FORMALISING PRIORITY CEILING PROTOCOL WITH DYNAMIC ADJUSTMENT OF SERIALIZATION ORDER IN REAL TIME DATABASES

Doan Van Ban

Institute of Information Technology

Nguyen Huu Ngu

College of Science, VNU

Ho Van Huong

Governmental Cipher Department

Abstract. In this paper, we apply a formal model of real time database systems using duration calculus (DC) to give formal specification of the Priority Ceiling Protocol with Dynamic Adjustment of Serialization Order (PCP-DA) and a formal proof for the correctness of the PCP-DA using the DC proof system. We devise a worst case schedulability analysis for PCP-DA which provides a better schedulability condition compared to R/WPCP. We then show that the number of priority inversion for transactions scheduled by PCP-DA may be more than one in a multiprocessor environment.

1. Introduction

In recents years, a lot of reseach work has been devoted to the design of database systems for real time applications. A real time database system is defined as a database system when transaction are associated with deadlines on their completion times. In addition, some of the data items in a real time database are associated with temporal constraints on their validity [5,12]. Example applications include systems for avionic and space, air traffic control, robotics, nuclear power plants, integrated manufacturing, stock trading, and network management.

The main goal of this paper is to formalise some aspects of RTDBS, in particular PCP-DA using DC. This will allow us to verify the correctness of PCP-DA formally using the proof system of the DC. We shows that the number of priority inversion for transactions scheduled by PCP-DA may be more than one in a multiprocessor environment. We make use of duration calculus because DC is a simple and powerful logic for reasoning about real time systems, and DC has been used successfully in many case studies, for example [6,8,9,10,13], we will take it to be the formalism for our specification in this paper.

Our approach is summarised as follows: We apply a formal model of RTDBS proposed by Ho Van Huong and Dang Van Hung [9] to specify and verify the Priority Ceiling Protocol with Dynamic Adjustment of Serialization Order. The paper is organized as follows: we give an informal abstract description of RTDBS and PCP-DA. Section 3 introduces a review of DC. Section 4 presents a fomalization of PCP-DA in DC and a formal

proof of correctness of this protocol in section 5. Section 6 shows the Blocking of PCP-DA in Multiprocessor Environment.

2. Preliminaries

We briefly recall in this section the main concepts of RTDBS and the integration of concurrency control with priority scheduling, which will justify our formal model given in later sections. We refer to [5, 9,12] for more comprehensive introduction to RTDBS.

A real time database systems can be viewed as an amalgamation of conventional database management system and real time system [5]. In RTDB, the transactions not only have to meet their deadline, but also have to use the data that are valid during their execution. Many previous studies have focused on integrating concurrency control protocols with priority scheduling in RTDBS [5,12].

For example, the Read/Write Priority Ceiling Protocol (R/WPCP) is an extension of the well-known Priority Ceiling Protocol (PCP) [12] in real time concurrency control, adopts Two Phase Locking (2PL) in preserving the serializability of transactions executions. However, R/WPCP is too conservative in scheduling transactions to access the shared data, resulting in unnecessary blockings.

Therefore, some studies (e.g., [5,11,12]) employed the notion of dynamic adjustment of serialization order. For example, the Priority Ceiling Protocol with Dynamic Adjustment of Serialization Order (PCP-DA) [11] shows that a higher priority transaction can preempt a lower priority transaction on data conflicts by using the notion of dynamic adjustment of serialization order, avoiding unnecessary blockings. The goal of designing their new protocol is to give critical transactions high priority in accessing the shared data so that they can complete their executions as soon as possible. The fewer the transaction blockings, the better the schedulability conditions for a transaction set. By dynamically adjusting the serialization order among conflicting transactions, PCP-DA allows a higher priority transaction to preempt uncommitted lower priority transactions while it prevents lower priority transactions from being restarted even in the face of data conflicts.

3. Duration calculus

The Duration Calculus(DC) represents a logical approach to formal design of real time systems. DC is proposed by Zhou, Hoare, and Ravn, which is an extension of real arithmetic and interval temporal logic. We refer to 7 for more comprehensive introduction to Duration Calculus.

We give now shorthands for some duration formulas which are often used. For an arbitrary state variable P, $\lceil\lceil P \rceil\rceil$ stands for $(\int P = \ell) \wedge (\ell > 0)$. This means that interval is a non-point interval and P holds almost everywhere in it. We use $\lceil\lceil\ \rceil\rceil$ to denote the predicate which is true only for point intervals. Modalities $\Diamond$, $\Box$ are defined as: $\Diamond D \widehat{=} \mathbf{true}^\frown D^\frown \mathbf{true}$,

$\Box D \widehat{=} \neg\Diamond\neg D$ (we use $\widehat{=}$ as a define). This means that $\Diamond D$ is true for an interval iff D holds for some its subinterval, and $\Box D$ is true for an interval iff D holds for every its subintervals.

DC with abstract duration domain is a complete calculus, which has a powerful proof system. Here we give only some rules and axioms that will be used later in this paper.

$$\text{(ITL1)(Monotonicity)} A \Rightarrow B \vdash (A^\frown C \Rightarrow B^\frown C) \wedge (C^\frown A \Rightarrow C^\frown B)$$
$$\text{(ITL2)(Associativity)} (A^\frown B)^\frown C \iff A^\frown (B^\frown C)$$
$$\text{(ITL3)(Unit)} (A^\frown [\![\,]\!]) \iff ([\![\,]\!]^\frown A) \iff A$$
$$\text{(ITL4)(Zero)} (A^\frown \textbf{false}) \iff (\textbf{false}^\frown A) \iff \textbf{false}$$
$$\text{(ITL5)}\ (x \geq 0 \wedge y \geq 0) \Rightarrow ((\ell = x + y) \Leftrightarrow ((\ell = x)^\frown (\ell = y))).$$

Forward Induction: Let $\mathcal{H}(\mathcal{X})$ be a DC formula schema containing the propositional letter $\mathcal{X}$, and let P be any state expression.

