# On Concurrency Control For Inverted Files

**A. MacFarlane*, S. E. Robertson, J. A. McCann**
*School Of Informatics, City University, UK*
* Email; A.MacFarlane@lpac.ac.uk

## Abstract

Few if any Information Retrieval (IR) systems have had to deal with Concurrency Control (CC) on inverted files. In order to examine the issues involved in CC on inverted files, the effects of various operations (e.g. Boolean) on the effectiveness of the IR system are examined using the example of interleaved transactions. Solutions to the problems identified are examined by discussing the three main CC mechanisms; Locking, Optimistic CC and Timestamp Ordering. The effect of delays and document availability are examined. The problem of stored sets is identified. The need for further work in the area is identified.

## 1 Introduction

To date Information Retrieval (IR) systems with inverted files have not had to deal with Concurrency Control (CC), since searching has taken priority over update. Insertions are usually done off-line and en-masse when no one is using the system e.g. overnight. Such methods are not suitable for systems where information is received at more frequent intervals and 24 hour access to this information is required e.g. a News Service. The spread of the Internet is likely to increase the need for systems that do fast update, while servicing multiple queries e.g. the Guardian newspaper has recently gone on-line [1]. To the best of our knowledge there has been no work on using CC mechanisms on inverted files to address the issue. This paper attempts to define some of the problems by looking at the effects of operations on queries in the presence of an incorrect or non-existent CC mechanism while updating the inverted file, and outline some areas of concern such as delays and availability, and stored sets. We do not propose any specific solutions at this stage. The structure of the paper is as follows. Section 2 describes the assumptions made in this paper. The scope of operations for queries and insertions is outlined in section 3. Section 4 gives a statement of the problem by looking at interleaved transactions. Section 5 describes the effect of incorrect or non-existent CC mechanisms on query operations. Section 6 discusses the three main CC mechanisms, locking, Optimistic CC and Timestamp Ordering within the context of the problem statement and the effects described in section 5. Section 7 discusses the

issue of delays and document availability and introduces the concept of document availability semantics. Section 8 examines the issue of stored sets. A summary and conclusion are given in section 9.

## 2 Assumptions

Throughout this paper it is assumed that no deletions or updates are done on Inverted files, only reads and insertions. This is because Text Databases tend to be archival in nature, hence exhibit dynamic behaviour that is that of growth rather than slight fluctuation in size or decrease in size. We also assume that an Inverted file may be fragmented and stored on several different disks. The Inverted File structure assumed in this paper is as follows; an Index or Dictionary file which contains lists of keywords together with the number of hits and a pointer to a list of postings (document identifiers) in which the keyword occurs; the Postings file or Inverted List which contains a document identifier, a link to the document in the documents file and a list of positions for a given keyword. The structure is taken from [2] and revised to include position information.

## 3 Scope of operations

An insertion of a term results in one of two things; i) a new term, its posting list and position information is added to the Inverted file; ii) for a existing term a posting is appended to that term's postings list, the position information is stored and the dictionary file entry is updated (number of postings entry is incremented). A read done on a particular term results in; i) a message to indicate that there are no occurrences; ii) a posting list (with position information if that is required) is returned for *n* document id's in which that term occurs. The operation WRITE_TERM takes on the semantics of an insertion and the operation READ_TERM takes on the semantics of a read. The action on both READ_TERM and WRITE_TERM is to consult the dictionary file first, and then the postings file.

## 4 Statement of the problem

The general problem of concurrency control arises because of possible conflicts between transactions operating at the same time on a database. In the context of IR this can best be illustrated by an example involving a more or less simultaneous query and addition of a new document to the database. Consider the following scenario in figure 1 where an insertion transaction is interleaved with a query transaction that access the same data set. Our aim is to make these transactions have the same effect as if they where executed sequentially i.e. they are serially equivalent [5]. The Insertion and Query access the same terms.

