ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.2, February 2007, pp.345−350
DOI: 10.1360/jos180345

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

多级多版本数据库管理系统全局串行化*

张 敏[1,2+], 冯登国[1,2], 徐 震[1,2]

[1](中国科学院 软件研究所 信息安全国家重点实验室,北京 100080)

[2](中国科学院 研究生院,北京 100049)

Global Timestamp Serialization in Multi-Level Multi-Version DBMS

ZHANG Min[1,2+], FENG Deng-Guo[1,2], XU Zhen[1,2]

[1](State Key Laboratory of Information Security, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate School, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: Phn: +86-10-82612797, Fax: +86-10-62520469, E-mail: mzhang@is.iscas.ac.cn

Zhang M, Feng DG, Xu Z. Global timestamp serialization in multi-level multi-version DBMS. *Journal of Software*, 2007,18(2):345−350. http://www.jos.org.cn/1000-9825/18/345.htm

Abstract: The concurrency control mechanism in the multi-level DBMS is required to promise the serializability of transactions and the multi-level security properties, avoid possible covert channels and the starving problem of high-level transactions. The multi-level multi-version timestamp ordering mechanisms satisfy these requirements but may cause transactions read old version data, and the scheduler is required to be a trusted process. This paper presents a multi-level multi-version global timestamp ordering (MLS_MVGTO) mechanism and the basic global timestamps generation steps based on the transaction's snapshot. This paper also presents two improvements according to the pre-knowledge of the read-only transactions. In addition it can be implemented as a set of untrusted schedulers. Given the pre-knowledge about transactions' operations, transactions are able to read more recent version.

Key words: global timestamp; MLS_MVGTO; multi-version; 1 copy serializable (1SR)

摘 要: 多级调度应该保证事务历史可串行化,满足多级安全特性,不会引入隐通道,并保证高级别事务不会因为无限等待而"饿死".与其他多级数据管理系统调度机制相比,多级多版本时戳调度机制满足上述要求,但该机制存在两个问题,一是事务可能读旧版本,二是要求调度器是可信进程.提出一种多级多版本全局时戳调度机制(MLS_MVGTO),以及依据事务快照生成其全局时戳的基本步骤.给出了预知只读事务信息时的两种改进方法.MLS_MVGTO 机制生成的事务历史可串行化,不引入隐通道等,并且该方法避免引入一个全局可信的调度器,并通过对只读事务的深入分析,允许事务读新版本.

关键词: 全局时戳;多级多版本全局时戳排序 MLS_MVGTO;多版本;单版本可串行化(1SR)

中图法分类号: TP309 文献标识码: A

* Supported by the National Natural Science Foundation of China under Grant Nos.60273027, 60025205 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2004AA147070 (国家高技术研究发展计划(863))

Received 2005-01-26; Accepted 2005-09-05

In a multi-level DBMS, the concurrency control mechanisms are required to promise not only the serializability of transactions, but also the multi-level security properties. In addition, the scheduler should avoid possible covert channels and the starving problem of high-level transactions. Considering the centralized architecture in a single version DB, it is comparably difficult to reconcile the conflict of the serializability, the multi-level security, and the fairness. For instance, the solution given in Ref.[1] may cause the unbounded lock of high-level transactions; The solution in Ref.[2] partially sacrifices the serializability of high-level transactions. The approaches in Ref.[3] prevent the conflict by adding extra limitations to transactions, such as the maximum numbers of the writing transactions and the fixed delay writing time for each transaction, etc.

The concurrency control mechanisms in a multi-version DB is more flexible compared with those in a single version DB. Unlike multi-version locking mechanism[4,5], multi-level multi-version timestamp ordering mechanism[6−8], which is extended from the traditional multi-version timestamp ordering mechanism[9], avoids the dilemma in a single version database by enabling transactions to read different versions of data. However, as the price of reconciliation, transactions may have to read the old version data. Another common problem of these approaches is that all transactions are scheduled by a central scheduler, therefore the scheduler is required to be a trusted process.

In this paper we present a new multi-level multi-version global timestamp ordering mechanism (MLS_MVGTO). A MLS_MVGTO scheduler produces 1SR multi version histories according to transactions' global timestamp order. It is free from covert channels, and transactions of different security levels have the same privilege to execute. In addition, the MLS-MVGTO mechanism can be implemented as a set of schedulers which only process the same level transactions. Given the pre-knowledge about transactions' operations, read-only transactions are able to read more recent version, which as a result enables the higher level transactions to read more recent version.