If $\mathcal{H}([\![\,]\!])$ *and* $\mathcal{H}(\mathcal{X}) \vdash \mathcal{H}(\mathcal{X} \vee (\mathcal{X}^\frown [\![P]\!]) \vee (\mathcal{X}^\frown [\![\neg P]\!]))$
then $\mathcal{H}(\textbf{true})$.

Backward Induction: Let $\mathcal{H}(\mathcal{X})$ be a DC formula schema containing the propositional letter $\mathcal{X}$, and let P be any state expression.

If $\mathcal{H}([\![\,]\!])$ *and* $\mathcal{H}(\mathcal{X}) \vdash \mathcal{H}(\mathcal{X} \vee ([\![P]\!]^\frown \mathcal{X}) \vee ([\![\neg P]\!]^\frown \mathcal{X}))$
then $\mathcal{H}(\textbf{true})$.

Using the proof system, we can easily prove the following theorems which will be used later. Below, x and y are assumed to be non-negative real numbers.

$$\text{DC1}\ [\![P]\!]^\frown [\![P]\!] \iff [\![P]\!]$$
$$\text{DC2}\ [\![P]\!] \wedge [\![Q]\!] \iff [\![P \wedge Q]\!]$$
$$\text{DC3}\ [\![P]\!] \wedge [\![A]\!]^\frown [\![B]\!] \iff ([\![P]\!] \wedge [\![A]\!])^\frown ([\![P]\!] \wedge [\![B]\!])$$
$$\text{DC4} \begin{pmatrix} (x > y \wedge \ell = x \wedge [\![S]\!]) \\ \Rightarrow ((([\![S]\!] \wedge \ell = y)^\frown [\![S]\!]) \end{pmatrix}.$$

4. Formalisation of Priority Ceiling Protocol with Dynamic Adjustment of Serialization Order in RTDB

In this section, we adapt a formal model of Real Time Database System (RTDBS) using DC [9] to specify PCP-DA.

As presented in section 2, PCP-DA is an extension of the well-known PCP in real time concurrency control. PCP-DA use dynamic adjustment of serialization order to redefine the semantics of the write/read conflicts between two transactions.

4.1 Formalisation of PCP-DA

In order to formalise the protocol, for each $i, j \leq n$, $x \in \mathcal{O}$, we introduce notations as follows. Let $WPL(x)$ be constants and $PN \in \mathcal{N}$ denote the set of priority

numbers, $T_i.Rlocked-data$, $T_i.NoRlocked-data$, $T_i.sysceil$ be temporal variables. In addition, we use some state variables below $T_i.request\_lock(x)$, $T_i.request\_rlock(x)$, $T_i.request\_wlock(x)$, $T_i.wait\_rlock(x)$, $T_i.wait\_wlock(x)$, $T_i.hold\_lock(x)$, $T_i.hold\_rlock(x)$, $T_i.hold\_wlock(x)$, $T_i.committed$, $T_i.period$, $T_i.run$, $T_i.ready$ which be specifed in [9] for our model.

The write priority ceiling $WPL(x)$ of data object x is equal to the highest priority of transactions which may write x.

$$WPL(x) \widehat{=} \max\{p_i | x \in WO_i, i \leq n\}.$$

$T_i.NoRlocked-data$ denotes a data object x that is not being read-locked by transactions other than T_i when T_i requests to lock x at time t.

$$T_i.NoRlocked-data \in [\text{Time} \rightarrow 2^O]$$
$$T_i.NoRlocked-data(t) = \{x \mid \neg T_j.hold\_rlock(x)(t), T_i \neq T_j\}.$$

$T_i.Rlocked-data$ denotes a data object x that is being read-locked by transactions other than T_i when T_i requests to lock x at time t.

$$T_i.Rlocked-data \in [\text{Time} \rightarrow 2^O]$$
$$T_i.Rlocked-data(t) = \{x \mid T_j.hold\_rlock(x)(t), T_i \neq T_j\}.$$

$T_i.sysceil$ denotes the highest write priority ceiling of data objects read-locked by transactions other than T_i at time t.

$$T_i.sysceil \in [\text{Time} \rightarrow PN]$$

$T_i.sysceil = 0$ if at time t object x is neither read-locked by some transactions.

$$T_i.sysceil(t) = \max\{WPL(x)(t) x \in T_i.Rlocked-data(t)\}.$$

T^* denotes the transaction holding a read-lock on a data object x whose write priority ceiling is equal to $T_i.sysceil$.

$$T^* \in [\text{Time} \rightarrow 2^T]$$
$$T^*(t) = \{T_i.hold\_rlock(x)(t) \mid WPL(x) = T_i.sysceil\}.$$

WO^* denotes the write set of T^*.

A transaction T_i is allowed to read-lock or write-lock a data object x if one of the locking conditions is true.

