Volume : 1 | Issue : 5 | Oct 2012 • ISSN No 2277 - 8160

Research Paper | Computer Science

Comparisons and Analysis of Object Locking , Field Locking and Commutativity Locking in Industrial Platform Environment

Dr. Jayesh M. Patel Associate Professor – MCA Department , Ganpat University, Gujarat , INDIA

ABSTRACT

While the dominant approach to persistent storage in practice is to use a relational DBMS, there are some specialist applications that rely on object stores. The performance of these applications depends on the efficiency of the object store's concurrency control mechanism. Today's predominant concurrency control mechanism is strict two-phase object locking. In the 1980s, an interesting alternative was developed: commutativity-based semantic locking. In theory, it can outperform traditional locking schemes in certain scenarios with appropriate commutativity potential. In this paper, we study the real-world performance of different locking strategies in a particular industrial application from the telecommunications sector. We compare object-based locking and commutativity- based semantic locking. We found that, in this application, semantic locking performs equally to, but no better than, object locking in reasonable mixes of real- world transactions, and that it only outperforms in a deliberately contrived mix.

KEYWORDS: Concurrency control, persistent object systems, object-oriented databases, transaction processing.

1. Introduction

MANY computer-based systems support the daily needs of business using persistent data that is shared between application programs. Although the data can often be represented well in a relational database, there are many specialist application domains that handle more complex data models. Thus, one approach to allow sharing of information between application programs uses the application model of the data as instances of classes in an object-oriented programming language; the application programs can store and retrieve these objects directly from an object-store. Just as in relational systems [6], concurrency control is needed to prevent anomalies such as lost update.

The simplest approach to concurrency control in object stores adopted traditional DBMS algorithms (such as two-phase locking or timestamps) which need no information about the data types or the application transactions other than seeing each read or write as it is submitted. This is the approach that has been followed in the commercial object databases that have achieved some impact. However, since the early 1980s, there has been recognition that, given knowledge of the semantics of the data and/or the transactions, one could allow some activities to run concurrently even though both modify the same data. The classic example is to allow concurrent transactions that include deposits to a bank account (of course, the actual deposit method calls must be synchronized properly, but transaction-duration write locks are not necessary). Since concurrent transactions are allowed to include operations when the order of operations is not significant, this type of concurrency control is called commutativity-based semantic locking.

While many papers propose different approaches to concurrency control, there has not been as much done to evaluate the impact of each algorithm on the performance of the resulting system. Much of the research has simply assumed that, if a concurrency control algorithm allows more situations where clients can run concurrently, then this will imply better performance. Two main lines of research have attempted to use simplified models of the software to determine how performance will be affected by different algorithms: one methodology has used simulation studies; the other has used queuing models. The important work in these directions is summarized in classic papers by Agrawal et al. [2] and Thomasian [15].

We show that in reasonable transaction mixes there is low commutativity potential, and all three concurrent locking approaches yield very comparable performances, which is typically 3-4 times better than a serial execution. Only in a transaction mix that was contrived to have high commutativity potential does semantic locking outperform the other locking concurrency control mechanisms by 60 percent.

2 BACKGROUND AND RELATED WORK

2.1 Related Work

The studies on concurrency control using semantically rich operations began in the 1980s by Weihl [17] and Schwarz and Spector [14]. Badrinath and Ramamritham extended the idea to a richer definition of recoverability [3] and presented simulation studies comparing performance [4]. Later, Xavier and Ramamri- tham [21] extended the performance evaluations to include other optimistic protocols. Agrawal and El Abbadi [1] presented a different locking protocol using commutativity. For O2, an object- oriented database [9], the idea to exploit method commutativity is in Cart and Ferrie [5]. Muth et al. [12] presented a locking protocol ´ for object databases that uses method commutativity. A semantic locking protocol by Resende et al. [13] deals with referentially shared objects. In 1993, Malta and Martinez discussed automating the construction of commutativity relations on object methods [11]; this was extended by Jun and Gruenwald [8]. The book by Lynch et al. [10] proved commutativity-based locking algorithms in detail. Jajodia and Mukkamala [7] offer a probabilistic model to estimate the reduction in transaction conflicts when commutativity was used for concurrency control. Topics on general commu- tativity, state-dependent commutativity, return value commuta- tivity, and commutativity-based reducibility are all discussed comprehensively in a 2001 book by Weikum and Vossen [18]. In 2003, we published a work on classifying commutativity categories in an industrial application, to compare the different concurrent locking potentials [19].

