# Concurrency Control for Embedded Real-Time Systems with Tighter Time Bounds Using STM and Closed Nesting

Mohammed Elshambakey
*ECE Dept*
*VT, 24060*
*Blacksburg, USA*
*shambake@vt.edu*

Binoy Ravindran
*ECE Dept*
*VT, 24060*
*Blacksburg, USA*
*binoy@vt.edu*

***Abstract*—We consider closed nested software transactional memory (STM) concurrency control for embedded multicore real-time software, and present a modified version of FBLT contention manager called closed nested FBLT. We upper bound transactional retries and task response times under closed nested FBLT, and identify when closed nested FBLT is a more appropriate alternative to non-nested FBLT.**

***Keywords*-Software transactional memory (STM), real-time contention manager**

## I. Introduction

This demo file is intended to serve as a "starter file" for IEEE conference papers produced under LaTeX using IEEEtran.cls version 1.7 and later.

All manuscripts must be in English. These guidelines include complete descriptions of the fonts, spacing, and related information for producing your proceedings manuscripts. Please follow them and if you have any questions, direct them to the production editor in charge of your proceedings at Conference Publishing Services (CPS): Phone +1 (714) 821-8380 or Fax +1 (714) 761-1784.

Embedded systems sense physical processes and control their behavior, typically through feedback loops. Since physical processes are concurrent, computations that control them must also be concurrent, enabling them to process multiple streams of sensor input and control multiple actuators, all concurrently while satisfying time constraints.

The de facto standard for concurrent programming is the threads abstraction, and the de facto synchronization abstraction is locks. Lock-based concurrency control has significant programmability, scalability, and composability challenges [?]. Transactional memory (TM) is an alternative synchronization model for shared memory objects that promises to alleviate these difficulties. With TM, code that read/write shared objects is organized as *memory transactions*, which execute speculatively, while logging changes made to objects. Two transactions conflict if they access the same object and at least one access is a write. When that happens, a contention manager (CM) [?] resolves the conflict by aborting one and allowing the other to commit, yielding (the illusion of) atomicity. Aborted transactions are restarted, after rolling back the changes. In addition to a simple programming model, TM provides performance comparable to lock-free approach, especially for high contention and read-dominated workloads (see an example TM system's performance in [?]), and is composable [?]. TM has been proposed in hardware, called HTM, and in software, called STM, with the usual tradeoffs: HTM has lesser overhead, but needs transactional support in hardware; STM is available on any hardware.

Given STM's programmability, scalability, and composability advantages, it is a compelling concurrency control technique also for multicore embedded real-time software. However, this requires bounding transactional retries, as real-time threads, which subsume transactions, must satisfy time constraints. Retry bounds under STM are dependent on the CM policy at hand.

Past real-time CM research has proposed resolving transactional contention using dynamic and fixed priorities of parent threads, resulting in Earliest Deadline CM (ECM) and Rate Monotonic CM (RCM) [?], [?], [?], which are intended to be used with global EDF (G-EDF) and global RMS (G-RMS) multicore real-time schedulers [?], respectively. In particular, [?] shows that ECM and RCM achieve higher schedulability – i.e., greater number of task sets meeting their time constraints – than lock-free synchronization only under some ranges for the maximum atomic section length. That range is significantly expanded with the Length-based CM (LCM) in [?], increasing the coverage of STM's timeliness superiority. ECM, RCM, and LCM suffer from transitive retry and cannot handle multiple objects per transaction efficiently. These limitations are overcome with the Priority with Negative value and First access CM (PNF) [?], [?]. However, PNF requires a-priori knowledge of all objects accessed by each transaction. This significantly limits programmability, and is incompatible with dynamic STM implementations [?]. Additionally, PNF is a centralized CM, which increases overheads and retry costs, and has a complex implementation. First Bounded, Last Timestamp CM (or FBLT) [?], in contrast to PNF, does not require a-priori knowledge of objects accessed by transactions. Moreover, FBLT allows each transaction to access multiple objects with shorter transitive retry cost than ECM, RCM and LCM. Additionally, FBLT is a decentralized CM and does not use locks in its implementation. Implementation of FBLT is also simpler than PNF.