Condition 1: T_i requests a write-lock on x and x is not being read-locked by other transactions at time t.

$$LC1 \widehat{=} (T_i.request\_wlock(x)(t) = 1) \wedge T_i.NoRlocked-data.$$

Condition 2: T_i requests a read-lock on x and T_i's priority is higher than the highest write priority ceiling of data objects read-locked by other transactions.

$$LC2 \widehat{=} (T_i.request\_rlock(x)(t) = 1) \wedge (p_i > T_i.sysceil).$$

Condition 3: T_i requests a read-lock on x and T_i's priority is higher than the highest priority of transaction that may write x and x is not in the write set of T^*.

$$LC3 \widehat{=} (T_i.request\_rlock(x)(t) = 1) \wedge (p_i > WPL(x)) \wedge x \notin WO^*.$$

Condition 4: T_i requests a read-lock on x and T_i's priority is equal to the highest priority of transaction that may write x and x is not being read- locked by other transactions and x is not in the write set of T^*.

$$LC4 \widehat{=} (T_i.request\_rlock(x)(t) = 1) \wedge (p_i = WPL(x)) \wedge T_i.NoRlocked - data \wedge x \notin WO^*.$$

When a transaction T_i attempts to lock a data object x, T_i will be blocked and the lock on an object x will be denied, if one of the locking conditions is false. Therefore, the *blockedby* state expression is:

$$T_i.blockedby(T_j) \widehat{=} (LC1 = \textbf{false}) \vee (LC2 = \textbf{false}) \vee (LC3 = \textbf{false}) \vee (LC4 = \textbf{false}).$$

Using the framework presented above, we present DC formula schemas for specifing PCP-DA. First, the formula schema for the preemptive priority scheduler is presented the same way in [9,10] as follows:

Let $HiPri_{PCP-DA}(T_i, T_j)$ be a boolean-valued function for denoting which transaction between T_i and T_j has a higher priority.

(a) $HiPri_{PCP-DA}$ is a partial order:

$$\bigwedge_{T_i \neq T_j \in \mathcal{T}} (HiPri_{PCP-DA}(T_i, T_j) \Rightarrow \neg HiPri_{PCP-DA}(T_j, T_i))$$

$$\bigwedge_{T_i \neq T_j \neq T_k \in \mathcal{T}} \left(\begin{array}{r} HiPri_{PCP-DA}(T_i, T_k) \wedge HiPri_{PCP-DA}(T_k, T_j) \\ \Rightarrow HiPri_{PCP-DA}(T_i, T_j) \end{array} \right).$$

(b) $HiPri_{PCP-DA}$ depends on the priority inherited by transactions:

$$\bigwedge_{T_i \neq T_j \neq T_k \in \mathcal{T}} \left(\begin{array}{r} T_k.blockedby(T_i) \\ \Rightarrow (HiPri_{PCP-DA}(T_k, T_j) \Rightarrow HiPri_{PCP-DA}(T_i, T_j)) \end{array} \right)$$

$$\bigwedge_{T_i \neq T_j \in \mathcal{T}} \left(\begin{array}{c} \bigwedge_{T_k \in \mathcal{T}} (\neg T_k.blockedby(T_i)) \\ \Rightarrow (HiPri_{PCP-DA}(T_i, T_j) \Rightarrow p_i > p_j) \end{array} \right).$$

The first formula expresses that when a transaction T_i inherits the priority of transaction T_k, if $HiPri_{PCP-DA}(T_k, T_j)$ then $HiPri_{PCP-DA}(T_i, T_j)$. The second formula shows

that if a transaction T_i does not inherit any priority, then the relation $HiPri_{PCP-DA}$ is consistent with the original assigned priorities.

The preemptive priority scheduler can be expressed as:

$$PPS \widehat{=} \bigwedge_{T_i \neq T_j \in \mathcal{T}} \Box([\![T_i.run]\!] \wedge [\![T_j.ready]\!] \Rightarrow [\![HiPri_{PCP-DA}(T_i, T_j)]\!]).$$

The Granting rule for PCP-DA can be expressed as:

Granting Rule used to decide if the lock data object requested is granted or not.

$$Gr \widehat{=} \bigwedge_{T_i \in \mathcal{T}} \bigwedge_{x \in \mathcal{O}} \Box \left(\begin{array}{r} [\![\neg T_i.hold\_lock(x)]\!]^\frown [\![T_i.hold\_lock(x)]\!] \\ \Rightarrow ((LC1 = \textbf{true}) \vee (LC2 = \textbf{true}) \vee (LC3 = \textbf{true}) \vee (LC4 = \textbf{true})) \end{array} \right)$$

The blocking rule for PCP-DA can be expressed as:

Blocking Rule used to decide whether a transaction is blocked on its request for a lock data object or not.

$$Bl \widehat{=} \bigwedge_{T_i \in \mathcal{T}} \bigwedge_{x \in \mathcal{O}} \Box \left(\begin{array}{r} ((LC1 = \textbf{true}) \vee (LC2 = \textbf{true}) \vee (LC3 = \textbf{true}) \vee (LC4 = \textbf{true})) \\ \Rightarrow [\![\neg T_i.wait\_lock(x)]\!] \end{array} \right).$$

Then, the unblocking rule can be specified as:

Unblocking Rule used for deciding which among the blocked transactions is to be granted the lock data object.

$$UnBl \widehat{=} \bigwedge_{T_i \neq T_j \in \mathcal{T}} \bigwedge_{x \in \mathcal{O}} \Box \left(\begin{array}{r} [\![T_i.wait\_lock(x) \wedge T_j.wait\_lock(x)]\!]^\frown \\ [\![\neg T_i.wait\_lock(x)]\!] \Rightarrow HiPri_{PCP-DA}(T_i, T_j) \end{array} \right).$$

By combining these formula schemas together, the scheduler, $PCP-DA$, is obtained:

$$PCP-DA \widehat{=} (SERIAL \wedge PPS \wedge Gr \wedge Bl \wedge UnBl).$$

For serializable condition, it has been proved in [11] that all executions of the transactions system produced by PCP-DA are serializable i.e $PCP-DA \models SERIAL$.