| TIME | INSERTION | QUERY |
|---|---|---|
| i | WRITE_TERM(term1, docid) | |
| i+1 | | $postings_1$ = READ_TERM(term1) |
| i+2 | | $postings_2$= READ_TERM(term2) |
| I+3 | WRITE_TERM(term2, docid) | |
| | | answer = $postings_1 \otimes postings_2$ |

FIGURE 1 - Example scenario

The problem in this scenario is that the query has done a read before the insertion has finished a write. In Distributed Systems this problem is known as a dirty read, which is caused because the READ_TERM operation on term2 in the query conflicts with the WRITE_TERM operation on term2 in the insertion. An easy answer would be to prevent the query from reading until the insertion is complete. However different operations $\otimes$ will produce different side effects, therefore a simple block may delay a query unnecessarily. The following section describes the effects of various operations.

## 5 Operations and their effects

Examples of effect of incorrect or non-existent CC mechanisms on the operations AND, OR, AND NOT, PLUS/DOT, ADJ, SAMES, LIMIT and MIXED are discussed below. Explanations of some of these operations are given in appropriate sections. It should be noted that the operations XOR, NAND, NOR and unary NOT are left out since they are not used in operational systems. These operations and their semantics are taken from [3]. A comparison of the effects is given at the end of the section.

### 5.1 ⊗ = AND

The side effect found where ⊗ = AND is that the query will fail to retrieve a relevant document because the postings list for term2 does not contain the identifier for the inserted document. To prevent this from happening the query must be excluded from reading information for term2 until the insertion has completed its action. The side effect is that of a false dismissal.

### 5.2 ⊗ = OR

The side effect will depend on whether position data is used by later operations. If no position data is involved, there is no apparent side effect where ⊗ = OR since the document identifier for the inserted term will be in the postings for term1. Therefore there is little point in blocking the read from term2, since it has no effect on the retrieval using that operation. To do so would delay the operation unnecessarily and result in a reduction in concurrency. We will name the side effect where an unnecessary block is made a false delay. However position information for term2 will not be included in any result set. Therefore any later operations on the sets that involve proximity operations may cause false dismissals e.g. given the query {*information* ADJ {*science* OR *retrieval*}} where term1 = *science* and term 2 = *retrieval*.

### 5.3 ⊗ = AND NOT

This operation is asymmetric, therefore the side effect is dependent on the order of the reads. For example given the sets term1 = {1,2,3,4} and term2 = {1,2} before insertion;

| INSERTION | QUERY | RESULT SET |
|---|---|---|
| | READ_TERM(term1) | {1,2,3,4} |
| WRITE_TERM(term1, 5) | | {1,2,3,4,5} |
| WRITE_TERM(term2, 5) | | {1,2,5} |
| | READ_TERM(term2) | {1,2,5} |
| | answer = {1,2,3,4} AND NOT {1,2,5} {3,4} | |

Figure 2 - Example interleaving with AND NOT

A document with an id = 5 is inserted in the Inverted file. The correct documents are retrieved. The side effect could be a false delay since the read operations do not conflict with the writes, *because of the order in which they were done*. However if the order is changed;

| INSERTION | QUERY | RESULT SET |
|---|---|---|
| | READ_TERM(term2) | {1,2} |
| WRITE_TERM(term1, 5) | | {1,2,3,4,5} |
| WRITE_TERM(term2, 5) | | {1,2,5} |
| | READ_TERM(term1) | {1,2,3,4,5} |
| | answer = {1,2,3,4,5}AND NOT {1,2} | {3,4,5} |

Figure 3 - Example interleaving with AND NOT; order reversed

The read and writes conflict because the id for document 5 is inserted in term2's set and not term1's set. The side effect is a false drop.

### 5.4 ⊗ = PLUS, ⊗ = DOT

These operations are generic functions for term weighting as per the probabilistic or vector space models (if you include normalisation in the latter). The result of the PLUS operator is a simple sum-of-weights [3]. The result of the DOT operator is a dot-product i.e. the sum of products of posting and set weights [3]. The side effect where ⊗ = PLUS or DOT is that the weight for term2 will not be included in $postings_2$ which could have the results; i) loss of weight for term2 drops the total weight down to an incorrect ranking or ii) the total weight for term2 drops below a stated threshold (as in the vector space model) and the document is incorrectly rejected. Therefore there are two side effects for the PLUS or DOT operators; either a rank drop or a false dismissal.