Previous CMs consider only non-nested transactions. Nested transactions [?], [?] can be: 1) flat: If a child transaction aborts, then the parent transaction also aborts. If a child commits, no effect is taken until the parent commits. Modifications made by the child transaction are only visible to the parent until the parent commits, after which they are externally visible. 2) Closed: Similar to flat nesting, except that if a child transaction conflicts, it is aborted and retried, without aborting the parent, potentially improving concurrency over flat nesting. 3) Open: If a child transaction commits, its modifications are immediately externally visible, releasing memory isolation of objects used by the child, thereby potentially improving concurrency over closed nesting. However, if the parent conflicts after the child commits, then compensating actions are executed to undo the actions of the child, before retrying the parent and the child.

We introduce closed-nested FBLT that extends original FBLT to closed nested transactions (Section V). We present the motivation for introducing closed nesting into FBLT (Section IV). We establish closed-nested FBLT's retry and response time upper bounds under G-EDF and G-RMA schedulers (Section VI). We also identify the conditions under which closed-nested FBLT is a better alternative to non-nested FBLT (Section VII).

## II. Related Work

The main focus of previous research on real-time STM's CMs is on non-nested transactions. Non-nested STM concurrency control for real-time systems has been previously studied in [?], [?], [?], [?], [?], [?], [?], [?], [?], [?].

[?] proposes a restricted version of STM for uniprocessors. [?] bounds response times in distributed systems with STM synchronization. They consider Pfair scheduling, limit to small atomic regions with fixed size, and limit transaction execution to span at most two quanta. [?] presents real-time scheduling of transactions and serializes transactions based on deadlines. [?] proposes real-time HTM. [?] assumes that the worst case conflict between atomic sections of different tasks occurs when the sections are released at the same time. [?] upper bounds retries and response times for ECM with G-EDF, and identify the tradeoffs with locking and lock-free protocols. Similar to [?], [?] also assumes that the worst case conflict between atomic sections occurs when the sections are released simultaneously. The ideas in [?] are extended in [?], which presents three real-time CM designs. [?] presents the ECM and RCM contention managers, and upper bounds transactional retries and task response times under them. [?] presents the LCM contention manager, and upper bounds its transactional retries and task response times under the G-EDF and G-RMA schedulers. [?], [?] restrict transactions to access only one object. [?] presents the PNF contention manager, which allows transactions to access multiple objects and avoids the consequent transitive retry effect. However, PNF requires a-priori knowledge of the objects accessed by each transaction, which is not always possible. [?] presents the FBLT contention manager. In contrast to PNF, FBLT does not require prior knowledge of required objects by each transaction. FBLT premits multiple objects per transaction.

Past research in distributed real-time database systems provide different concurrency control algorithms for nested transactions [?], [?], [?], [?]. [?] presents a priority assignment scheme called *flexible high reward for nested transactions (FHRN)* and a real-time concurrency control protocol for nested transactions in DRTDBS called *two phase locking with high priority for nested transactions (2PL-HPN)*. FHRN considers number of parameters in priority assignment for each nested transaction. These parameters include deadline, assigned value, slack time, communication time, number of leaves and nesting depth. 2PL-HPN is based on two phase locking protocol with high priority scheme (2PL-HP) **[?]** with priority inheritance and conditional restart. Under 2PL-HPN, higher priority transaction access objects first. Higher priority transaction is blocked if its slacking time allows current lower priority transactions to complete. Priority inheritance is used to prevent priority inversion. Transactions belonging to the same family should not abort each other. Priority inheritance is used among transactions of the same family.

[?] presents *multiversion optimistic concurrency control for nested transactions* (MVOCC-NT) protocol for mobile real-time nested transactions in mobile broadcast environments. In MVOCC-NT, each transaction is assigned a timestamp at its start. Each object can have multiple versions. Each version has two flags indicating largest timestamps of transactions created and read that object version. Each transaction has a validating interval that is initialized to $[0,\infty]$. For each conflict, lower or upper bound of the validating interval is modified. Thus, serialization is maintained without unnecessary restarts. If the validating interval is empty, then transaction must be restarted. [?] presents 2PL-NT-HP concurrency control protocol and 2PC-RT-NT commit protocol. Priorities are assigned to each sub-transaction by adding the sub-transaction level to its parent priority. The base priority of the outermost transaction is assigned by EDF. Under 2PL-RT-HP, conflicts are resolved based on higher priority. (Sub)transactions in the same family do not abort each other. Lower priority (sub)transactions are allowed to complete if higher priority (sub)transactions have enough slack. Priority inheritance is used to prevent priority inversion. [?] extends [?] to deal with impreciseness in nested transactions. (Sub)transactions can be essential (with firm deadlines) and non-essential (without firm deadline).