Properties:

The properties for the PCP-DA are blocked at most once and deadlock free like R/WPCP, BAP in [9,10], we have:

$$BAO \widehat{=} \bigwedge_{T_i} \Box([\![\bigvee_{x \in \mathcal{O}} T_i.hold\_lock(x)]\!] \Rightarrow [\![\bigwedge_{x \in \mathcal{O}} \neg T_i.wait\_lock(x)]\!]),$$

$$DLF \widehat{=} \Box\neg([\![\bigwedge_{T_i \in \mathcal{T}} \bigwedge_{x \in \mathcal{O}} (T_i.committed \vee T_i.wait\_lock(x)) \wedge \bigvee_{T_i \in \mathcal{T}} \bigvee_{x \in \mathcal{O}} T_i.wait\_lock(x)]\!]).$$

4.2 The schedulability condition of PCP-DA in RTDB

For schedulability condition, it has been proved in [14] that *a set of n periodic transactions using rate monotonic priority assignment under R/WPCP can always meet*

their deadlines if the following conditions are satisfied: $\sum_{i=1}^{n} C_i/P_i + B_i/P_i$ *is no greater than* $n(2^{1/n} - 1)$. Where B_i denotes the worst case blocking time of transaction T_i.

It can be easily seen that the above schedulability conditions were also applicable to PCP-DA. The schedulability condition for a transaction set depends on the value of B_i. The smaller the value of B_i is the better the schedulability condition.

We now determine the value of B_i in PCP-DA and compare it with that in R/WPCP as follows.

In PCP-DA, since write operations are preemptable, only read operations of lower priority transactions may block the write operations of higher priority transactions. A transaction T_L with a priority (p_L) lower than p_i may block T_i if T_L reads a data object x such that $WPL(x) \geq p_i$. Hence, we can use BTS_i denotes the set of transactions that may block T_i (i.e a set of transactions with priorities lower than p_i that may read a data object x such that $WPL(x) \geq p_i$. We have

$$BTS_i = \{T_L \mid p_L < p_i \text{ and } T_L \text{ reads } x \text{ and } WPL(x) \geq p_i\}.$$

On the other hand, R/WPCP, as shown in [14] has BTS_i:

$$BTS_i = \{T_L \mid p_L < p_i \text{ and } (T_L \text{ reads } x \text{ and } WPL(x) \geq p_i \text{ or } T_L \text{ writes } x \text{ and } APL(x) \geq p_i)\}$$

For both PCP-DA and R/WPCP, the worst case blocking time of transaction T_i is determined as follows:

$$B_i \widehat{=} \max\{C_L | T_L \in BTS_i, i \leq n\},$$

where C_L denotes the excution time of T_L. It can be observed that BTS_i in R/WPCP is a superset of that in PCP-DA. If the worst case blocking time B_i occurs in R/WPCP when T_L *writes* x *and* $APL(x) \geq p_i$, the value of B_i can be reduced in PCP-DA because T_L will not be included in BTS_i in PCP-DA.

Let $C_i^* = C_i + B_i$. For above conditions, we can formalise the schedulability condition for PCP-DA as:

$$(ENV \wedge Usys \wedge PCP - DA \wedge \sum_{i=1}^{n} C_i^*/P_i \leq n(2^{1/n} - 1)) \Rightarrow (\bigwedge_{i=1}^{n} (T_i.period \Rightarrow (\int T_i.run \geq C_i^*)))$$

where $ENV \wedge Usys$ are the set of the formulas to capture the axioms for the state variables introduced in a formal model of real time database systems in [9]. With limmited space no detailed specify is included. We refer interested readers to [9] for details.

5. Formal proof of the Priority Ceiling Protocol with Dynamic Adjustment of Serialization Order

In this section, we will show how we can use a formal model of real time database systems which proposed by Ho Van Huong and Dang Van Hung [9] to prove properties of PCP-DA are blocked at most once and deadlock free and the schedulability condition of PCP-DA.

In order to prove this properties, we need to make a distinction between a transaction being in the preempted state and blocked state. We make the assumption that while a transaction is preempted by a higher priority transaction, it is not blocked.

$$NB \widehat{=} \bigwedge_{T_i \neq T_j \in \mathcal{T}} \bigwedge_{x \in \mathcal{O}} \Box \left(\begin{array}{l} [\![T_i.run]\!]^\frown [\![\bigvee_{T_j \neq T_i \in \mathcal{T}} (T_j.run \wedge p_j > p_i)]\!] \\ \\ \Rightarrow [\![T_i.run]\!]^\frown [\![\bigwedge_{x \in \mathcal{O}} \neg T_i.wait\_rlock(x)]\!] \end{array} \right).$$

We need to give definitions as follows:

Definition 1 $ASS \widehat{=} PCP - DA \wedge ENV \wedge Usys.$

Definition 2

$$R_{PCP-DA}(T_i, x) \widehat{=} \bigwedge_{T_j \neq T_i \in \mathcal{T}} (\bigwedge_{x \in \mathcal{O}} (T_j.hold\_rlock(x) \vee T_j.hold\_wlock(x))$$
$$\Rightarrow (LC1 = \textbf{true}) \vee (LC2 = \textbf{true}) \vee (LC3 = \textbf{true}) \vee (LC4 = \textbf{true})).$$

5.1 PCP-DA is Deadlock Free

We prove this property by contradiction.