### 5.5 ⊗ = ADJ

The ADJ operator is used to find two terms that are adjacent to each other [3]. The discussion of this operator is based on an ordered ADJ. We assume with the operator ADJ that term1 and term2 have matching position information on the insertion. Position information must be in both sets for a given term for any meaningful comparison to take place. Therefore since no match can be made where the position information for either term1 or term2 is absent, the resulting side effect is a false dismissal. It should be

noted that while the operation is asymmetric i.e. *information* ADJ *retrieval* is different from *retrieval* ADJ *information* the side effect is symmetric i.e. no matter what term set is read in first, the result is still a false dismissal.

### 5.6 ⊗ = SAMES

The SAMES operator is used to find terms that occur in the same sentence [3]. The assumption on ADJ position information applies to the SAMES operator. Unlike ADJ, the operator SAMES is symmetric. As with ADJ no match can be made where the position information for either term1 or term2 is absent, therefore the resulting side effect is a false dismissal.

### 5.7 ⊗ = LIMIT

In [3] it is suggested that term1 LIMIT term2 is restricted to a search on term1, limited to items that satisfy term2. Therefore in strictly Boolean terms the LIMIT operator is identical to AND; therefore the side effect is identical i.e. a false dismissal. While the operator is not symmetric, the side effect is.

### 5.8 Mixed operations

A query may contain more than two terms and may use any of the above operations, where such is legal e.g. term1 ADJ (term2 OR term3) is valid, but term1 ADJ (term2 AND term3) is not. What is being considered here is the interaction of binary operations or nesting of binary operations. The complexity of the side effects for these nested binary operations could be considerable. We give an example interleaving in figure 4 with a query; *retrieval* AND (*science* AND NOT *information*). The sets for each of the terms before the interleaving are; *retrieval* = {1,2,3}; *science* = {1,2,3,4}; *information* = {3,4}.

| INSERTION | QUERY | RESULT SET |
|---|---|---|
| | READ_TERM(retrieval) | {1,2,3} |
| WRITE_TERM(retrieval,5) | | {1,2,3,5} |
| WRITE_TERM(science,5) | | {1,2,3,4,5} |
| | READ_TERM(science) | {1,2,3,4,5} |
| | READ_TERM(information) | {3,4} |
| WRITE_TERM(information,5) | | {3,4,5} |
| | {1,2,3,4,5} AND NOT {3,4} | {1,2,5} |
| | {1,2,3} AND {1,2,5} | {1,2} |

Figure 4 - Example interleaving with MIXED operations

There are a number of observations to be made on this interleaving; the first being that the read on the query and write on the insertion conflict on the term "information"; as a consequence, the next-to-last result is incorrect. However this does not effect the final result since the error is masked by the AND operation. This is a by product of this particular query; other more complicated examples may actually reintroduce the problem. We can infer that the interaction between binary operations determines which side effects occur, if any. The interleaving will also have an effect, but is not just a characteristic of mixed operations. We cannot decide what the side effects will be by simply looking at the constituent binary operations.

## 5.9 Comparison of operator effects

Table 1 shows a comparison of the effects on the operators discussed above.

| *OPERATOR* | *SYMMETRIC* | *SIDE EFFECT* | *COMMENTS* |
|---|---|---|---|
| AND | YES | False Dismissal. | - |
| OR | YES | False Delay.<br>False Dismissal. | For unnecessary blocks.<br>For position data only. |
| AND NOT | NO | False Drop, False Delay. | Depends on order of term insertion. |
| PLUS, DOT | YES | Rank Drop or False Dismissal | Side effect depends on threshold limit set (if any). |
| ADJ | NO | False Dismissal. | Side Effect is symmetrical. |
| SAMES | YES | False Dismissal. | - |
| LIMIT | NO | False Dismissal. | Side Effect is symmetrical. |
| MIXED | POSSIBLY | Any or all. | Side effects could be very complex. |

Table 1 - Comparison of operator effects

# 6 Concurrency control mechanisms

It should be noted that the above are only a small number of simple scenarios. But it does give a flavour of some of the problems that may occur if an incorrect or non-existent CC mechanism is used. From the above we can deduce two very important facts. The first is that we need to vary the exclusiveness of blocks on term postings according to the operation. The second leads on from the first and suggests that the query model used will have implications for side effects. The concept of isolation levels much used in Database systems[4] is regarded as useful.

