SAML: A Self-Adjusting Multi-Granularity Locking Protocol For Object-oriented databases

Deepa Saha, Joan Morrissey
School of Computer Science
University of Windsor
Canada
sahag@uwindsor.ca, joan@uwindsor.ca

ABSTRACT: *Object-oriented databases (OODBs) have the potential to be used for data-intensive, multi-user applications that are not well served by traditional databases. Despite the fact that there has been extensive research done for relational databases in the area of concurrency control, many of the approaches are not suitable for the complex data model of object-oriented databases. This paper presents a self-adjusting multi-granularity locking protocol (SAML) which facilitates choosing an appropriate locking granule according to the requirements of the transactions and encompasses less overhead and provides better concurrency compared to some of the existing protocols. Though there has been another adaptive multi-granularity protocol called AMGL [Chang, C.T.K. 2002], which provides the same degree of concurrency as SAML: SAML has been proven to have significantly reduced the number of locks and hence the locking overhead compared to AMGL. Experimental results show that SAML performs the best when the workload is high in the system and transactions are long-lived.*

Keywords: Object-Oriented systems, Concurrency control, Locking, Performance of locking protocols

Received: 21 December 2009, Revised 29 January 2010, Accepted 1 March 2010

© D-LINE. All rights reserved

1. Introduction

OODBs are used for data-intensive, multi-user, object-oriented applications that are difficult to do in traditional databases. OODBs face severe performance challenges with these applications as they have complicated requirements. Their transactions are usually long-lived and perform extensive computations on a large number of interrelated objects [Chang, C.T.K. 2002].

Concurrency control mechanisms resolve conflicts when multiple transactions are trying to access the same data. It is a factor for measuring the performance of any database and a significant research issue for OOBDs. There has been little research on concurrency control mechanisms for OODBs. Many of the traditional relational concurrency control methods can be applied to OODBs. However, they are not suitable for the long-duration transactions typical of OODBs. This is the main issue addressed in our paper.

Locking is the most commonly used concurrency control mechanism. However, in object-oriented applications, a large number of objects can be involved and the overhead is significant when locks are used. Applications may operate on objects for a long time, be interactive and have long lived transactions. Poor performance can occur if the wrong locking granularity is chosen. The locking granularity is the size of the object to be locked.

Multi-granularity locking [Grey et al. 1998] aims to reduce the locking overhead in relational databases. The database is organized into a hierarchy and locking at a high level entails locking all lower levels. In earlier OODBs, for example, ORION and O2 [Kim, W. 1990] the database hierarchy is replaced with simple class and object hierarchies. The protocols used strict two phase locking [Kim, W. 1990]. However, the rich semantics of OODBs [Rao, B.R. 1994] are lost with this method. Taniguchi *et al.* [Taniguchi et. all. 1995] did a performance evaluation of three multi-granularity locking mechanisms: Non-class hierarchy locking, class granularity locking, and class hierarchy locking. They found that class granularity locking usually reduces the locking overhead. However, they have chosen a static strategy to predetermine the locking granularity. A static strategy means that the locking granularity cannot be changed once the transaction starts. [Lee et. all. 1996] proposed a multi-granularity locking method which includes some features of OODBs, for example, the class hierarchy, a composite object hierarchy and the schema evaluation during locking. They tried to accommodate the concept of composite objects as a logical lockable granule. They used two different granules for locking: Schema lock and Instance lock. They proved that their multi-granularity protocol for OODBs gives improved concurrency than the protocol used in ORION [Kim, W. 1990].

Chang [Chang, C.T.K. 2002] proposed the Adaptive Multi-Granularity Locking (AMGL) protocol for OODBs. It uses different locking granularity units, namely, instance, class and class hierarchy. All are run dynamically. A dynamic strategy means that the locking granularity can be adjusted during the lifetime of the transaction. Their method reduces the locking overhead in some cases when compared to several typical locking protocols. Its performance is better when the number of locking conflicts is low. However, when examining the locking overhead, it typically performs worse than instance granularity locking. Hence, this protocol can not effectively address the trade-off between the locking overhead and the degree of concurrency.