Theorem 1 $NB \wedge ASS \vdash DLF.$

Proof the Theorem 1

$$(1) \neg\Box\neg \left([\![\bigwedge_{T_i \in \mathcal{T}} \bigwedge_{x \in \mathcal{O}} (T_i.committed \vee T_i.wait\_lock(x)) \wedge \bigvee_{T_i \in \mathcal{T}} \bigvee_{x \in \mathcal{O}} T_i.wait\_lock(x)]\!] \right) \quad \text{(Assume)}$$

$$(2) \Diamond \left(\begin{array}{l} \bigwedge_{T_i \in \mathcal{T}} \bigwedge_{x \in \mathcal{O}} [\![(T_i.committed \vee T_i.wait\_lock(x))]\!] \wedge \\ \\ \bigvee_{T_i \in \mathcal{T}} \bigvee_{x \in \mathcal{O}} [\![T_i.wait\_lock(x)]\!] \end{array} \right) \quad \text{(ITL)}$$

$$(3)\ \Box \left(\begin{array}{l} \bigwedge_{T_i \in \mathcal{T}} ([\![\bigvee_{x \in \mathcal{O}} T_i.wait\_lock(x)]\!] \\ \Rightarrow [\![\bigvee_{T_j \in \mathcal{T}} \bigvee_{x \in \mathcal{O}} T_j.hold\_lock(x)]\!]) \end{array} \right) \quad (PCP - DA, \text{ENV})$$

$$(4) \Box \left(\begin{array}{l} [\![\bigvee_{T_i \in \mathcal{T}} \bigvee_{x \in \mathcal{O}} T_i.hold\_lock(x)]\!] \\ \\ \Rightarrow [\![\bigvee_{T_i \in \mathcal{T}} (\bigwedge_{x \in \mathcal{O}} \neg T_i.wait\_lock(x) \wedge \bigvee_{x \in \mathcal{O}} T_i.hold\_lock(x))]\!] \end{array} \right) \quad ((3),\ \text{BAO, ITL})$$

$$(5)\ \Box\left(\begin{array}{l}\bigwedge_{T_i\in\mathcal{T}}(\lceil\!\lceil\bigvee_{x\in\mathcal{O}}T_i.wait\_lock(x)\rceil\!\rceil\\ \Rightarrow\lceil\!\lceil\bigvee_{T_j\in\mathcal{T}}(\bigwedge_{x\in\mathcal{O}}\neg T_j.wait\_lock(x)\wedge\bigvee_{x\in\mathcal{O}}T_j.hold\_lock(x))\rceil\!\rceil\end{array}\right)\quad((3),\ (4),\ \mathrm{PL})$$

$$(6)\Diamond\left(\begin{array}{l}\bigwedge_{T_i\in\mathcal{T}}\bigwedge_{x\in\mathcal{O}}\lceil\!\lceil T_i.committed\vee T_i.wait\_lock(x)\rceil\!\rceil\\ \wedge\bigvee_{T_i\in\mathcal{T}}\lceil\!\lceil\bigwedge_{x\in\mathcal{O}}\neg T_i.wait\_lock(x)\wedge\bigvee_{x\in\mathcal{O}}T_i.hold\_lock(x)\rceil\!\rceil\end{array}\right)\quad((2),\ (5),\ \mathrm{PL})$$

$$(7)\Diamond\left(\bigvee_{T_i\in\mathcal{T}}\bigvee_{x\in\mathcal{O}}\lceil\!\lceil\neg T_i.wait\_lock(x)\wedge T_i.hold\_lock(x)\wedge(T_i.committed\vee T_i.wait\_lock(x))\rceil\!\rceil\right)$$

$$((6),\ \mathrm{PL})$$

(8) **false** $\quad((7),\ ENV,\ \mathrm{PL})$

(9) DLF $\quad((1),\ (8),\ \mathrm{PL})$

□

5.2 Blocked at most once of PCP-DA

The property of PCP-DA where a transaction is blocked at most once can be expressed as follows:

Theorem 2

$$NB\wedge ASS\vdash BAO$$

Proof the Theorem 2

We prove this by induction: First, assume:

$$\mathcal{H}(\mathcal{X})\widehat{=}\mathcal{X}\wedge\lceil\!\lceil\bigvee_{x\in\mathcal{O}}T_i.hold\_lock(x)\rceil\!\rceil\Rightarrow\lceil\!\lceil\bigwedge_{x\in\mathcal{O}}\neg T_i.wait\_lock(x)\rceil\!\rceil$$

$$\Gamma\widehat{=}NB\wedge ASS$$

Base case:

$$\begin{array}{ll}\Gamma\vdash\mathcal{H}(\lceil\!\lceil\ \rceil\!\rceil)&\\ \Rightarrow\lceil\!\lceil\ \rceil\!\rceil\wedge\lceil\!\lceil\bigvee_{x\in\mathcal{O}}T_i.hold\_lock(x)\rceil\!\rceil&\\ \Rightarrow\lceil\!\lceil\bigwedge_{x\in\mathcal{O}}\neg T_i.wait\_lock(x)\rceil\!\rceil&\textbf{(false)}\\ \Rightarrow\lceil\!\lceil\bigwedge_{x\in\mathcal{O}}\neg T_i.wait\_lock(x)\rceil\!\rceil&(\mathrm{ITL})\\ \Rightarrow\ \textbf{true}&(\mathrm{ITL})\end{array}$$

For the inductive step, we must establish:

$$\Gamma, \mathcal{H}(\mathcal{X}) \vdash \mathcal{H}(\mathcal{X} \vee (\mathcal{X}^\frown [\![R_{PCP-DA}(T_i, x)]\!]) \vee (\mathcal{X}^\frown [\![\neg R_{PCP-DA}(T_i, x)]\!])).$$

We now consider two cases:

1. $\Gamma, \mathcal{H}(\mathcal{X}) \vdash \mathcal{H}(\mathcal{X}^\frown [\![R_{PCP-DA}(T_i, x)]\!])$,
2. $\Gamma, \mathcal{H}(\mathcal{X}) \vdash \mathcal{H}(\mathcal{X}^\frown [\![\neg R_{PCP-DA}(T_i, x)]\!])$.