The rest of the paper is organized as follows: Section 1 introduces the MLS_MVGTO mechanism and the global timestamp generating methods. Section 2 provides two possible improvements on read-only transactions. Section 3 gives the conclusion and the open questions to be discussed in the future work.

1 The Multi-Version Global Timestamp Ordering on the MLS DBMS

This paper accepts the basic definitions of transactions and histories in multi-version DB given in Ref.[11], and the formal expressions are written by Z specification language[10].

1.1 The global timestamp

When transaction T starts, it is assigned with two timestamps: the real timestamp $ts(T_i)$ ("timestamp" in short) and the virtual timestamp $vts(T_i)$, which reflects the serialization order of all level transactions. A global timestamp of a transaction is made up of its timestamp, virtual timestamp, and its security level, as defined below:

Definition 1.1. The global timestamp of a transaction $gts(T_i)$ is a three tuple of T_i's virtual timestamp $vts(T_i)$, timestamp $ts(T_i)$, and security level $L(T_i)$: $gts(T_i)=(vts(T_i),L(T_i),ts(T_i))$.

The timestamp is required to be unique for all transactions which are in the same security level. The virtual timestamp may not be unique for each transaction. However, in any cases a global timestamp can uniquely identify a transaction, and the MLS-MVGTO scheduler processes operations by global timestamp order.

Definition 1.2. the global timestamp order of two transactions T_i,T_j is defined as follows:

$$gts(T_i) < gts(T_j) \Leftrightarrow vts(T_i) < vts(T_j) \vee (vts(T_i) = vts(T_j) \wedge L(T_i) > L(T_j)) \vee (vts(T_i) = vts(T_j) \wedge L(T_i) = L(T_j) \wedge ts(T_i) < ts(T_j)).$$

This definition clarifies that the global timestamp order is firstly decided by the virtual timestamp order, then by security level order (from low to high), finally by the timestamp order.

1.2 The MLS_DBMS properties

In a MLS_DBMS, the operations of all transactions must satisfy the security properties given in the security model. In the MLS_MVGTO mechanism, two additional properties must be satisfied. The MLS-MVGTO_SR Property ensures that the MLS-MVGTO scheduler produce 1SR histories and the Write-Invalidation-Free property ensures that no covert channel will be introduced.

1.2.1 The MLS-MVGTO-SR property

A MLS-MVGTO scheduler produces multi-version history H, which satisfies the MLS-MVGTO-SR Property. The main characteristics can be formally described into the following sub_properties:

MLS-MVGTO-SR.1 $\forall T_i, T_j \in T \mid L(T_i) = L(T_j) \bullet ts(T_i) = ts(T_j) \Rightarrow i = j$.

MLS-MVGTO-SR.2 $\forall T_i \in T \mid vts(T_i) \le ts(T_i)$.

MLS-MVGTO-SR.3 $\forall r_i[x_k] \in H \mid i \ne k \bullet gts(T_i) > gts(T_k)$.

MLS-MVGTO-SR.4.1.

$$\forall r_i[x_k], w_j[x_j] \in H \mid L(T_i) = L(x) \wedge k \ne j \bullet gts(T_i) < gts(T_j) \vee gts(T_j) < gts(T_k) \vee (j = i \wedge r_i[x_k] < w_j[x_j])\ .$$

MLS-MVGTO-SR.4.2 $\forall r_i[x_k], w_j[x_j] \in H \mid L(T_i) > L(x) \wedge k \ne j \bullet gts(T_i) < gts(T_j) \vee gts(T_j) < gts(T_k)$.

MLS-MVGTO-SR.5 $\forall r_i[x_k] \in H \mid c_i \in H \wedge i \ne k \bullet c_k < c_i$.

MLS-MVGTO-SR.1 says that all transactions in the same level must have different timestamps. This implies that the tuple ($L(T_i)$,$ts(T_i)$) can uniquely identify a transaction, therefore $gts(T_i)$ is unique. MLS-MVGTO-SR.2 says that the virtual timestamp of any level transaction is always no earlier than its timestamp. MLS-MVGTO-SR.3 says that T_i only reads a version written by T_k when $gts(T_k)$ is less than $gts(T_i)$. MLS-MVGTO-SR.4 says that when the scheduler processes a read $r_i[x]$, it returns the version x_k written by T_k which has the largest timestamp less than $gts(T_i)$. This property implies that if the scheduler receives a late write operation $w_j[x_j]$ after it outputs a read down operation $r_i[x_k]$, and $gts(T_k)<gts(T_j)<gts(T_i)$. Then the scheduler will reject $w_j[x_j]$ and abort T_j. We call this the write invalidation, $w_j[x_j]$ is the late write and $r_i[x_k]$ invalidates $w_j[x_j]$. MLS-MVGTO-SR.4.1 describes the same level read operations, while MLS-MVGTO-SR.4.2 is for the read-down operations. MLS-MVGTO-SR.5 says that the history H is recoverable.