In order to solve this problem, we propose an improved adaptive multi-granularity protocol, SAML. Since no application specific information is needed for this protocol, it can work in any OODB. SAML is designed to improve the degree of concurrency of OODBs while alleviating certain locking overhead issues that frequently occur in OODB specific applications.

2. Design and Development of SAML

In a database, the finer the granularity the more parallelism is possible. However, this will result in a higher locking overhead. Conversely, a coarse granularity unit means fewer locks and less overhead but we get less concurrency as active transactions may hold locks on more resources than they need. Hence, there is an unavoidable trade-off between increasing concurrency and decreasing the locking overhead. We propose that the database management system should provide a range of granularity units depending on the requirements of different transactions during their run times.

Our SAML protocol aims to maximize concurrency while keeping the locking overhead at a minimum. It chooses an appropriate locking granularity based on the needs of a transaction. It uses the semantics of OODBs to decide on the different granularities. It uses three levels of granularity for locking, specifically: instance, class and class hierarchy locks. SAML first locks the coarsest granularity unit and keeps reducing the granule size whenever a conflict occurs. Thus, SAML enables the lockable granules to be adjusted automatically while a transaction runs. SAML is an optimistic locking protocol and uses conservative two-phase locking to avoid deadlocks.

2.1 Explicit and Implicit Locks

SAML proposes a number of lock modes for our three granularity units. In addition, two types of locks, explicit and implicit, are used. An explicit lock is one that is requested by and is granted to a transaction. Implicit locks are those needed, in either direction in the database hierarchy, for explicit locks. In SAML, the term 'implicit lock' denotes a lock which propagates down the hierarchy due to an explicit lock on a specific node. This implies that the modes of the implicit locks will be the same as those of the explicit lock.

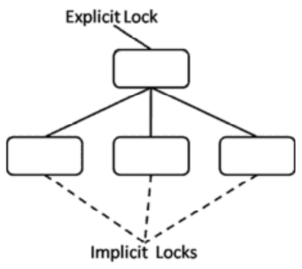

Figure 1. Explicit and Implicit Locks

2.2 Soft Locks

In SAML, a soft lock at any level of the hierarchy means that an explicit lock exists some place down the hierarchy. In OODBs, to lock any resource means locking all its ancestors so that no transaction can modify them. If any node is explicitly locked, soft locks are implicitly applied on the ancestors to prevent the granting of exclusive lock requests on them by other transactions.

2.3 Lock Modes in SAML

Based on the granularity units used in SAML, the lock modes have been characterized into three different groups: Instance, class and class hierarchy locks. All lock modes are either shared, exclusive or variations of these two lock modes.

Transactions deal with a set of instances of one or more classes. Instances are the smallest granularity unit used. In SAML an instance can be locked either in shared (IS) or exclusive (IX) mode. A shared instance means it can be read by one or more transactions but cannot be modified by any of them. An exclusive lock on an instance means it can be read and modified only by the transaction which has acquired the lock.

Class locks are the next highest granule for locking. A class can have several instances. A class can be locked in shared (CS), exclusive (CX), soft shared (CST) or soft exclusive (CXT) mode. A CS or CX lock on a class implies all the instances of the class are implicitly locked in IS or IX mode. A CST or CXT lock on a class implies one or more instances of the class are locked in IS or in IS/IX mode, with at least one IX lock.

Locking a class in a hierarchy mode implies that all its instances, all its descendents and their instances, are locked in the same mode. We have four kinds of hierarchy lock modes: Shared Hierarchy lock (HS), Exclusive Hierarchy lock (HX), Soft Shared Hierarchy lock (HST) and Soft Exclusive Hierarchy lock (HXT). A HS or HX lock on a class means that it, its subclasses and all the instances of those classes are implicitly locked in shared or exclusive mode. A HST lock on a class means one or more descendent classes of it, or the class itself, is locked in CS, CST or HS (only for descendent classes) mode by one or more transactions. Other transactions can lock the entire hierarchy, or any part of it, only in shared mode. A HXT lock on a class means one or more descendent classes of it, or the class itself, is locked in CS, CX, CST, CXT mode or HS/HX mode (only for descendent classes) with at least one lock in exclusive mode, by one or more transactions. No transaction can lock the entire hierarchy. However, any part of it can be locked by other transactions in shared or exclusive mode (IS/IX, CS/CX, or HS/HX) only if they do not conflict with any of the existing locks.