[?] introduces a new model for nested transactions based on Serialization Graphs (SG). An SG is serializable if it is acyclic. Usually, for nested SG, each parent and its children are represented by one node. So, any conflict between a (sub)transaction and another (sub)transaction appears as a conflict between the roots of both (sub)transactions. In the new model, only "leaf" transactions are "legal" transactions. A transaction "virtually commits" when it completed its work, all transaction it reads from are committed, and some of its children have not yet committed. When a transaction "virtually commits", it cannot be aborted. A transaction actually commits when it virtually committed and all its children have committed. To avoid deadlock in the new model, a (sub)transaction that writes to a parent of sub-transactions, extends edges to these sub-transactions. [?] discusses some issues related to nested transactions in DRTBS like propagating deadline from parents to children. One way is the absolute deadline propagation, in which deadline of the child is the same as the parent except for communication delay. Another way is the normal propagation approach, in which remaining time of the parent is taken into consideration. The average priority propagation assigns the same priority to all sub-transactions in the same family. The assigned priority value is the average of the sum of normal (sub)transactions' priorities.

[?] presents a Reactive Transactional Scheduler (RTS) to schedule closed nested transactions in distributed systems. [?] assumes a data flow model in which objects move to transactional nodes. RTS compares between aborting or enqueuing a parent transaction. The choice depends on execution length of aborted transaction and contention level.

## III. Preliminary

We consider a multiprocessor system with $m$ identical processors and $n$ sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$. The $k^{th}$ instance (or job) of a task $\tau_i$ is denoted $\tau_i^k$. Each task $\tau_i$ is specified by its worst case execution time (WCET) $c_i$, its minimum period $T_i$ between any two consecutive instances, and its relative deadline $D_i$, where $D_i = T_i$. Job $\tau_i^j$ is released at time $r_i^j$ and must finish no later than its absolute deadline $d_i^j = r_i^j + D_i$. Under a fixed priority scheduler such as G-RMA, $p_i$ determines $\tau_i$'s (fixed) priority and it is constant for all instances of $\tau_i$. Under a dynamic priority scheduler such as G-EDF, a job $\tau_i^j$'s priority, $p_i^j$, differs from one instance to another. A task $\tau_j$ may interfere with task $\tau_i$ for a number of times during an interval $L$, and this number is denoted as $G_{ij}(L)$.

*Shared objects.* A task may need to read/write shared, in-memory data objects while it is executing any of its atomic sections (transactions), which are synchronized using STM. The set of atomic sections of task $\tau_i$ is denoted $s_i$. $s_i^k$ is the $k^{th}$ atomic section of $\tau_i$. Each object, $\theta$, can be accessed by multiple tasks. The set of distinct objects accessed by $\tau_i$ is $\Theta_i$ without repeating objects. The set of atomic sections used by $\tau_i$ to access $\theta$ is $s_i(\theta)$, and the sum of the lengths of those atomic sections is $len(s_i(\theta))$. $s_i^k(\theta)$ is the $k^{th}$ atomic section of $\tau_i$ that accesses $\theta$. $s_i^k$ can access one or more objects in $\theta_i$. So, $s_i^k$ refers to the transaction itself, regardless of the objects accessed by the transaction. We denote the set of all accessed objects by $s_i^k$ as $\Theta_i^k$. While $s_i^k(\theta)$ implies that $s_i^k$ accesses an object $\theta \in \Theta_i^k$, $s_i^k(\underline{\Theta})$ implies that $s_i^k$ accesses a set of objects $\Theta = \{\theta \in \Theta_i^k\ \}$. $\bar{s_i^k} = \bar{s_i^k}(\Theta)$ refers only once to $s_i^k$, regardless of the number of objects in $\Theta$. So, $|\bar{s_i^k}(\Theta)|_{\forall\theta\in\Theta} = 1$. $s_i^k(\theta)$ executes for a duration $len(s_i^k(\theta))$. $len(s_i^k) = len(s_i^k(\theta)) = len(s_i^k(\Theta)) = len(s_i^k(\Theta_i^k))$ The set of tasks sharing $\theta$ with $\tau_i$ is denoted $\gamma_i(\theta)$.