2.2 Commutativity-Based Semantic Locking

The definition of commutativity is as follows: For a given object X, an operation pair ðOP i; OP jÞ acting on X is commutative if the

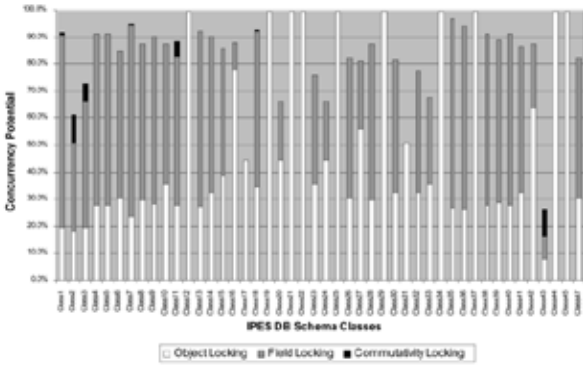


Fig. 1. Concurrency potential of the locking approaches. In each class, the tip of the bar represents the maximum concurrency potential achievable expressed as a percentage of the total number of conflict pairs in the class. Class 46 is not included in the figure because it has no operation pairs.

execution of OP i by transaction T 1 followed by the execution of OP j by transaction T 2 has the same final effect on X and the same results for the transactions as the execution in reverse order (OP j followed by OP i).

Commutativity allows greater concurrency than object locking. Consider a bank account with a deposit operation. which is a write operation since it modifies the state of the bank account. Under object locking, two deposit operations are in conflict and their respective transactions are not allowed concurrent execution. However, running two deposit

operations in either order makes no difference to the final observable state of the bank account (provided the return value of a deposit operation, if there is one, is a success indication rather than the new balance). Thus, semantic locking allows these in concurrent transactions.

Commutativity applies readily to databases based on the object model. An object has a set of operations (i.e., its methods), and conflict between these operations can be defined by commutativityity. A transaction in an object database consists of a series of method calls. If each of these method calls is considered a step and locks are acquired and released at the level of these steps, then commutativity locking can happen in which these steps are interleaved for concurrent executions. In contrast to a single compatibility matrix of lock modes with only read and write operations, commutativity locking will use multiple commutativ- ity matrices with a lock mode for every method, and every class in the database schema will require a separate commutativity matrix. The traditional two-phase rules are used to govern the granting and the releasing of these locks.

2.3 Industrial Platform

The industry platform made available for our research to evaluate semantic locking is the IP-PBX switch IPES, made initially by Lucent Technologies, and later belonging to Avaya Inc. IPES is a distributed, server-based soft switch that allows a client organization to combine voice, data, fax, and internet communications over an existing IP network. It is one of the early examples of voice and data convergence solutions which are now becoming widespread in the telecommunications industry. IPES is an open standards- based system; it is a solution built from open (rather than proprietary) hardware and software components. It leverages open computing and communications standards to create a scalable switching and feature service platform. The IPES source code is written in Java. The part of the application that is of interest to our research is the section that accesses its embedded object database; in that section, there are 47 data classes defined (together representing the database schema) and 110 transaction types written to manipulate these data class objects in various ways.

2.4Commutativity Potential Analysis

In earlier work [19], we analyzed the commutativity relationship on all operation pairs of the 47 data classes defined in IPES. For each class, we examine every possible combination of method pairs (each called a conflict pair) to determine whether the pair of operations is commutative or not, and if so, why. There are read/ read (RR) pairs, pairs which access different fields in the same object (DF pairs), and semantic commutative (SC) pairs. This analysis was done entirely by hand, and was quite laborious. We considered parameters, return values and exception-throwing.

There is a direct correspondence between the classification of each pair of operations, and the behavior of the different concurrency control mechanisms. The most commonly implemented algorithm, object locking, allows concurrent transactions to

We found that out of all possible pairs of operations on the same object in the IPES classes, 29.1 percent are RR pairs, 62.1 percent are DF pairs, and 0.5 percent are SC pairs.