Theorem 1.1. Each history H produced by a MLS-MVGTO scheduler is 1SR (We omit the proof here).

1.2.2 The Write-Invalidation-Free property

MLS-MVGTO-SR.4 promises 1SR history, but the write invalidation may cause covert channels. To avoid the covert channel, a MLS_MVGTO scheduler must also hold the following Write-Invalidation-Free property.

Definition 1.3. **Write-Invalidation-Free Property**: Transaction T_i satisfies the Write-Invalidation-Free Property, if and only if for its each read-down operations $r_i[x_k]$, there is NO transaction T_j that satisfies: $w_j[x_j] \in T_j \wedge r_i[x_k] \in H \wedge L(T_j)=L(T_k)<L(T_i) \wedge gts(T_k)<gts(T_j)<gts(T_i)$. If all transactions in the transaction set T satisfies this property, then we say T satisfies the Write-Invalidation-Free Property.

1.3 The basic global timestamp generation steps

The implementation of a MLS_MVGTO scheduler which satisfies the above two properties requires a concrete global transaction timestamp generating method. Because $L(T_i)$ and $ts(T_i)$ of T_i are fixed, this section presents a basic $vts(T_i)$ generation function that satisfies above two properties.

$vts(T_i)$ is generated based on a T_i's snapshot function. Function $Snap(T_i,l)$ records the transactions at level l which is active (transactions are started, but not committed yet) when T_i starts, that is:

$$Snap(T_i, l) = \begin{cases} \{T_j : T \mid L(T_j) = l \wedge ts(T_j) < ts(T_i) < end\_time(T_j)\}, & l < L(T_i) \\ \varnothing, & l \ge L(T_i) \end{cases}.$$

T_i's complete snapshot is: $Snap(T_i)=\bigcup_{0<l<L(T_i)} Snap(T_i,l)$.

The basic global timestamp generating steps is described as follows:

(1) When T_i starts, $ts(T_i)$ is generated and assigned to T_i.

(2) $Snap(T_i)$ is calculated and assigned to T_i. The function is defined above.

(3) Compute the smallest virtual timestamp of $Snap(T_i)$ by the $MVS()$ function which is defined as follows:

$$MVS(T)=\{T_k\in T \mid vts(T_k)\} \quad \forall T_j\in T \mid \bullet vts(T_j)\geq MVS(T).$$

(4) $vts(T_i)$ is defined as follows:

$$vts(T_i)=\begin{cases} ts(T_i), & Snap(T_i)=\varnothing \\ \min\{ts(T_i),MVS(Snap(T_i))\}, & Snap(T_i)\neq\varnothing \end{cases}.$$

(5) The final global timestamp of T_i is: $gts(T_i)=(vts(T_i),L(T_i),ts(T_i))$.

Step four implies that $vts(T_i)\leq ts(T_i)$. All other MLS_MVGTO_SR properties are naturally satisfied by the definition of the scheduler. With this basic function, a transaction is scheduled before any active lower level transactions when it starts. It can be proved that this virtual timestamp function implies the following lemmas(We omit the proof here):

Lemma 1.1. $\forall T_i,T_j\in T \mid L(T_i)=L(T_j)\bullet ts(T_i)<ts(T_j)\Rightarrow gts(T_i)<gts(T_j)$.

Lemma 1.2. A MLS-MVGTO scheduler that uses $vts()$ function satisfies Invalidation-Write-Free property.

Lemma 1.1 means that in any security level, the global timestamp order of transactions are the same as their timestamp order. Therefore the scheduler can process the same level transactions based on their timestamp only. By Lemma 1.2, we can say that MLS-MVGTO scheduler will not introduce any covert channel.

According to Lemma 1.1, for the same level read operations, each scheduler of one security level performs as the traditional MVTO scheduler does, while for read-down operations, the scheduler use global timestamp instead. Since these schedulers only need to get information from lower level schedulers and this type of access does not violate the security property, they need not to be implemented as trusted code.

