whether the technique is of interest.

Use of simplifying assumptions can be used to make analysis tractable, but can also result in an unrealistic model. Authors should ensure that their models are a reasonable approximation.

Another example of unrealistic assumptions is the use of complexity analysis to condemn B-trees. While it is true that key lookup in a B-tree of n keys has $\log_2 n$ CPU cost, disk costs dominate, and in terms of disk accesses the base of the log is the branching factor of the tree—typically in the hundreds for common database applications. With such large branching factors, only the leaves of the tree will reside on disk, so that (on current hardware) the principal cost of key lookup is likely to be a single disk access.

Moving targets To compare two systems, what is being compared must be clearly defined.

It is crucial to avoid shifting the grounds as comparison as made. For example, a signature file index is variable in size, and so it is not incorrect to claim that signature files can result in very small indexes; and nor is it incorrect to claim that false match rates using a signature file index can be kept arbitrarily low. But the two claims are mutually inconsistent.

Shifting of grounds can arise in subtle ways. For example, signature files can be improved by stopping common words; but inverted files should not then be criticised on the grounds that they give poor performance on common-word queries.

Sauce for the gander A researcher developing a new algorithm is naturally enthusiastic about the work, and will often propose a series of minor refinements and improvements to their method—not significant enough to be of interest by themselves, but certainly worth mentioning in the context of the description of the main algorithm. But what is often not considered is that these minor refinements can apply equally well to rival algorithms.

It is unreasonable to make allowances on behalf of one method but not make similar allowances on behalf of the other. If, for example, one method uses a little more memory than the other, to deliberately set the maximum buffer space to fall between the two memory requirements is not fair practice. It is best to err on the side of generosity to the rival technique.

Chalk and cheese Like should be compared with like. For example, an advocate of signature files could point out that extending a bitsliced scheme to support adjacency queries (in which query terms must be adjacent in answers) results in only a small increase in the size of the index, whereas extending inverted files to support adjacency requires increasing the index size by a factor of three or four. But such a comparison is only fair if it can be demonstrated that the other power that comes with extending the inverted index—support for proximity and word position queries—is for some reason not of interest.

Sometimes the claim is made that two systems are incomparable: that they are apples and oranges. In some senses this claim is not unreasonable; there are many ways of trading off between the criteria listed in Section 2. But comparisons can always be made by fixing values for some criteria and comparing on those that remain, and also by ranking the criteria in order of desirability. In the majority of practical database systems, query evaluation speed is the most important measure of performance, and in typical situations it is valuable to simply compare speeds.

Fish in a barrel When describing the performance of a new technique, it is helpful to compare it to a well-known standard. But there are dangers in this approach, since something that is well known may not be recent work, and may be regarded by other researchers as poor by current standards. For example, some researchers still judge—and denigrate—inverted files by reference to two older papers: one, written in 1981,[2] that estimated that inverted files require 50%–300% of the space required for the data; and another, written in 1975,[3] that estimated that each term occurrence requires up to 80–110 bits of index. These papers no longer reflect the capabilities of inverted files, and should not be used as a basis of comparison. Likewise, signature files should not be condemned on the basis of bitstring techniques.

Furthermore, it is not reasonable to characterise a rival technique by an implementation that has unrepresentatively poor performance. Comparisons should be to a competent, pragmatic implementation. Indeed, one should actively seek the *best* rival implementation.

Proof of the pudding Ultimately, claims should be "sensible" and capable of being verified by other researchers, preferably by a variety of mechanisms. Experiments should be blind, and not over-parameterised. It is not acceptable, for example, to develop a system that requires that values be specified for a variety of parameters and constant coefficients, tune the values for those parameters to give excellent performance on one particular set of test data, and then claim that similar performance is likely on any data set.

Finally, lack of comparison to any other actual system is always a weakness. Vague claims that "the method performs well" are insufficient defence against the most important question of all: does the proposed method improve the state of the art in some useful and interesting manner.

5 Conclusions

This paper has been a chance for us to articulate a range of observations accumulated over an extended period of time. We have been frustrated by being asked to judge the work of others, and finding that insufficient information was provided to allow that work to be fairly evaluated.

In some of the other fields of computing, frameworks for comparison are well established. For example, one would not hope to present a new algorithm to the algorithms community without a rigorous proof of correctness and an asymptotic analysis that explicitly states the conditions under which the new method might hope to be superior.

For database indexing this formality, even were it observed, is not always sufficient, since most methods are asymptotically linear or near-linear and constant coefficients must be involved in all arguments as to superiority. We hope that the various points of comparison we have listed will be adopted and perhaps extended, and that authors of future papers on indexing will take care to answer the questions explicitly and implicitly raised in our list of points. We believe that adoption of such a "comparison checklist" will benefit all the members of this diverse research community.

Acknowledgements

We would like to thank the Multimedia Database Systems and Deductive Database groups at the Collaborative Information Technology Research Institute CITRI. This work was supported by the Australian Research Council.

[2]R.L. Haskin, "Special purpose processors for text retrieval", *Database Engineering*, 4(1):16–29, 1981.

[3]A.F. Cárdenas, "Analysis and performance of inverted data base structures", *Communications of the ACM*, 18(5):253–263, 1975.