Isolation is defined as the degree of interference a transaction can tolerate [4]. We can define an isolation level for queries which in which non-conflicting situations occur e.g. reads can be allowed on OR's thus preventing the false delay side effect. A further point is that blocks on terms that are popular (i.e. there is a high rate of retrieval on them) are likely to cause bottlenecks. Therefore the blocking granularity (the size of object being blocked) used for the postings file will be crucial not only in determining the retrieval performance, query throughput and system utilisation, but the CC mechanism performance as well. If the granularity is too large then unnecessary blocks will be made causing delays. The following discuss the three main CC techniques used in Distributed Systems [5] in the light of the above. The mechanisms are; locking, optimistic CC and timestamp ordering.

## 6.1 Locking

The Lock method is the most common form of CC. The method works by setting a lock on a data item that blocks out other conflicting operations. What you tend to find is that concurrent reads are allowed but only one write is allowed at any given moment and a read may not be allowed while a write is being serviced. This is known as the many readers, single writer problem. Having reads which lock out each other is far too exclusive and reduces concurrency. The main advantage of the lock method is that it is better in environments where operations are predominately updates. The main disadvantage is deadlocks occur (see below) which need to be resolved, particular if more than one fragment of an Inverted file is accessed.

The operation of Locks is simple. In the case of the scenario in section 4 term1 is blocked by the insertion preventing the read on term1 in the query. The query is blocked on term1 until the insertion releases the lock on both the accessed terms in one go; the query can then retrieve both terms as normal with no conflict. However for the situation with OR's this causes false delays. We remedy this problem by setting a level of isolation that allows queries with the operation OR to proceed without attempting to set a lock. A lock is released when the transaction has finished with that particular term. If a lock is set for a data item that already has a lock held on it , it delays that request until the lock is released.

How do we resolve deadlocks in Inverted files? Consider the following scenario in figure 5;

| TIME | $INSERTION_1$ | $INSERTION_2$ |
|---|---|---|
| i | WRITE_TERM(term1, $docid_1$) | |
| i+1 | | WRITE_TERM(term2, $docid_2$) |
| i+2 | | WRITE_TERM(term1, $docid_2$) |
| I+3 | WRITE_TERM(term2, $docid_1$) | |

Figure 5 - Example deadlock

$Insertion_1$ requests a lock on term1 and $Insertion_2$ requests a lock on term2. The problem occurs when $Insertion_2$ requests a lock on term1 and $Insertion_1$ requests a lock on term2. Both are now deadlocked and the data items are inaccessible until one or the other of the insertions is aborted (and re-started at a later time). There are a number of ways to decide which insertion to abort; i) abort the youngest insertion to allow the older one to commit straight away; ii) choose an insertion that uses up the least machine cycles and abort that. In the case of distributed deadlock servers need to reach some form of distributed agreement on which insertion should be aborted. Further consideration is needed for the Lock method including the use of hierarchic Locks and Lock promotion, particularly for MIXED query operations.