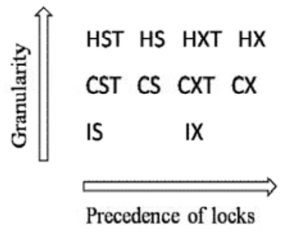

Figure 2. Precedence and Coverage of Lock Modes

In SAML, an IS lock is contained in a CS or CST lock while a CS or CST lock is contained in a HS or HST lock. Again, at the same granularity level, an exclusive lock has a higher precedence over a shared lock and an explicit lock has a higher precedence over a soft lock. A lock L1 is said to be covered by another lock L2 if L2 has higher granularity than L1 or if L2 has a higher precedence over L1 where both L1 and L2 have the same granularity level.

2.4 Lock Compatibility Matrix

Two locks of the same or different modes on one object can be granted concurrently if they are compatible according to their meaning. The compatibility matrix for our proposed lock modes is shown in Table 1. In SAML, a transaction can only request explicit locks. The soft locks are implicitly applied. So, the new lock mode being requested can only be one of the six explicit locks. Each cell in the compatibility matrix states whether a new lock mode is compatible with the existing one. A “Y” means that a lock will be granted. An “N” means we have a conflict and a lock will not be granted. A “D” denotes de-escalation of the new lock. That is, we go to a finer granularity. A “DD” means de-escalation of both the existing and requested lock.

		New Lock Modes					
		IS	**IX**	**CS**	**CX**	**HS**	**HX**
Existing Lock Modes	IS	Y	N				
	IX	N	N				
	CST			Y	D	Y	D
	CS			Y	DD	Y	DD
	CXT			D	D	D	D
	CX			DD	DD	DD	DD
	HST			Y	Y	Y	D
	HS			Y	DD	Y	DD
	HXT			Y	Y	D	D
	HX			DD	DD	DD	DD

Table 1. Lock Compatibility Matrix

2.5 Main Components of SAML

SAML uses a lock table and a class inheritance graph (CIG) of the database to maintain all locks. The lock table holds the explicit locks. The CIG is used to store the soft locks. Using a CIG, SAML does not have to retain any information about soft locks and the size of the lock table and locking overhead is much smaller than in other protocols. The benefit of using a CIG is that it remains static unless the schema changes so the maintenance overhead is insignificant.

Class Inheritance Graph (CIG): The CIG is used to track the soft locks as they propagate recursively upwards in the class hierarchy to the root. The graph allows navigation from a class to its subclasses or super classes.

Figure 3 is an example of a CIG along with some locks on objects. Each node in a CIG stands for a class and each node has two colors: Class and hierarchy color. The term "color" is used to indicate the lock mode on the node. In Figure 3 the shaded half of a node represents the hierarchy color, the other half is the class color. As shown, class C2 has a CS lock on itself. One subclass of C2 has a HS and the other one has a HX lock. Because at least one of C2's descendents has an exclusive lock, C2 as well as all its super classes must have HXT locks. Class C4 is locked in CS mode and its only subclass is locked in HS mode. Since there is no exclusive lock in this hierarchy, the hierarchy field of C4 will be colored as HST.

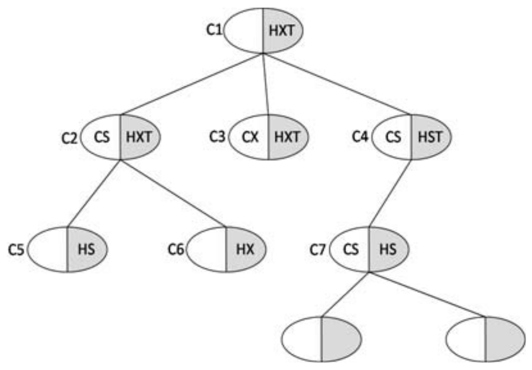

Figure 3. Class Inheritance Graph (CIG)

Lock Table: In SAML, the lock table contains the list of explicit locks present in the system. Each lock entry there is a Lock ID, a Transaction ID, the set of target resources along with the intended lock mode, the current resource locked and the mode of the lock.