Fig. 1 illustrates the difference in concurrency potential among the three types of locking, class by class. The graph is a stacked column chart. In each class, the white bar represents the RR pairs, the gray bar represents the DF pairs, the black bar represents the SC pairs, and the gap beyond that up to 100 percent represents noncommutative pairs. So, the area up to the tip of the white bar shows the concurrency allowed under object locking, the area up to the tip of the grey bar shows the concurrency allowed under field locking, and the area up to the tip of the black bar shows the concurrency allowed under semantic locking.

It can be seen from the graph that there is hardly anything to separate the concurrency potential between semantic locking and field locking; the only noticeable differences are in classes 1, 2, 3, 7, 11, 18, and 43. However, the large number of DF pairs gives hope for greater performance when using semantic locking rather than object locking.

Further investigation shows that the few semantic commutative pairs in IPES are mainly from two situations:

- Set-collection maintenance. An operation that just manip ulates a set-valued attribute via add, remove, or removeall is commutative with itself.
- OID generator. Transactions that create new persistent objects generate a unique OID using a central OID.

This analysis of the commutativity potential in the IPES application considered the class definitions, but not the way calls of these classes are grouped into transactions. That is, the numbers and the graph in this section provide a static data-centric analysis of commutativity potential in the IPES data set. In order to see the actual impact on performance of semantic locking, we ran some experiments on the transactions that make up the IPES application, in a context where each concurrency control algorithm can be tried.

3. PERFORMANCE EVALUATION

We measure whether the extra concurrency potential translates into better performance with semantic locking for our real-world application, compared to object locking.

3.1Semantic Locking Layer

In order to provide a basis for experimental evaluation of the performance of the IPES application with semantic locking, we designed and developed a concurrency control layer that implements the algorithm. It is called the Commutative Concurrency Control Layer or C3Layer for short. The layer also implements object locking, field locking, and serial locking. The C3Layer serves as a middle layer between IPES and its embedded database PSE Pro. It performs various locking functions to manage the concurrent IPES transactions accessing the PSE Pro database. To do this, calls to the C3Layer are inserted into parts of the IPES source code; specifically, the data classes and the transaction code. The basic workings of the Lock Manager are similar to that of the generic lock manager described by Gray and Reuter [6], but adapted so that it supports semantic locking. Here, we sketch the key decisions; details are in [20].

The locking is done at the method level. The lock manager has a lock() call which is inserted at the beginning of every method of every data class. The lock() takes the parameters of the class ID, the instance ID, and the method ID to create a unique lock instance when the lock request is received. The difference between locking levels lies in which method pairs of an object are treated as conflicting. With object locking, only the RR pairs are nonconflicting and all other pairs conflict. With field locking, all RR and DF pairs are nonconflicting while the others are not. With commutativity locking, all RR, DF, and SC pairs are nonconflicting. The lock manager reads in all prebuilt commutativity matrices and stores them in multidimensional integer arrays. RR pairs are represented by 1, DF pairs by 2, SC pairs by 3, and non- commutative pairs by 9. The assigned locking level for a particular experiment run is also represented by an increasing number; 1 for object locking, 2 for field locking, and 3 for commutativity locking. Locks for two methods are nonconflicting with respect to the locking level assigned if the operation pair's numerical representation is equal to or less than the locking level's numerical semantically representation. Since the three concurrent locking levels will all incur the same overhead per lock (or unlock) operation, performance differences (if any) will reflect only the opportunity for generator.concurrent execution. With serial execution, no lock needs to be mutative instantiated when a lock request is received, and the coordination the calling As well as the transaction-duration locks, the C3layer provides short-term exclusion during execution of a data object's method through sync()/unsync() calls, and even shorter exclusion during iteration through a non-thread-safe PSE Pro collection by latch()/ unlatch() calls.

TABLE 1
Cluster 1 Mix Transaction Profiles

Name	LOC	Steps	Duration	Access Mode	Distribution Ratio
Transaction1	159	38	258ms	UPDATE	5%
Transaction3	127	15	1,515ms	UPDATE	5%
Transaction4	29	200	5ms	READONLY	5%
Transaction5	103	35	5ms	READONLY	45%
Transaction7	277	36	559ms	UPDATE	40%

A number of modifications are also added to the IPES transactions' source code to enable concurrency control by the C3layer. All clients are made part of a single PSE Pro session and transaction, to eliminate the built-in database level locking done by PSE Pro. Instead, calls to C3Layer's begin(), commit(), abort(), and inTransaction() are inserted in all IPES transactions. Also, deadlock handling code is added to all transactions.