## 6.2 Optimistic CC

The Optimistic CC method takes a different view of blocking; it avoids it all together. Isolation levels are therefore not needed. An Optimistic strategy is used which allows transactions to proceed irrespective of the effect unless a conflict is found. It should be noted that the other two

methods, locks and timestamps use a pessimistic strategy. The method works by keeping a tentative version of a data item, while the transaction is being processed. The use of tentative versions allows the transaction to abort without the need to rollback the effect of a given operation. There are three phases to Optimistic CC; i) Read Phase; Data items are read in from disk and are put in tentative versions. This phase is never interrupted. The various operations such as writing are done on the tentative versions. ii) Validation Phase; After the Read Phase is complete the transaction is compared with other transactions and if any conflicts are found, a transaction is aborted. Otherwise the transaction proceeds to the Write Phase. iii) Write Phase; Read only transactions can commit immediately while transactions with writes in them make their tentative versions permanent. There are two types of Validation; forward and backward. Forward Validation checks the current transaction with later transactions and works by comparing the write set of that transaction with the read set of later overlapping transactions. Backward Validation checks the current transaction with earlier transactions and works by comparing the read set of that transaction to the write set of earlier overlapping transactions. The main advantage of the method is that it is fast in the presence of few conflicts, but the disadvantage is that a substantial body of work may need to be repeated if there are many conflicts and starvation (non-service of a transaction) may occur with some transactions; this could have an effect on system utilisation and throughput. No deadlocks occur with the method.

Non conflict operations (OR's) do not need to be validated against any other transaction. With Backward Validation a query or insertion is checked against earlier insertions; if any overlaps are found the query or insertion is aborted. With Forward Validation we compare an insertion with later queries or insertions and either abort the transaction being validated or the later transaction. We can see that in Forward Validation we have the option to either abort the transaction or a later one; with Backward Validation we only have one choice, to abort the current transaction since earlier transactions have already committed. However Forward Validation is more complex than Backward since it has to account for new transactions starting whilst still in the validation process. The practicality of using the method on the Inverted File CC mechanism will depend on the update rate i.e. the higher the update rate, the more chance of conflict and the less the method is useful. For situations where insertions are rare, Optimistic CC could be useful.

## 6.3 Timestamp ordering

With this method a transaction is assigned a timestamp when it is initiated. The timestamp can take on a physical or logical value. The method works by comparing timestamps and if there are conflicts then a transaction is aborted. Each operation is validated as it is executed. There are three simple rules for transaction conflicts; i) To be able to write, a transaction must have the maximum read timestamp to prevent conflict on reads of other transactions; ii) To be able to write, a transaction must have the maximum write timestamp to prevent conflicts on writes of other transactions; iii) a transaction can only read a data item where the timestamp has a later value than the committed version to prevent conflicts on reads. We can allow for an isolation level in the event of a non-conflicting operation by ignoring timestamp comparison. The main advantage of the method is that is better for environments where reads outnumber writes. The main disadvantage is the timestamps determine the order of serialisation statically, according to the value assigned.

In IR systems the method is simple and would works as follows. A timestamp is compared only if the isolation level requires it. Operations where conflicts could occur have their timestamps checked as per the rules above. A problem could occur if an insertion has an older timestamp than a query. Because of rule i) we have a conflict and therefore have to abort the insertion. As with Optimistic CC this could be problematic in certain applications.

## 6.4 Comparison of CC methods

From the above we have a number of choices for CC. The target application will determine which of the three would be suitable. If there are more queries than insertions, then we would suggest that the Timestamp ordering method be used. If insertions are more frequent than queries, then we suggest that the Locking method be used. If there are few insertions, then the optimistic CC method could be useful. It is possible to use a more analytical approach to estimate the actual delays and overheads of each mechanism that would provide a more accurate comparison, but this is not attempted here. Such would include the cost of lock overheads, transaction rollback etc.