Transaction List: In SAML, this is a list of the transactions that are currently in the system. Each entry of the list has a Transaction ID, the set of locks held by the transaction, the set of instances to be locked and the lock mode.

2.6 Rules for Requesting Locks

When locking any node, SAML updates all the nodes' color along the path to the CIG root recursively using the rules described here.

Before locking an instance with an IS or IX lock, we search the lock table for conflicting locks and the corresponding class node in the CIG must be colored as CST or CXT.

Before locking a class in CS or CX mode, we check node's color in the CIG for conflicting locks. Then we color the node as CS or CX, if it is not already colored or it is colored as CST (for CS only). To color a node as CS or CST the super class node must be colored as HST. To color a node as CX or CXT the super class node must be colored as HXT.

Before we lock a class hierarchy in HS or HX mode, we check the node's color in the CIG for conflicting locks. Then we color the node as HS or HX, if it is not already colored or is colored as HST (for HS only). To color a node in HS or HST the super class node must be colored as HST. To color a node in HX or HXT the super class node must be colored as HXT.

2.7 Lock Release

When a transaction commits or aborts, all the locks held are released by removing them from the lock table. Then we search the updated lock table to check if there is any explicit lock on the object. If there is, then the corresponding node color in the CIG remains unchanged or else the color of all the children of the current node are checked and the current node color is updated accordingly. Finally, the current node's super class is re-colored.

2.8 Lock De-escalation

In any locking protocol, the number of objects being locked, the object type and the lock coverage can affect the lock maintenance overhead. SAML tries to minimize the number of locks in use. Most protocols choose a locking granularity statically. If a protocol can use the finest granularity unit, then the maximum degree of concurrency is obtained. However, the lock maintenance overhead will be very high. With a coarser granularity lock maintenance is reduced. In the case of static protocols, if a better granularity is available as a transaction runs the level of concurrency cannot be increased. So the granularity unit may not be optimal. To solve this problem, we use dynamic lock de-escalation. It picks the appropriate locking granularity for a transaction as it runs. The largest granularity unit is used first even if most of the objects are not required. However, SAML keeps track of the objects being accessed and the modes employed. This information is used for de-escalation. Later, if there is a conflict, lock de-escalation is performed. Each conflicting transaction then locks at the next higher granularity unit that is not in conflict. So when the locks of two transactions conflict both transactions keep on reducing their granule size until the conflict is resolved.

SAML uses two types of lock de-escalation: Class lock de-escalation and hierarchy lock de-escalation. A hierarchy lock on a class after de-escalation can generate either a class lock on itself or hierarchy locks on the sub classes of it. A hierarchy lock after de-escalation can generate either a single class lock, a set of hierarchy locks or a set consisting of both class locks and hierarchy locks. A class lock on a class after de-escalation can generate one or more instance locks on the instances of the class.

2.9 Simulation Overview

The database model we have used in our simulation is similar to the one used in AMGL proposed by Chang [Chang, C.T.K. 2002]. The simulation model is constructed by retaining only the parts of an OODB which are relevant for our simulation.

Java's multi-threading feature has been used to simulation concurrent processing of multiple transactions. Each component of the simulation environment, for example, the concurrency control manager, the lock manager and the transaction manager run concurrently in separate threads during the simulation process. In addition, each active transaction runs in a separate thread. We have assumed that there is no schema change and no multiple inheritances in the database.

Transactions are generated offline by a transaction generator prior to the simulation. During the simulation, a transaction injector injects the transactions into the system using a Poisson distribution [Kim, W. 1990], with mean rate of ten transactions per time unit. Transactions are placed at the end of the ready queue. The concurrency control protocol takes one transaction at a time from the front of the queue and requests a lock for the transaction. If the request is granted, then the transaction is put into the active transaction list and a new thread is created for the transaction which remains alive during its lifetime. If the lock request is not granted then the transaction goes to the waiting transaction list. The conflicting transaction in the active transaction list is updated with this information. When a transaction ends, it is removed from the active transaction list. If there is another transaction waiting for the completion of the finished transaction, it is removed from the waiting transaction list and put at the front of the ready queue. Figure 4 is a pictorial representation of the simulation model discussed.