The maximum-length atomic section in $\tau_i$ that accesses $\theta$ is denoted $s_{i_{max}}(\theta)$, while the maximum one among all tasks is $s_{max}(\theta)$, and the maximum one among tasks with priorities lower than that of $\tau_i$ is $s_{max}^i(\theta)$. $s_{max}^i(\Theta_h^i) = max\{s_{max}^i(\theta) : \forall\theta \in \Theta_h^i\}$.

*STM retry cost.* If two or more atomic sections conflict, the CM will commit one section and abort and retry the others, increasing the time to execute the aborted sections. The increased time that an atomic section $s_i^p(\theta)$ will take to execute due to a conflict with another section $s_j^k(\theta)$, is denoted $W_i^p(s_j^k(\theta))$. If an atomic section, $s_i^p$, is already executing, and another atomic section $s_j^k$ tries to access a shared object with $s_i^p$, then $s_j^k$ is said to "interfere" or "conflict" with $s_i^p$. The job $s_j^k$ is the "interfering job", and the job $s_i^p$ is the "interfered job".

Due to *transitive retry* [?], [?], an atomic section $s_i^k(\Theta_i^k)$ may retry due to another atomic section $s_j^l(\Theta_j^l)$, where $\Theta_i^k \cap \Theta_j^l = \emptyset$. $\Theta_i^*$ denotes the set of objects not accessed directly by atomic sections in $\tau_i$, but can cause transactions in $\tau_i$ to retry due to transitive retry. $\Theta_i^{ex}(=\Theta_i + \Theta_i^*)$ is the set of all objects that can cause transactions in $\tau_i$ to retry directly or through transitive retry. $\Theta_i^{k^{ex}}$ is the subset of objects in $\Theta_i^{ex}$ that can cause direct or transitive conflict to $s_i^k$. $\gamma_i^*$ is the set of tasks that accesses objects in $\Theta_i^*$. $\gamma_i^{ex}(=\gamma_i + \gamma_i^*)$ is the set of all tasks that can directly or indirectly (through transitive retry) cause transactions in $\tau_i$ to abort and retry. $\gamma_i^k$ is the set of tasks that can directly cause $s_i^k$ to abort and retry. $\gamma_i^{k^{ex}}$ is the set of tasks that can directly or indirectly (through transitive retry) cause $s_i^k$ to abort and retry.

The total time that a task $\tau_i$'s atomic sections have to retry over $T_i$ is denoted $RC(T_i)$. The additional amount of time by which all interfering jobs of $\tau_j$ increases the response time of any job of $\tau_i$ during $L$, without considering retries due to atomic sections, is denoted $W_{ij}(L)$.