Case 1:

$$\begin{aligned}
&\Gamma, \mathcal{H}(\mathcal{X}) \vdash \mathcal{H}(\mathcal{X}^\frown [\![R_{PCP-DA}(T_i, x)]\!]) \\
&\Rightarrow \mathcal{X}^\frown [\![R_{PCP-DA}(T_i, x)]\!] \wedge [\![\bigvee_{x \in \mathcal{O}} T_i.hold_lock(x)]\!] \\
&\Rightarrow (\mathcal{X} \wedge [\![\bigvee_{x \in \mathcal{O}} T_i.hold_lock(x)]\!]) \\
&{}^\frown([\![R_{PCP-DA}(T_i, x)]\!] \wedge [\![\bigvee_{x \in \mathcal{O}} T_i.hold_lock(x)]\!]) && \text{(DC3)} \\
&\Rightarrow [\![\bigwedge_{x \in \mathcal{O}} \neg T_i.wait_lock(x)]\!]^\frown([\![R_{PCP-DA}(T_i, x)]\!] \wedge [\![\bigvee_{x \in \mathcal{O}} T_i.hold_lock(x)]\!]) && \text{(H(X))} \\
&\Rightarrow [\![\bigwedge_{x \in \mathcal{O}} \neg T_i.wait_lock(x)]\!]^\frown [\![\bigwedge_{x \in \mathcal{O}} \neg T_i.wait_lock(x)]\!] && \text{(PCP-DA, Def.2)} \\
&\Rightarrow [\![\bigwedge_{x \in \mathcal{O}} \neg T_i.wait_lock(x)]\!]. && \text{(DC1)}
\end{aligned}$$

Case 2: The proof this case can be done the same way as above and it is omitted here.

5.3 Proof Theorem: The schedulability condition of PCP-DA

In this theorem, we only need to consider the interval $[0, P_n]$, where as we recall, P_n is the largest period. An important concept used in Liu and Layland's informal proof is that of *full utilisation* of the processor. They merely stated that the processor is fully utilised if any increase of the required execution time C_i^* will cause the scheduling to be infeasible. We give a precise and simple definition of fully utilisation.

Definition 3 *Transactions T_1, T_2,...,T_n, with required execution time C_1^*, C_2^*,...,C_n^* and periods P_1, P_2,...,P_n, are said to fully utilise the processor, denoted as $F_u(C_1^*, \cdots, C_n^*, P_1, \cdots, P_n)$, iff for any $0 \leq x \leq P_n, \sum_{i=1}^{n} \lceil x/P_i \rceil C_i^* \geq x$.* At any time point x, $\sum_{i=1}^{n} \lceil x/P_i \rceil C_i^*$ is the maximal requested execution time, and only when it is less than x, the processor is idle. Therefore, F_u implies that the processor cannot be idle in the interval $[0, P_n]$, and therefore any increase of C_i^* will clearly make transaction T_n miss its deadline, causing the scheduling to be infeasible.

Denote $(C_1^*, \cdots, C_n^*)$ by C^* and $(P_1, \cdots, P_n)$ by P, we shall abbreviate $F_u(C_1^*, \cdots, C_n^*, P_1, ..., P_n)$ as $F_u(C^*, P)$. Similarly, let $C^{*\prime}$ denote $(C_1^{*\prime}, ..., C_n^{*\prime})$, P' denote $(P_1', ..., P_n')$, we shall abbreviate $F_u(C_1^{*\prime}, ..., C_n^{*\prime}, P_1, ..., P_n)$ as $F_u(C^{*\prime}, P)$ and $F_u(C_1^{*\prime}, ..., C_n^{*\prime}, P_1', ..., P_n')$ as $F_u(C^{*\prime}, P')$.

Definition 4 $lub(n) \widehat{=} min\{\sum_{i=1}^{n} C_i^*/P_i \mid F_u(C^*, P)\}$.

Lemma 1

$$lub(n) = n(2^{1/n} - 1).$$

The proof of this lemma involves many technical details, and it will be omited here. It follows that $lub(n) \leq lub(k)$ if $k \leq n$, and using this property, we can prove the schedulability condition of PCP-DA theorem.

Theorem 3

$$(ASS \wedge \sum_{i=1}^{n} C_i^*/P_i \leq lub(n)) \Rightarrow (\bigwedge_{i=1}^{n}(T_i.period \Rightarrow \int T_i.run \geq C_i^*)).$$

Proof: For any $1 \leq i \leq n$. The proof is by contradiction. Suppose that there exists $1 \leq k \leq n$, $\ell = P_k$ such that the above does not hold.

$$(1)\ ASS \wedge \sum_{i=1}^{n} C_i^*/P_i \leq lub(n) \wedge \bigvee_{k=1}^{n} \neg(T_k.period \Rightarrow \int T_k.run \geq C_k^*)$$

$$\Rightarrow ASS \wedge \sum_{i=1}^{n} C_i^*/P_i \leq lub(n) \wedge \bigvee_{k=1}^{n} (T_k.period \Rightarrow \int T_k.run < C_k^*) \qquad \text{(PL)}$$

$$\Rightarrow (5.1) \bigvee_{k=1}^{n} \left(\begin{array}{r} ASS \wedge \sum_{i=1}^{n} C_i^*/P_i \leq lub(n) \wedge (\ell = P_k) \wedge [\![\bigvee_{i=1}^{k} T_i.run]\!] \\ \wedge (T_k.period \Rightarrow \int T_k.run < C_k^*) \\ \wedge ((T_k.period \Rightarrow \int T_k.run = 0) \vee (T_k.period \Rightarrow \int T_k.run > 0)) \end{array} \right) \qquad \text{(PL,Usys)}$$