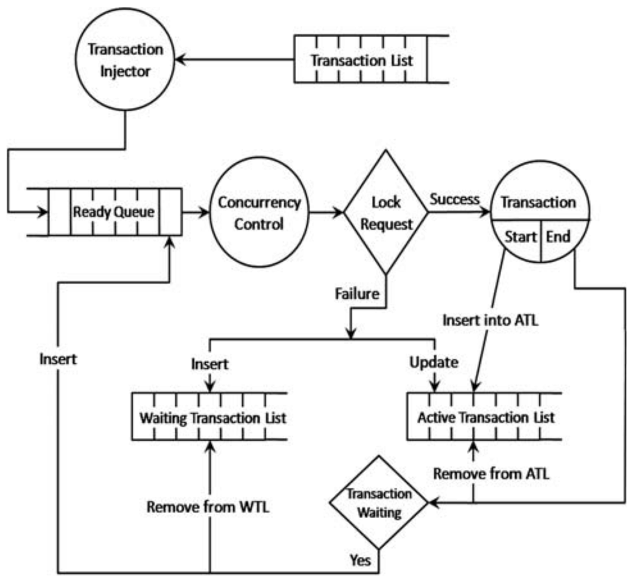

Figure 4. Simulation Model

3. Experiments, Results and Analysis

SAML has been compared experimentally with Instance Granularity Locking (ILP) and Class Granularity Locking (CLP) and analytically with AMGL [Chang, C.T.K. 2002] to show how SAML optimizes the trade-off between the degree of concurrency and the locking overhead. Instance granularity locking uses instances as the lock granule and class granularity locking uses classes. However, SAML uses instance, class or class hierarchy depending on the requirements of the transactions. With the exception of AMGL and SAML, ILP offers a higher degree of concurrency than existing locking protocols but it has a high locking overhead. Similarly, CLP has very low locking overhead but it provides low concurrency.

SAML and AMGL have completely different locking granularity methods compared to CLP and ILP. At any given instance, it is impossible to say how many instance, class and hierarchy locks are present in the system while using SAML or AMGL. It depends on the transaction's workload, duration, access area, write ratio and the rate of transactions coming into the system. Therefore, SAML was compared experimentally with ILP and CLP and compared analytically with AMGL. The SAML protocol is an improved version of the AMGL protocol. The key difference is in the way the two protocols handle soft locks. Therefore, only an analytical comparison between them can properly compare their respective performances.

In the simulation experiments, three criteria have been measured to evaluate the performance of each protocol.

Lock Count (LC) is the size of the lock table at any instance. It represents the number of explicit locks on various resources in the database.

Active Transactions (AT) is the number of transactions running in the system.

Waiting Transactions (WT) is the number of transactions that are waiting because of the presence of conflicting transactions.

3.1 Values of Simulation Parameters

For experimental purposes, three kinds of databases have been used: Type-1 has a lot of instances but less subclasses per class; Type-2 is similar but the hierarchy depth is double that of Type-1; Type-3 has the same depth as Type 1, but has more subclasses per class and less instances per class.

Database Parameters	**Database Type**		
	Type 1	Type 2	Type 3
Number of Subclasses per Class	3	3	10
Number of Instances per Class	50	50	15
Depth of Class Hierarchy	5	10	5

Table 2. Database Generation Parameters

For simulation purposes one unit time is one hundred milliseconds. The transactions used for the experiments can be classified into two categories depending on the number of instances they work with: a heavy load or a small load. Heavy load transactions consist of two hundred instances whereas small load transactions use twenty. Each experiment ran four hundred transactions. Each of these two types has been tested for two different durations: Short and long. Short duration transactions exist for two units of time, long for four units. These transactions with different loads and durations can have three types of access areas in the database. This signifies the area of database from where the instances, with which the transactions work with, have been chosen: Concentrated near to the root; near to leaf or the entire database.