3.2Experimental Design

We designed benchmarks which run mixes of the transactions found

in the IPES application code, in order to compare the performance of the four locking levels supported in the C3Layer. Like TPC-C [16], the benchmarks measure sustained throughput over a period of time. In a given transaction mix, there are several different transaction types. Each transaction type has a proportional frequency of appearance in the mix. We report results for two transaction mixes. Our benchmarks test throughput with from 1 to 100 client threads concurrently invoking transactions.

Each mix runs only four or five transaction types. The IPES transactions can be categorized into several clusters. Each cluster of transactions completes a logical business process, and typically the transactions perform work on a small number of target classes. For example, one cluster implements the user management, another is the device management. Transactions from different clusters often do not access data in common, so running them concurrently will yield few conflicts. Mix 1 is representative: It contains five transactions from the cluster corresponding to the most common business process. The other mix is a specially concocted combina- tion of transactions chosen from the application code to cause high contention involving SC operation pairs. This mix is designed to highlight the potential benefit of semantic locking over all other locking levels. The transaction mixes are called Cluster 1 Mix and Semantic Mix, respectively. The profiles of transactions in each mix are listed in Tables 1 and 2. For business confidentiality reasons, the original names of the IPES transaction are replaced by generic names like

TABLE 2
Semantic Mix Transaction Profiles

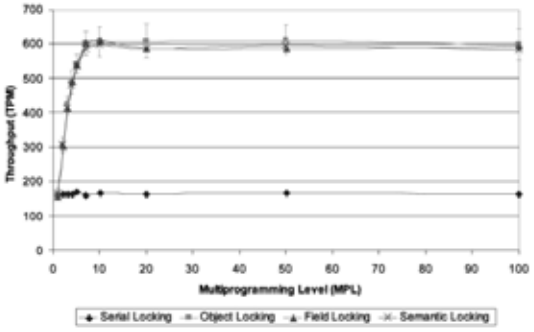
Name	LOC	Steps	Duration	Access Mode	Distribution Ratio
Transaction14	88	6	1,105ms	UPDATE	25%
Transaction22	84	14	153ms	UPDATE	25%
Transaction32	85	19	654ms	UPDATE	25%
Transaction44	162	42	1,828ms	UPDATE	25%

transaction5. LOC is the lines-of-code count of the transaction in the source code. Steps is the number of data class method calls in the code. Duration is the average time taken to execute the transaction in isolation. Access Mode shows whether the transaction has updates or is read only. Distribution Ratio shows its frequency within the mix; the ratios for Mix 1 reflect our sense of typical usage for this business process. Within each transaction mix, an experiment run is performed for every locking level and at varying multiprogramming levels. Each experiment has a ramp-up period of 2 minutes, and then measures each minute from the next 21 minutes, discarding one measurement if it is an extreme outlier (which did occur in several runs). The experiment platform is a desktop server, single CPU Intel Pentium 4 running at 2.533GHz, with 1GB of RAM, a single IDE hard drive. Details of the software configuration are in [20]. The database has just over 1 million objects but the common transactions of mix 1 operate on a hot spot of 100 objects out of 20,000 of the appropriate data class.

The performance metric of the experiments is the throughput TPM in transactions per minute. This figure is the average of all measurements recorded at the measurement intervals for the experiment run. MPL is the multiprogramming level: the number of concurrent clients (each issuing one transaction at a time). Other metrics were also recorded during the experiment, and details are in [20]. A key conclusion is that the C3layer overhead is low, usually under 5 percent of the original transaction execution time.

3.3 Cluster 1 Mix Results

Fig. 2 shows the throughput in Cluster 1 Mix against MPL, for each locking approach.



In this transaction mix, the baseline throughputs with serial execution are around 170 TPM and (as expected) they do not vary with MPL except for random fluctuations. All concurrency locking levels improve the throughput with increasing MPL until they reach a plateau around 600 TPM. There are no significant separations in performance among the concurrent locking levels: Fig. 2 shows the error bars at one standard deviation for object locking, and the range within these bars always includes the

Fig. 2. Cluster 1 Mix throughputs for the four levels of locking at various multiprogramming levels. Note the similarity in performance between the three concurrent locking levels.