We can divide the proof into two cases according to (5.1). In the first case. There exit $C_k^{*\prime}$ such that $0 < C_k^{*\prime} < C_k^*$ and

$$(2) \left(\begin{array}{l} ASS \wedge \sum_{i=1}^{n} C_i^*/P_i \leq lub(n) \wedge (T_k.period \Rightarrow \int T_k.run < C_k^*) \\ \wedge (\ell = P_k) \wedge [\![\bigvee_{i=1}^{k} T_i.run]\!] \wedge (T_k.period \Rightarrow \int T_k.run > 0) \end{array} \right)$$

$$\Rightarrow \left(\begin{array}{r} ASS \wedge \sum_{i=1}^{n} C_i^*/P_i \leq lub(n) \\ \wedge (\ell = P_k) \wedge [\![\bigvee_{i=1}^{k} T_i.run]\!] \wedge (T_k.period \Rightarrow \int T_k.run = C_k^{*\prime}) \end{array} \right) \qquad \text{(PL)}$$

$$\Rightarrow \forall 0 \le x \le P_k \left(\begin{array}{c} \sum_{i=1}^{n} C_i^*/P_i \le lub(n) \\ \wedge \left(\begin{array}{c} \ell = x = \sum_{i=1}^{k} \int T_i.run \\ \le \sum_{i=1}^{k-1} \lceil \ell/P_i \rceil C_i^* + C_k^{*\prime} \end{array} \right) \\ \frown(\ell = P_k - x) \end{array} \right) \quad \text{(DC4,ITL5,Usys)}$$

$$\Rightarrow \forall 0 \le x \le P_k \left(\begin{array}{c} \sum_{i=1}^{n} C_i^*/P_i \le lub(n) \\ \wedge x \le \sum_{i=1}^{k-1} \lceil x/P_i \rceil C_i^* + C_k^{*\prime} \end{array} \right)$$

$$\Rightarrow \left(\begin{array}{c} \sum_{i=1}^{n} C_i^*/P_i \le lub(n) \\ \wedge F_u(C_1^*, \cdots, C_{k-1}^*, C_k^{*\prime}, P_1, \cdots, P_k) \end{array} \right) \quad \text{(Def of Fu)}$$

$$\Rightarrow \exists 0 < C_k^{*\prime} < C_k^* \left(\begin{array}{c} \sum_{i=1}^{n} C_i^*/P_i \le lub(n) \\ \wedge lub(k) \le \sum_{i=1}^{k-1} C_i^*/P_i + C_k^{*\prime}/P_k \end{array} \right)$$

$$\Rightarrow lub(k) < lub(n) \quad \text{(Lemma 1)}$$

$$\Rightarrow \quad \textbf{(false)}$$

This completes the proof for the first case. In the second case, we have

$$(3) \left(\begin{array}{c} ASS \wedge \sum_{i=1}^{n} C_i^*/P_i \le lub(n) \\ \wedge (\ell = P_k) \wedge \lceil\!\lceil \bigvee_{i=1}^{k} T_i.run \rceil\!\rceil \wedge (T_k.period \Rightarrow \int T_k.run = 0) \end{array} \right)$$

$$\Rightarrow \forall 0 \le x \le P_{k-1} \left(\begin{array}{c} \sum_{i=1}^{n} C_i^*/P_i \le lub(n) \\ \wedge \left(\begin{array}{c} \ell = x = \sum_{i=1}^{k-1} \int T_i.run \\ \le \sum_{i=1}^{k-1} \lceil \ell/P_i \rceil C_i^* \end{array} \right) \\ \frown(\ell = P_k - x) \end{array} \right) \quad \text{(DC4, ITL5, Usys)}$$

$$\Rightarrow \ldots$$

$$\Rightarrow \textbf{false}$$

The proof this case can be done the same way as above and it is omitted here.

$$(4)(ASS \wedge \sum_{i=1}^{n} C_i^*/P_i \leq lub(n)) \wedge \neg(\bigwedge_{i=1}^{n}(T_i.period \Rightarrow \int T_i.run \geq C_i^*)) \Rightarrow \textbf{false} \quad ((1),(2),(3))$$

$$(5)(ASS \wedge \sum_{i=1}^{n} C_i^*/P_i \leq lub(n)) \Rightarrow (\bigwedge_{i=1}^{n}(T_i.period \Rightarrow \int T_i.run \geq C_i^*)) \quad ((4))$$

6. Blocking of PCP-DA in Multiprocessor Environment

We shall show in the following example that PCP-DA in a parallel processing environment may result in a large number of priority inversion. A graphical representation of the example PCP-DA schedule in multiprocessor environment is shown in Figure 1.

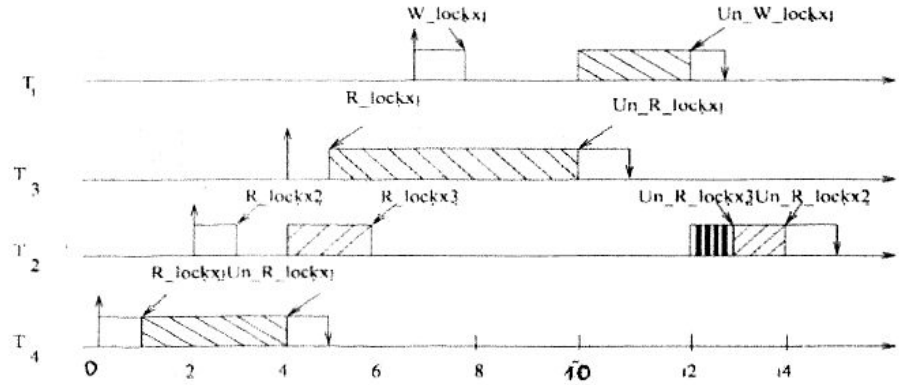

Figure 1: The example PCP-DA schedule in multiprocessor environment

Let system consist of four transactions T_1, T_2, T_3 and T_4 in a two processor environment CPU_1 and CPU_2 where T_1, T_3 and T_2, T_4 executes on CPU_1, CPU_2, respectively. Let the priority of T_1, T_2, T_3 and T_4 be 1, 2, 3 and 4, respectively, where 1 is the highest and 4 is lowest. Suppose that T_1 may write data object x_1 and T_3 and T_4 may read from data object x_1. T_2 can read data objects x_2 and x_3. T_3 can read and write data objects x_2 and x_3. According to the definition ceilings, the write priority ceiling $WPL(x_1)$ is equal to the priority of T_1, i.e., 1. The write priority ceiling $WPL(x_2)$ is equal to 5. The write priority ceiling $WPL(x_3)$ is equal to 2.