3.2 Experiments

Depending on the types of the databases used, the experiments performed can be categorized into three major groups. For each type of database, transactions of different workloads and various durations have been run on three different access areas of the database. T_1, T_2 and T_3 are the three types of databases used. A_R, A_L and A_O denote the access areas: Near the root, near the leaf or the entire database respectively. L_S and L_H stand for small load and heavy load. D_1, D_2 and D_4 represent the different durations of transactions. D_1 uses one unit of time as the duration of transaction. D_2 and D_4 use two and four units of time respectively. Therefore, an experiment denoted by $T_3A_OL_HD_4$ means that the experiment has been done on a Type-3 database with access to the entire database for transactions with heavy load and four units of duration.

Type-1 Database: Our Type-1 database has very few classes. Each class has three sub-classes and fifty instances. The depth of the class hierarchy is five. Therefore, the total number of classes in the database is 121. Hence, the degree of conflict among transactions is very high.

	Lock Count (Avg)			**Active Transaction(Avg)**			**Waiting Transaction (Avg)**		
	SAML	CLP	ILP	SAML	CLP	ILP	SAML	CLP	ILP
$T_1A_OL_SD_2$	364	58	367	19	7	18	7	126	7
$T_1A_OL_SD_4$	656	61	664	33	7	33	27	160	26
$T_1A_OL_HD_2$	454	17	454	2	1	3	154	166	165
$T_1A_OL_HD_4$	460	18	455	2	1	3	169	180	186
$T_1A_LL_SD_2$	362	58	369	19	7	18	7	119	7
$T_1A_LL_SD_4$	648	62	648	33	7	33	30	155	30
$T_1A_LL_HD_2$	451	16	438	2	1	2	151	173	169
$T_1A_LL_HD_4$	453	18	457	2	1	3	166	178	186
$T_1A_RL_SD_2$	365	57	372	19	7	19	7	122	7
$T_1A_RL_SD_4$	670	61	670	34	7	34	28	156	28
$T_1A_RL_HD_2$	460	18	432	2	1	2	157	164	166
$T_1A_RL_HD_4$	449	18	457	2	1	3	160	174	185

Table 3. Experimental Results for the Type-1 Database

Our experiments show that the performance of SAML is the same as IPL in terms of lock count, active transactions and waiting transactions. Because of the high rate of conflicts among the transactions, SAML has to de-escalate almost all the locks to the instance level in order to enable more concurrent transactions However, in the CLP protocol each transaction holds locks on a good portion of the database until it completes, which means low concurrency. The locking overhead of CLP is very low compared to SAML or ILP.

Our experiments show that in a Type-1 database, the total time required by the CPL to complete all the four hundred transactions is always three to five times the time required by ILP or SAML.

Although ILP and SAML have the same locking overhead and the same number of active and waiting transactions, SAML has an additional overhead for the lock de-escalations and maintaining the class inheritance graph. We conclude that for Type-1 databases ILP gives the best concurrency control and that CLP gives the worst.

Type-2 Database: Here the class hierarchy is twice as deep as in Type 1. Therefore, it has a larger number of classes and hence less conflicts among the transactions.

	Lock Count (Avg)			**Active Transaction(Avg)**			**Waiting Transaction (Avg)**		
	SAML	CLP	ILP	SAML	CLP	ILP	SAML	CLP	ILP
$T_1A_OL_SD_2$	174	169	383	19	19	19	<1	<1	<1
$T_1A_OL_SD_4$	346	329	741	37	37	37	<1	1	<1
$T_1A_OL_HD_2$	707	493	3809	19	18	19	2	5	2
$T_1A_OL_HD_4$	1685	920	4374	37	34	37	6	19	6
$T_1A_LL_SD_2$	170	167	382	19	19	19	<1	<1	<1
$T_1A_LL_SD_4$	344	328	744	38	37	37	1	1	1
$T_1A_LL_HD_2$	725	502	3788	19	19	19	3	5	3
$T_1A_LL_HD_4$	1763	920	7330	37	34	37	7	20	7
$T_1A_RL_SD_2$	172	169	382	19	19	19	<1	<1	<1
$T_1A_RL_SD_4$	343	329	742	37	36	37	1	1	1
$T_1A_RL_HD_2$	706	496	3774	19	19	20	2	4	2
$T_1A_RL_HD_4$	1681	935	7331	37	35	37	6	18	6