*Nested transactions:* A nested transaction $s_i^k$ is represented as a *tree* or a *transactional family*, where $s_i^k$ is the *root* or *Top Tree Level (TTL)*. Any (sub)transaction that contains other (sub)transactions is called a *parent*, while a sub-transaction is called a *child*. Thus, a sub-transaction can be a parent and a child at the same time. Root does not have a parent. A sub-transaction without children is called a *leaf*. $s^k_{i*}$ can be any (sub)transaction lying in the tree whose root is $s^k_i$, including $s^k_i$ itself, down to the leaves. Root of the tree to which $s^k_{i*}$ belongs is denoted as $R(s^k_{i*})$. $s^k_{i*}$ begins after start of $s^k_i$ by at least $\nabla^k_{i*}$. Set of leaves of the tree whose root is $s^k_{i*}$ is denoted as $L(s^k_{i*})$. Parent of $s^k_{i*}$ is denoted as $Par(s^k_{i*})$. Set of direct children of $s^k_{i*}$ is denoted as $Ch(s^k_{i*})$. Set of children of any (sub)transaction $s^k_{i*}$, including grand children down to leaves, are called *descendants* of $s^k_{i*}$, $Des(s^k_{i*})$. Set of parents and grand parents of $s^k_{i*}$ up to the root are called *ancestors*, $Anc(s^k_{i*})$. A (sub)transaction $s^k_{i*}$ and its descendants are represented as a set of (sub)transactions $\{s^k_{i*}\}$. Thus, $\{s^k_{i*}\}$ is a *tree* or a *transactional family* whose root is $s^k_{i*}$. (Sub)transactions in $\{s^k_{i*}\}$ are ordered by their start time relative to $s^k_{i*}$ with ties broken arbitrarily. Thus, if $s^k_{i1*}(s^k_{i2*})$ begins after start of $s^k_{i*}$ by $\nabla^k_{i1*}(\nabla^k_{i2*})$ respectively, and $\nabla^k_{i1*} < \nabla^k_{i2*}$, then $s^k_{i1*}$ comes before $s^k_{i2*}$ in $\{s^k_{i*}\}$. The $a^{th}$ direct child of $s^k_{i*}$ is $s^k_{i*-a}$. Thus, $s^k_{i*-a} \in \{s^k_{i*}\}$. The set of $s^k_{i*-a}$ and its descendants is a subset of $\{s^k_{i*}\}$ (i.e., $\{s^k_{i*-a}\} \subseteq \{s^k_{i*}\}$). A parent precedes its children in order, thus $s^k_{i*} < s^k_{i*-a}$. $len(s^k_{i*})$ includes lengths of all its children, as any $s^k_{i*-a}$ executes inside $s^k_{i*}$. $\Theta^k_{i*}$ represents set of objects accessed by $s^k_{i*}$. $\Theta^k_{i*-a}$ represents set of objects accessed by $s^k_{i*-a}$. $\Theta^k_{i*-a}$ may contain objects not in $\Theta^k_{i*}$. Thus, if $\theta$ is accessed by both $s^k_{i*}$ and $s^k_{i*-a}$, then $\theta \in \Theta^k_{i*}, \Theta^k_{i*-a}$. If $\theta$ is accessed by $s^k_{i*}$ but not $s^k_{i*-a}$, then $\theta \in \Theta^k_{i*}$ but $\theta \notin \Theta^k_{i*-a}$. If $\theta$ is accessed by $s^k_{i*-a}$ but not by $s^k_{i*}$, then $\theta \in \Theta^k_{i*-a}$, but $\theta \notin \Theta^k_{i*}$. The set of objects accessed by all (sub)transactions in $\{s^k_{i*}\}$ is $\{\Theta^k_{i*}\} = \cup_{\forall s^k_{i*} \in \{s^k_{i*}\}} \Theta^k_{i*}$. $\Theta^{k^{ex}}_i$ is the same for any $s^k_{i*} \in \{s^k_i\}$ (i.e., $\Theta^{k^{ex}}_i = \Theta^{k^{ex}}_{i*}, \forall s^k_{i*} \in \{s^k_i\}$). This is because children abort with their parents under closed nesting. Thus, objects accessed by $Par\{s^k_{i*}\}$ are included in $\Theta^{k^{ex}}_{i*}$. $p(s^k_{i*})$ is priority of (sub)transaction $s^k_{i*}$. Generally, we assume $p(s^k_{i*}), \forall s^k_{i*} \in \{s^k_{i*}\}$ is the same unless otherwise stated.

*Conflicting (sub)Transaction:* $CT(s^k_{i*}, s^l_{j*}, a)$ is the $a^{th}$ subtransaction in $\{s^l_{j*}\}$ that can conflict directly with any (sub)transaction in $\{s^k_{i*}\}$. Let the set of objects accessed by $CT(s^k_{i*}, s^l_{j*}, a)$ be $\Theta 1$. By definition of $CT(s^k_{i*}, s^l_{j*}, a)$, $\Theta 1 \cap \{\Theta^k_{i*}\} \neq \emptyset$. Let $CT(s^k_{i*}, s^l_{j*}, a)$ begins after start of $s^l_{j*}$ by $\nabla 1$. Let $CT(s^k_{i*}, s^l_{j*}, b)$ begins after start of $s^l_{j*}$ by $\nabla 2$. If $a < b$, then $\nabla 1 < \nabla 2$. The first (sub)transaction in $\{s^k_{i*}\}$ that can be interfered by $CT(s^k_{i*}, s^l_{j*}, a)$ is defined as *Inverse Conflicting (sub)Transaction* $CT^{-1}(s^k_{i*}, s^l_{j*}, a)$. $CT^{ex}(s^k_{i*}, s^l_{j*}, a)$ is the same as $CT(s^k_{i*}, s^l_{j*}, a)$ except that $\Theta 1 \cap \Theta^{k^{ex}}_{i*} \neq \emptyset$. By definition of $\Theta^{k^{ex}}_{i*}$, $CT^{ex}(s^k_{i*}, s^l_{j*}, a)$ may not directly conflict with any (sub)transaction in $\{s^k_{i*}\}$. Thus, there will be no definition for *Inverse Conflicting (sub)Transaction* in transitive retry.