At time 0, T_4 arrives and starts execution on CPU_2. At time 1, T_4 read locks x_1 successfully and sets $T_i.sysceil(x_1)$ equal to $WPL(x_1)(=1)$. Since, the priority of T_4 is higher than $T_i.sysceil(x_1)(=0)$ (LC2 is **true**). At time 2, T_2 arrives on CPU_2. The read lock request of T_2 on data object x_2 successfully at time 3 and sets $T_i.sysceil(x_2)$ equal to $WPL(x_2)$ (=5), because the priority of T_2 is higher than $WPL(x_2)$ and $x_2 \notin WO(T_2)$(LC3 is **true**). At time 4, T_4 unlocks x_1. At time 4, T_3 arrives and starts execution on CPU_1. At time 5, the read lock request of T_3 on data object x_1 is granted and sets $T_i.sysceil(x_1)$

equal to $WPL(x_1)$ (=1), because the priority of T_3 is higher than $T_i.sysceil(x_2)(= 5)$(LC1 is **true**). However, read lock of T_3 creates a priority inversion for T_2 at time 6 when T_2 issues a read lock request on data object x_3. It is because the priority of T_2 is not higher than $T_i.sysceil(x_1)$ (= 1)(LC1,LC2 is **false**) and $x_3 \in WO(T_3)$(LC3, LC4 is **false**). The read lock request of T_2 on data object x_3 can not be granted until T_3 and T_1 unlock x_1, although T_3 may inherit the priority of T_2 when the blocking happens at time 6. We must point out that if T_2 later tries to issue another read-lock on some other data object after at time 12, then T_2 may another priority inversion again if some other lower priority transaction happens to have read-locked x_1 at that time. The above example shows that the number of priority inversion for transactions scheduled by PCP-DA may be more than one when there are more than one processor in the system. It is definitely not acceptable for many time critical systems. Therefore, we should improve PCP-DA to be suitable for a multiprocessor environment.

7. Conclusion

In this paper, we apply a formal model of real time database systems in previous our work to specify and verify formally the Priority Ceiling Protocol with Dynamic Adjustment of Serialization Order in Real Time Databases using the proof system of DC. We devise a worst case schedulability analysis for PCP-DA which provides a better schedulability condition compared to R/WPCP. We show that the number of priority inversion for transactions scheduled by PCP-DA may be more than one in a multiprocessor environment. These frameworks can be used in the future for specifying many other issues of RTDBS, we easily can specify and verify for a set of the concurrency control protocols in RTDBS.

Acknowledgements. The authors would like to thank Dr. Dang Van Hung for his kind helps.

References

1. Doan Van Ban, Ho Van Huong, Duration Calculus and Application. Proccedings of Hanoi University of Sciences, National University of Vietnam, Nov, 2000.
2. Doan Van Ban, Ho Van Huong. A Formal Specification of the Read/Write Priority Ceiling Protocol in Real Time Databases. Proccedings of National Information Technology, Hai Phong, June, 2001.
3. Doan Van Ban, Ho Van Huong, Serializability of Two Phase Locking Concurrency Control Protocol in Real Time Database. · *Jounal of Computer Science and Cybernetics*, No 3(17), 2001.
4. Doan Van Ban, Nguyen Huu Ngu, Ho Van Huong, Concurrency control protocol in Real Time Databases. Proccedings of Institute of Information Technology, Nov, 2001.
5. Azer Bestavros, Kwei-Jay Lin and Sang Hyuk Son. *Real-Time Database Systems: Issues and Applications*. Kluwer Academic Publishers, 1997.
6. Philip Chan and Dang Van Hung. Duration Calculus Specification of Scheduling

for Tasks with Shared Resources UNU/IIST Report No. 44, UNU/IIST, P.O. Box 3058, Macau, June, 1995.

7. M.R. Hansen and Zhou Chaochen. Duration Calculus: Logical Foundations. *Formal Aspects of Computing*, 1997, 9:283-330.
8. Dang Van Hung. Real-time Systems Development with Duration Calculus: an Overview. UNU/IIST Report No. 255, UNU/IIST, P.O. Box 3058, Macau, June, 2002.
9. Ho Van Huong and Dang Van Hung. Modelling Real-Time Database Systems in Duration Calculus UNU/IIST Report No.260 , UNU/IIST, P.O. Box 3058, Macau, August, 2002.
10. Ho Van Huong. A Formal Specification of the Abort-Oriented Concurrency Control for Real Time Databases in Duration Calculus. *Jounal of Computer Science and Cybernetics*, No 1(16), 2003.
11. Kwok-wa Lam, Sang H.Son, Sheung-Lun Hung, and Zhiwei Wang. Scheduling Transactions With Stringent Real Time Constraints. *Information Sytstems*, 2000, 25(6):431–452.
12. Kam-Yiu Lam and Tei-Wei Kuo. *Real-Time Database Systems: Architecture and Techniques*. Kluwer Academic Publishers, 2001.
13. Ekaterina Pavlova and Dang Van Hung. A Formal Specification of the Concurrency Control in Real Time Database UNU/IIST Report No. 152, UNU/IIST, P.O. Box 3058, Macau, January, 1999.
14. Lui Sha ,Ragunathan Rajkumar and John P.Lehoczky. A Real Time Locking Protocol. *IEEE Transactions on computers* 40(7):793-800, 1991.