Table 4. Experimental Results for the Type-2 Database

Table 4 presents our experimental results. Here, for a small workload the locking overhead of SAML is similar to CLP and much lower than ILP. SAML has the same degree of concurrency as ILP. For a heavy workload the number of locks held by SAML is slightly more than CLP but much lower than ILP. SAML has same degree of concurrency as ILP since both protocols have almost the same number of active and waiting transactions. The CLP protocol provides less concurrency than both SAML and ILP. It has less active transactions and more waiting transactions at any point in time. Hence, it takes longer to complete all the transactions, especially for long, heavy workload transactions.

Although both SAML and ILP provide the same degree of concurrency, the locking overhead of SAML is significantly less than ILP despite the additional overhead of maintaining the class inheritance graph. Therefore, for this type of database, SAML provides better concurrency control with a heavy workload. If the workload is light then CPL performs better.

Type-3 Database: Our Type-3 database has more classes and less instances per class. Therefore, using SAML, each lock blocks a very small portion of the database. However, CLP provides better concurrency in this case.

Table 5 presents the experimental results of the three protocols on various access areas of a Type-3 database for transactions with different workloads and durations.

For transactions with a small workload all the three provide a similar degree of concurrency since the conflicts among the transactions is very low. The ILP protocol has a high locking overhead, whereas the CLP and SAML protocols have a similar

locking overhead. However, it is much less than ILP. We conclude that for small load transactions CLP works best because SAML has the additional overhead of lock de-escalations.

Transactions with a heavy workload and of long duration have many more conflicts. Therefore, CLP provides less concurrency and takes longer to finish all transactions when compared to ILP or SAML. The locking overhead of SAML in this case is higher than in CLP. However, it is much less than ILP. The degree of concurrency in SAML is very close to ILP. Therefore, considering the concurrency and the locking overhead, we conclude that SAML works best for Type-3 databases with a heavy workload.

	Lock Count (Avg)			**Active Transaction(Avg)**			**Waiting Transaction (Avg)**		
	SAML	CLP	ILP	SAML	CLP	ILP	SAML	CLP	ILP
$T_1A_OL_SD_2$	180	173	381	19	19	19	<1	1	<1
$T_1A_OL_SD_4$	358	329	740	37	36	37	1	4	1
$T_1A_OL_HD_2$	1900	657	3419	17	11	17	27	90	26
$T_1A_OL_HD_4$	2661	749	4794	21	13	24	53	142	82
$T_1A_LL_SD_2$	180	175	382	19	19	19	<1	1	<1
$T_1A_LL_SD_4$	361	342	747	37	40	37	1	4	1
$T_1A_LL_HD_2$	1862	667	3410	16	11	17	28	88	29
$T_1A_LL_HD_4$	2770	751	4725	21	13	24	59	138	85
$T_1A_RL_SD_2$	180	175	385	19	19	19	<1	1	<1
$T_1A_RL_SD_4$	364	341	744	37	37	37	1	5	1
$T_1A_RL_HD_2$	1801	356	3384	16	11	17	29	90	29
$T_1A_RL_HD_4$	2725	735	4677	21	12	24	53	138	84

Table 5. Experimental Results for the Type-3 Database

3.3 Experimental Summary

Table 6 lists the best and the worst performances of the SAML, ILP and CLP protocols on different types of databases for transactions of differing workloads. We also considered the locking overhead and the degree of concurrency. We found that it is the workload of the transaction which significantly affects the concurrency and locking overhead of the protocols. For Type-1 databases, SAML and ILP performed similarly in terms of locking overhead and the number of waiting and active transactions. However, SAML has more overhead due to lock de-escalations and maintaining the CIG graph. We conclude that for Type-1 databases ILP works best. For Type-2 and Type-3 databases the SAML protocol performs best with a heavy workload whereas the CLP protocol performed well with a small workload. In every case, the performance of SAML was never the worst.