## 7 Delays and availability

The availability of documents will depend on the application being considered e.g. a News Service or an On-Line Public Access (OPAC) system. The availability semantics would determine what CC mechanism to use. We could perhaps limit availability of documents to the log-on time, where any documents are unavailable if they are inserted during the session. In such a case no CC mechanism would be needed, since documents could be inserted overnight. This would not be suitable for a News Information Service application, where news from around the world is needed as soon as possible. We term these semantics as Log-On availability semantics. If we take the semantics that do not make documents available until the last write in the insertion is complete, then many of the mechanisms described above would not be needed since a false dismissal would not be deemed to have occurred. False dismissals cannot occur with such semantics. We will call such semantics Late Availability. However other side effects cannot be so easily ignored. To allow queries and insertions to go ahead without blocking could lead to false drops or ranks drops. Therefore some blocking will be needed in Late Availability semantics. This has the unfortunate side effect of delaying some queries for a period, when such blocks are needed. If we take semantics that make documents available when the first write in the insertion is completed, a full set of the mechanisms described above would be required. We will call such semantics Early Availability. Where queries share terms with insertions and a timestamp say needs to be registered, all queries will need to be delayed. This is clearly undesirable. By its nature the Inverted File technology gives priority to queries. This is because Inverted File search is comparatively cheap while Inverted File update is very expensive. Searching speed is the reason that Inverted Files have become the dominant technology in IR. Therefore the assumptions made in this paper may not be sustainable for many applications. However certain side effects such as rank drops or false drops would be very undesirable in an IR system. To prevent these some delays may be needed. It is a question of determining how far search is offset against insertion. This can be more accurately determined using an analytical model.

Another important factor that could have a dramatic impact on performance is unnecessary processing. It could occur with Early Availability semantics because of the possibility of starvation of queries, which may be aborted and re-started a number of times if one term or group of terms are being updated constantly for a time period (e.g. a spurt of News

on a particular subject). It could occur with Late Availability semantics where a term1 say is read before an insertion is complete and has to be compared with a number of other term's information. Where postings lists are large this processing could be prohibitive.

From the above it can be seen that we have a number of conflicting requirements. We want to make documents available as soon as possible, without delaying queries unduly. We do not want to process information for a query that is not relevant, but on the other hand we do not want the occurrence of multiple aborts. Such choices come about because of the "black and white" nature of Late and Early Availability semantics, which take rather an extreme viewpoint on document availability. We need to find another form of availability semantics that does not introduce too many unnecessary delays and does not do too much unnecessary processing. It should be noted that we are unlikely to be able to find a system of semantics in which no delays occur and unnecessary processing is never done.

## 8 Stored sets

Many practical IR systems store sets during the course of a user session. The sets could either be result sets or the complete sets for all the terms in the query and any intermediate result sets. These may or may not be stored on disk for future use. The problem occurs because the sets will become outdated because of insertions, therefore we cannot rely on the accuracy of stored sets in the presence of document updates. Depending of the required document availability semantics we may need to update these stored sets to take account of new information.

## 9 Summary and conclusion

Operators and the possible side effects found in the presence of an incorrect or non-existent CC mechanisms have been identified. The three main CC mechanisms (lock, Optimistic CC and Timestamp Ordering) are used to show how the side effects from these operations can be avoided, in particular the effect on document availability. Three semantics for document availability have been introduced i.e. Log-On, Late and Early. Late Availability or Log-On semantics would be suitable for many IR systems e.g. OPACS. We do not see the Early Availability semantics as being practical at this point. The problem of Stored Sets is identified. Further work is needed at some stage to identify the complexity of the side effects found with

MIXED operations and how CC mechanisms are to be used in conjunction with Inverted Files. We hope to encourage some discussion on the subject that will address some or all of the problems stated above, in particular the investigation of different types of document availability semantics. We also intend to do some further research on CC in inverted files at some stage, by devising an analytical model in order to examine the use of the three main CC mechanisms discussed in this paper.

## References

**1.** The Guardian, 16 November 1995.
**2.** Harman D, Fox E, Baeza-Yates R, Lee, W. Inverted Files, In: W. B. Frakes and R. Baeza-Yates (Eds), Information Retrieval Data Structures & Algorithms, Prentice Hall, 1992.
**3.** Robertson S.E, Walker S. On the logic of search sets and non-Boolean retrieval, unpublished paper, School of Informatics, City University.
**4.** Date C.J, An Introduction To Database Systems, Volume II, Addison-Wesley, 1983.
**5.** Colouris G, Dollimore J, Kindberg T, Distributed Systems: Concepts and Design, Second Edition, Addison-Wesley, 1994.