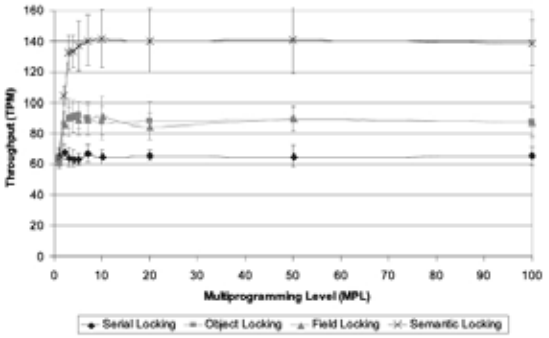


Fig. 3. Semantic mix throughputs for the 4 levels of locking at various multiprogramming levels. Note semantic locking outperforming the other two concurrent locking levels.

average throughputs of object and field locking levels (for better readability, we omitted the similar error bars of the other locking levels). We also measured the number of lock blocks per completed transaction, and found that for both object locking and field locking, the ratios fall between 0.01 to 0.11 across the MPL range. For semantic locking the ratios are slightly lower peaking at 0.04. Thus, there are not enough lock conflicts in this transaction mix to separate out the three concurrent locking levels.

In this mix, the probability of a semantic commutative conflict is very low. Semantic commutative operations of Class 1 occur in T1 and T5; operations of Class 3 occur in T3, and operations of Class 43 occur in T1. Since T1 and T3 are quite rare, any added concurrency potential between them does not help overall throughput greatly. The semantic commutative pairs between T1 and T5 are never invoked on the same object because of the way parameters are given to the two transactions (for example, transactions that create a new entity have a unique ID for the entity, and this ID can't be used in other transactions until the creation has committed). The only semantic commutative pairs that can be invoked by frequent transactions are where two threads are concurrently running T5 and both call methods of Class 1; here, the probability that the two will access the same object instance from the class is low (only 1 percent). Similarly, there are very few instances where concurrent transactions access the same object but on its different fields. Thus, there is nothing to separate the performances between object locking and field locking. In [20], we report on another natural cluster, confirming this pattern.

3.4 Semantic Mix Results

Fig. 3 shows the results for the contrived Semantic Mix. Serial execution is around 65 TPM, object locking and field locking gives 90 TPM, and semantic locking is 140 TPM. For this mix, there is a clear separation in performance between semantic locking and object/field locking. This has been achieved because transactions T22, T32, and T44 all access semantic commutative methods of a singleton object (the OID generator for the whole system) from Class 43. In these cases, the probability of accessing the same instance is 1! Thus, semantic locking has great potential to improve on more restrictive locking approaches. Blocking ratios for both object locking and field locking are between 0.43 and 0.75 across the MPL range. For semantic locking, the ratios are enormously lower at around 0.004 across the MPL range. Having semantic locking outperform object/field locking in this transaction mix shows that despite very low quantities of SC operation pairs in the

IPES data set, in situations where more of them come together in a

transaction mix then they can make a measurable impact to the semantic locking performance.

4 Conclusions

We have examined an application that is used in production in the telecommunication industry, to evaluate the impact on perfor- mance of the commutativity-based semantic locking algorithm, compared to other concurrency control mechanisms that do not need analysis of the application code. Because all the algorithms were implemented in the same way, we have a level playing field to see what benefit, in practice, would occur to justify the effort of analysis of the semantics.

We have shown that there are workloads possible in which the use of semantics does provide improved throughput. We found some transactions that access objects with methods that have semantic commutativity. When a workload was deliberately chosen to emphasize these situations, there is about a 60 percent increase in throughput with semantic locking, compared to other algorithms.

In future work, we hope to find a model that can predict the performance impact of semantic locking, from features of the code of the transactions, such as the number of commutative pairs accessed and the overall length of the transaction. This could be used at design time, to see whether the effort of using semantic locking will be justified.

The coding style in the IPES application is straightforward. Most methods are getters and setters for individual attributes, or else simple insertion or deletion from a collection. We would like to repeat our analysis for a real application with a more sophisticated object-oriented structure.