Database type	Workload	Best	Worst
Type 1	Small	ILP	CLP
	Heavy	ILP	CLP
Type 2	Small	CLP	ILP
	Heavy	SAML	ILP
Type 3	Small	CLP	ILP
	Heavy	SAML	CLP

Table 6. Best and Worst Performance

3.4 SAML Compared to AMGL

Both SAML and AMGL provide the same degree of concurrency. Both protocols require exactly the same number of lock de-escalations to lock any object in the database. However, they differ in time and space complexity.

To lock any class or instance at level *l*, the AMGL protocol needs (*l*–1) explicit soft locks along with the explicit firm locks. In SAML to lock any class or instance we need only to change the color of the corresponding node in the CIG (only for class or hierarchy locks) and update the color of the immediate ancestor node. SAML requires only one entry in the lock table for any lock. This significantly reduces the size of the lock table and the cost of maintaining it. For example, if *n* different transactions simultaneously lock an object *o* at level *l* of the class hierarchy, then AMGL requires $n \times (l–1)$ soft locks and *n* firm locks. The total number of locks for AMGL will be $n \times l$, which is directly proportional to the level of the class in the hierarchy. Thus, the number of entries in the lock table increases exponentially with the depth of the hierarchy. SAML is independent of the level of the object in the hierarchy. Therefore, the number of locks required by SAML is *n*, which is linear and therefore significantly less. The space complexity is the main parameter to compare here. The larger the lock table size, the more the space and time complexity will increase. The space complexity for AMGL is $O(n \times l)$. For SAML it is $O(n)$, which is linear and therefore an improvement. This shows that SAML will have less locking overhead than AMGL in all cases. This is a important improvement over AMGL.

4. Conclusion and Future Work

This paper proposes an improved adaptive multi-granularity locking protocol, SAML, which allows an appropriate locking granularity to be chosen according to the requirements of a transaction. This optimistic locking protocol uses conservative two-phase locking to prevent deadlocks. It maximizes the concurrency of transactions while keeping the locking overhead as low as possible. By using a CIG to maintain soft locks, it is capable of greatly reducing the locking overhead in comparison to AMGL. SAML works best when the workload is high in the system and transactions are long-lived and thus is of greater importance and use in object-oriented applications.

SAML assumes single inheritance in the system and does not consider composite objects separately from primitive objects. Furthermore, we have not considered the effects of any schema modifications in SAML. These issues can be addressed in future work. We have assumed that there will be no schema modification of the database. But in reality, that might not be the case always. So, including the effects of schema modification in SAML protocol could be an interesting area to explore.

5. Acknowledgements

This research was entirely funded by an NSERC grant to Dr. Joan Morrissey.

References

[1] Kim, W. (1990). Introduction to object-oriented databases. MIT Press.
[2] Chang, C.T.K. (2002). Adaptive multi-granularity locking protocol in object-oriented databases, PhD thesis, University of Illinois.
[3] Gray, J.N., Lorie, R.A., Putzolu, G.R., Traiger, I. L. (1998). Granularity of locks and degrees of consistency in [3] a

shared database. Morgan Kaufmann Publishers Inc.
[4] Rao, B.R. (1994). Object-Oriented Databases: Technology, Applications, and Products. McGraw-Hill Companies.
[5] Taniguchi, S., Budiarto, Shojiro, N. (1995). On locking protocols in object-oriented database systems. *IEICE transactions on information and systems*, E78-D (11) 1449-1457.
[6] Lee, S.Y., Liou, R.L. (1996). A multi-granularity locking model for concurrency control in object-oriented database systems. *IEEE Transactions on Knowledge and Data Engineering*, 8 (1) 144-156.
[7] Atkinson, M.E. (1989). The Object-Oriented Database Manifesto. *In:* Proceedings of First International Conference on Deductive and Object-Oriented Databases, Koyto, p. 223-240.

Authors Biographies

Deepa Saha is pursuing her Ph.D. under Dr. J. Morrissey. Her research interests are transaction management, information retrieval.

Joan Morrissey received her Ph.D. from University College Dublin, Ireland. Her research interests include distributed query optimization, transaction management and certain aspects of information retrieval.