2 Improved Virtual Timestamp Functions

The above steps could not prevent a transaction read old version data, because it is too strict to assume that all active lower level transactions may have late write operations. Given more pre-knowledge about transactions' operations, a transaction has a virtual timestamp much close to its timestamp. In this section, we propose two types of improvements based on read-only transactions, which still hold the above two properties. Read-only transactions are one special type of transactions, and T_i is called a read only transaction if and only if its write set is empty $WS(T_i)=\varnothing$.

2.1 Given the set of levels

The first improvement is based on the assumption that a read-only transaction's read-down level set is known. Suppose $Levels(T_i)$ returns the set of lower levels that a read-only transaction T_i may read from:

$$Levels(T_i)=\{l:L \mid r_i[x]\in T_i\wedge L(x)=l\wedge l<L(T_i)\}.$$

T_i has no late writes, therefore it can be removed from higher level transactions' snapshots. That is:

$$Snap'(T_i,l)=\begin{cases}\{T_j:T \mid L(T_j)=l\wedge WS(T_j)\neq\varnothing\wedge ts(T_j)<ts(T_i)<end\_time(T_j)\}, & l<L(T_i)\\ \varnothing, & l\geq L(T_i)\end{cases}.$$

It is obvious that any non-empty set $Levels(T_i)$ satisfies $\forall l\in Levels(T_i)\mid l<L(T_i)$ and there are no late write operations in other security levels. Therefore the snapshot function of read only transactions $Snap\_RO(T_i,l)$ can be further modified as:

$$Snap\_RO(T_i,l)=\begin{cases}Snap'(T_i,l), & l\in Levels(T_i)\\ \varnothing, & \text{otherwise}\end{cases}.$$

The virtual timestamp function is then modified as:

$$vts(T_i)=\begin{cases}\min\{ts(T_i),MVS(Snap'(T_i))\}, & WS(T_i)\neq\varnothing\wedge Snap'(T_i)\neq\varnothing\\ \min\{ts(T_i),MVS(Snap\_RO(T_i))\}, & WS(T_i)=\varnothing\wedge Snap\_RO(T_i)\neq\varnothing\\ ts(T_i), & \text{otherwise}\end{cases}.$$

The global timestamp of transaction T_i is: $gts(T_i)=(vts(T_i),L(T_i),ts(T_i))$.

For any T_i and l there is $Snap\_RO(T_i,l)\subseteq Snap'(T_i,l)$, therefore $Snap\_RO(T_i)\subseteq Snap'(T_i)$. Then according to the virtual timestamp definition, $vts\_RO(T_i)\geq vts(T_i)$. This means that for a read only transaction T_i, it is possible to have a more "recent" virtual timestamp compared with the basic virtual timestamp function.

Here Lemma 1.1 does not hold any more, the global timestamp order of two same level transactions may differ from their timestamp order. In stead, the following Lemma 2.1 is true (We omit the proof here):

Lemma 2.1. $L(T_i)=L(T_j)\wedge ts(T_i)<ts(T_j)\wedge WS(T_i)\neq\varnothing\wedge WS(T_j)\neq\varnothing\Rightarrow gts(T_i)\leq gts(T_j)$.

Lemma 2.1 means that in any security level, the global timestamp order of NOT read only transactions are the same as their timestamp order. If we remove all read-only transactions from the transaction set T, then Lemma 3.1 equals to Lemma 1.1. Both the MLS-MVGTO-SR property and the Write-Invalidation-Free property are satisfied.

However there may be late write operation in the same level. This improvement enhances the chance of the abortion of a transaction T_i by other same level transactions $T_j(L(T_i)=L(T_j))$.

2.2 Given the read set and write set

Moreover, if the read data set and write data set of any transaction are available, the virtual timestamp function can be further modified to be more precisely close to its timestamp. Let us suppose when transaction T_i starts, it reports two data sets to the scheduler, the read-down set $RDS(T_i)$, a set of data items x which is read by T_i: $x\in RDS(T_i)\Leftrightarrow r_i[x]\in T_i\wedge L(x)<L(T_i)$, and the write set $WS(T_i)$, a set of data items x which T_i writes on: $x\in WS(T_i)\Leftrightarrow w_i[x]\in T_i\wedge L(x)=L(T_i)$.