REFERENCES

[1] D. Agrawal and A.E. Abbadi, "A Non-Restrictive Concurrency Control for Object Oriented Databases," Proc. Int'l Conf. Extending Database Technology (EDBT '92), pp. 469-482, 1992. | [2] R. Agrawal, M.J. Carey, and M. Livny, "Concurrency Control Performance Modeling: Alternatives and Implications," ACM Trans. Database Systems, vol. 12, no. 4, pp. 609-654, 1987. | [3] B. Badrinath and K. Ramamritham, "Semantic-Based Concurrency Control: Beyond Commutativity," ACM Trans. Database Systems, vol. 17, no. 1, pp. 163-199, 1992. | [4] B.R. Badrinath and K. Ramamritham, "Performance Evaluation of Semantics-Based Multilevel Concurrency Control Protocols," Proc. ACM SIGMOD '90, pp. 163-172, 1990. | [5] M. Cart and J. Ferrie, "Integrating Concurrency Control into an ObjecOriented Database System," Proc. Int'l Conf. Extending Database Technology (EDBT '90), pp. 363-377, 1990. | [6] J. Gray and A. Reuter, Transaction Processing—Concepts and Techniques. Morgan Kaufmann Publishers, 1993. | [7] S. Jajodia and R. Mukkamala, "Reduction in Transaction Conflicts Using Semantic-Based Concurrency Control," Performance of Concurrency Control Mechanisms in Centralized Database Systems, pp. 387-405, Prentice-Hall, 1996. | [8] W. Jun and L. Gruenwald, "Semantic-Based Concurrency Control in Object-Oriented Databases," J. Object-Oriented Programming, vol. 10, no. 8, pp. 33-39, 1998. | [9] C. Lecluse, P. Richard, and F. Velez, "O2, An Object-Oriented Data Model," Proc. ACM SIGMOD '88, pp. 424-433, 1988. | [10] N. Lynch, M. Merritt, W. Weihl, and A. Fekete, Atomic Transactions. Morgan Kaufmann Publishers, 1994 | [11] C. Malta and J. Martinez, "Automating Fine Concurrency Control in Object- Oriented Databases," Proc. Int'l Conf. Data Eng. (ICDE '93), pp. 253-260, 1993. | [12] P. Muth, T. Rakow, G. Weikum, P. Brossler, and C. Hasse, "Semantic Concurrency Control in Object-Oriented Database Systems," Proc. Int'l Conf. Data Eng. (ICDE '93), pp. 233-242, 1993. | [13] R. Resende, D. Agrawal, and A.E. Abbadi, "Semantic Locking in Object- Oriented Database Systems," Proc. Conf. Object-Oriented Programming Systems Languages and Applications (OOPSLA '94), pp. 388-402, 1994. | [14] P.M. Schwarz and A.Z. Spector, "Synchronizing Shared Abstract Types," ACM Trans. Computer Systems, pp. 223-250, 1984. | [15] A.Thomasian, "Concurrency Control: Methods, Performance, and Analy- sis," ACM Computing Surveys, vol. 30, no. 1, pp. 70-119, 1998. | [16] TPC Benchmark C Standard Specification, Transaction Processing Perfor- mance Council, Revision 5.2, http://www.tpc.org/tpcc/, 2003. | [17] W. Weihl, "Commutativity-Based Concurrency Control for Abstract Data Types," IEEE Trans. Computers, vol. 37, no. 12, pp. 1488-1505, Dec. 1988. | [18] G. Weikum and G. Vossen, Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann Publishers, 2001. | [19] P. Wu and A. Fekete, "An Empirical Study of Commutativity in Application Code," Proc. Int'l Database Eng. and Applications Symp. (IDEAS '03), pp. 358-369, 2003. | [20] P. Wu, A. Fekete, and U. Rohm, "The Efficicacy of Commutativity-Based ¨ Semantic Locking in Real-World Application," Technical Report 612, School of Information Technologies, Univ. of Sydney, Aug. 2007. | [21] S. Xavier and K. Ramamritham, "Implementation Considerations and Performance Evaluation of Object-Based Concurrency Control Protocols," Performance of Concurrency Control Mechanisms in Centralized Database Systems, pp. 356-386, Prentice-Hall, 1996 |