The $Snap''()$ for read-only transaction is slightly different from the basic snapshot function:

$$Snap''(T_i,l)=\begin{cases}\{T_j:T\mid L(T_j)=l\wedge WS(T_j)\cap RDS(T_i)\neq\varnothing\wedge ts(T_j)<ts(T_i)<end\_time(T_j)\}, & l\in Levels(T_i)\\ \varnothing, & \text{otherwise}\end{cases}.$$

The difference between $Snap''(T_i,l)$ and $Snap\_RO(T_i,l)$ is that the predicate $WS(T_j)\cap RDS(T_i)\neq\varnothing$ takes place of $WS(T_j)\neq\varnothing$. This change indicates that for any T_i and l there is $Snap''(T_i,l)\subseteq Snap\_RO(T_i,l)$, therefore $Snap''(T_i)\subseteq Snap\_RO(T_i)$. This means that for a read only transaction T_i, it is possible to have a more "recent" virtual timestamp.

The snapshot function for read-write transactions is still $Snap''(T_i,l)$. And the virtual timestamp function is expressed as:

$$vts(T_i)=\begin{cases}\min\{ts(T_i),MVS(Snap'(T_i))\}, & WS(T_i)\neq\varnothing\wedge Snap'(T_i)\neq\varnothing\\ \min\{ts(T_i),MVS(Snap''(T_i))\}, & WS(T_i)=\varnothing\wedge Snap''(T_i)\neq\varnothing\\ ts(T_i), & \text{otherwise}\end{cases}.$$

In this improvement, Lemma 1.1 is satisfied. This function also satisfies the MLS-MVGTO-SR property and the Write-Invalidation-Free property.

3 Conclusions

In this paper we present the concept of global timestamp and a multi-level multi-version global timestamp

ordering (MLS_MVGTO) mechanism. It can be proved that all history produced by a MLS_MVGTO scheduler is 1SR, and it is free from covert channels. This approach does not need a global trusted scheduler. Instead, it can be built on the enhanced untrusted multi-version schedulers. Therefore we also propose two improvements which enable the transaction to read a more recent version. However some problems remain open in this approach, and one of them is the storage problem because those old versions cannot be erased from system in time. This is the target of our future work.

References:

[1] McDermott J, Jajodia S. Orange locking: Channel-Free database concurrency control via locking. In: Thuraisingham BM, Landwehr CE, eds. Database Security, VI: Status and Prospects. New York: Elsevier Science, 1993. 267−284.

[2] Bertino E, Jajodia S, Mancini L, Ray I. Advanced trans. Proc. in multilevel secure file stores. IEEE Trans. on Knowledge and Data Engineering, 1998,10(1):120−135.

[3] Jajodia S, Mancini LV, Setia S. A fair locking protocol for multilevel secure databases. In: Proc. of the 11th Computer Security Foundations Workshop. Washington: IEEE Computer Society Press, 1998. 168−178.

[4] Mancini LV, Ray I. Secure concurrency control in MLS databases with two versions of data. In: Bertino E, Kurth H, Martella G, Montolivo E, eds. Proc. of the 4th European Symp. on Research in Computer Security (ESORICS). Springer-Verlag, 1996. 304−323.

[5] Pal S. A locking protocol for multilevel secure database using two committed versions. In: Proc. of the 10th Annual Conf. on Computer Assurance, COMPASS'95. 1995. 197−210.

[6] Keefe TF, Tsai WT. Multiversion concurrency control for multilevel secure database systems. In: Proc. of the IEEE Symp. on Security and Privacy. 1990. 369−383.

[7] Keefe TF, Tsai WT. A multiversion transaction scheduler for centralized multilevel secure database systems. In: Proc. of the 1st High-Assurance Systems Engineering Workshop (HASE'96). Washington: IEEE Computer Society, 1996. 206−213.

[8] Ammann P, Jacckle F, Jajodia S. Concurrency control in secure multilevel databases via a two-snapshot algorithm. Journal of Computer Security, 1995,3(3):87−113.

[9] Bernstein PA, Hadzilacos V, Goodman N. Concurrency Control and Recovery in Database Systems. Massachussetts: Addison-Wesley, 1987.

[10] Woodcock J, Davies J. Using Z Specification, Refinement, and Proof. Hertfordshire: Prentice Hall Europe, 1996.

ZHANG Min was born in 1975. She is a Ph.D. candidate and research assistant at the Institute of Software, the Chinese Academy of Sciences. Her current research areas are mobile agent and database security.

FENG Deng-Guo was born in 1965. He is a professor and doctoral supervisor of the Institute of Software, the Chinese Academy of Sciences, and a CCF senior member. His research areas are cryptography theory and information security.

XU Zhen was born in 1976. His current research areas are database security and system security.