## IV. Motivation

When a child transaction aborts in closed and open nesting, the parent transaction does not have to abort. By proper organization of objects within (sub)transactions, only child transactions need to be aborted. Thus, retry cost can be reduced.

Behaviour of CM, such as PNF [?], can make nesting useless. PNF requires a priori knowledge of accessed objects within transactions. Only the first $m$ non-conflicting transactions are allowed to execute concurrently and non-preemptively. Thus, PNF makes no use of nesting. A non-preemptive parent transaction will enforce its children to be non-preemptive also. So, no other (sub)transaction can abort a non-preemptive transaction, nor its children. If PNF is modified such that only objects accessed by parent is known a priori, then deadlock is inevitable. This is illustrated by the following example.

**Example 1**: Assume $s^k_i(\theta_1)$ is a parent transaction that has a child sub-transaction $s^k_{i-1}(\theta_2)$. $s^k_i$ accesses only $\theta_1$, and $s^k_{i-1}(\theta_2)$ access only $\theta_2$. $s^l_j(\theta_2)$ is another parent transaction that has a child $s^l_{j-1}(\theta_1)$. $s^l_j(\theta_2)$ accesses only $\theta_2$, and $s^l_{j-1}(\theta_1)$ accesses only $\theta_1$. Initially, $s^k_i(\theta_1)$ and $s^l_j(\theta_2)$ can execute non-preemptively in parallel. $s^k_{i-1}(\theta_2)$ conflicts with the non-preemptive transaction $s^l_j(\theta_2)$. So, $s^k_{i-1}(\theta_2)$ has to wait until $s^l_j(\theta_2)$ finishes. $s^l_{j-1}(\theta_1)$ conflicts with the non-preemptive transaction $s^k_i(\theta_1)$. Thus, $s^l_{j-1}(\theta_1)$ waits until $s^k_i(\theta_1)$ finishes. But $s^k_i(\theta_1)$ waits until its child $s^k_{i-1}(\theta_2)$ finishes, and $s^l_j(\theta_2)$ waits until its child $s^l_{j-1}(\theta_1)$ finishes. This cycle represents a deadlock.

FBLT [?], by definition, depends on LCM. LCM, in turn, depends on ECM (RCM) for G-EDF (G-RMA), respectively. Experimental results show superiority of FBLT over LCM, ECM and RCM [?]. Thus, we extend FBLT to closed-nested FBLT to reduce retry cost than the non-nested FBLT.

## V. Closed-nested FBLT

Closed-nested FBLT depends on FBLT which in turn depends on LCM [?]. Thus, LCM is extended to closed-nested LCM for closed nested transactions. In all the following Claims, it will be assumed that $\Pi(x) = \begin{cases} x & \text{, if } x > 0 \\ 0 & \text{, Otherwise} \end{cases}$.

### A. Closed-nested LCM

Design of closed-nested LCM depends on Claim 1.

**Claim 1.** *Let $\{s^k_i\}$ and $\{s^l_j\}$ be conflicting nested transactions where $p(s^l_j) > p(s^k_i)$. $\alpha^{kl}_{ij}$ is the result of interference of $s^k_i$ by $s^l_j$ as defined by LCM. Using the same $\alpha^{kl}_{ij}$ to resolve conflict of all (sub)transactions $s^k_{i*} \in \{s^k_i\}$ interfered by any $s^l_{j*} \in \{s^l_j\}$ is preferable than using different $\alpha^{kl}_{i*j*}$ for different (sub)transactions (i.e., $s^k_{i*}$ aborts if $s^l_{j*}$ conflicts with $s^k_{i*}$ before $s^k_i$ reaches $\alpha^{kl}_{ij}$ of $len(s^k_i)$).*

*Proof:*

Let there be only one $\alpha^{kl}_{ij}$ for any (sub)transaction $s^k_{i*} \in \{s^k_i\}$ interfered by $s^l_{j*} \in \{s^l_j\}$. $\{s^k_i\}$ can be represented as a single non-nested transaction, and so is $\{s_j^l\}$. Thus, using the same $\alpha_{ij}^{kl}$ for nested transactions will not increase retry cost nor blocking time over non-nested case. On the other hand, using different $\alpha_{i*j*}^{kl}$ for different (sub)transactions can increase retry cost and/or blocking time as given by the following cases:

1) Each (sub)transaction $s_{i*}^k \in \{s_i^k\}$ has its own $\alpha_{i*j}^{kl}$ due to interference by $s_j^l$. Assume $s_i^k$ has a child $s_{i-1}^k$. Assume $s_j^l$ has $\alpha_{ij}^{kl}$ ($\alpha_{i-1j}^{kl}$) when it interferes with $s_i^k(s_{i-1}^k)$ respectively. $p_j^l > p_i^k, p_{i-1}^k$. Assume conflict between $s_j^l$ and $s_{i-1}^k$ is detected before $s_j^l$ conflicts with $s_i^k$ and before $s_i^k$ reaches $\alpha_{ij}^{kl}$ from its execution length. Let $s_{i-1}^k$ has passed $\alpha_{i-1j}^{kl}$ of its execution length. By LCM, $s_j^l$ will abort and retry. In non-nested case, $s_j^l$ will not abort because $s_i^k$ has not reached $\alpha_{ij}^{kl}$ from its execution length yet. Thus, blocking time due to lower priority tasks in closed nesting is increased over non-nested case. The closer $s_{i*}^k$ gets to the start of $s_i^k$ and the shorter $len(s_{i*}^k)$ becomes, the more blocking time of $s_j^l$ suffers. This problem is avoided if there is only one $\alpha_{ij}^{kl}$ for any $s_{i*}^k \in \{s_i^k\}$ interfered by $s_j^l$.
2) Using the previous scenario. Let $s_{i*}^k$ finishes before $s_i^k$ reaches $\alpha_{ij}^{kl}$ of its execution length. $s_j^l$ aborts and retries due to $s_{i*}^k$. After $s_{i*}^k$ finishes, $s_i^k$ will abort and retry because it has not reached $\alpha_{ij}^{kl}$ of its execution length yet. $s_{i*}^k$ will also retry with $s_i^k$. Thus, previous abortion of $s_j^l$ is useless. This problem is avoided if there is only one $\alpha_{ij}^{kl}$ for any $s_{i*}^k \in \{s_i^k\}$ interfered by $s_j^l$.
3) $s_j^l$ detects conflict with $s_{i-1}^k$ before $s_j^l$ conflicts with $s_i^k$. Conflict between $s_j^l$ and $s_{i-1}^k$ occurs before $s_{i-1}^k$ reaches $\alpha_{i-1j}^{kl}$ from its execution length, but after $s_i^k$ passes $\alpha_{ij}^{kl}$ from its execution length. By LCM, $s_{i-1}^k$ aborts and retries in favor of $s_j^l$. After that, $s_j^l$ detects conflict with $s_i^k$. By LCM, $s_j^l$ aborts and retries in favor of $s_i^k$. Thus, retry cost of $s_i^k$ is increased by the retry cost of $s_{i-1}^k$ which would not have happened in non-nested case. This problem is avoided if there is only one $\alpha_{ij}^{kl}$ for any $s_{i*}^k \in \{s_i^k\}$ interfered by $s_j^l$.
4) Previous cases consider multiple (sub)transactions in $\{s_i^k\}$ interfered by the same $s_j^l$. Now, we consider one transaction $s_i^k$ interfered by multiple (sub)transactions in $\{s_j^l\}$. Let $s_j^l$ and $s_{j-1}^l$ interfere with $s_i^k$. $\alpha_{ij}^{kl}$ and $\alpha_{ij-1}^{kl}$ result from interference of $s_i^k$ by $s_j^l$ and $s_{j-1}^l$. By definition of LCM, $\alpha_{ij-1}^{kl} len(s_i^k) \geq \alpha_{ij}^{kl} len(s_i^k)$. Assume $s_{j-1}^l$ conflicts with $s_i^k$ when $s_i^k$ reaches some point in $\alpha_{ij-1}^{kl} - \alpha_{ij}^{kl}$ of its execution length. $s_j^l$ has not conflicted with $s_i^k$ yet. According to $\alpha_{ij-1}^{kl}$, $s_i^k$ should abort and retry, but according to $\alpha_{ij}^{kl}$, $s_j^l$ is the one to abort and retry. In non-nested case, $s_j^l$ would abort and retry. Thus, using $\alpha_{ij-1}^{kl}$ increases retry cost of $s_i^k$ over non-nested case. On the other hand, if $s_{j-1}^l$ aborts, blocking time of $s_j^l$ due to $s_i^l$ would be the same as in non-nested case. The increase in retry cost is avoided if there is only one $\alpha_{ij}^{kl}$ for any $s_i^k$ interfered by multiple (sub)transactions in $\{s_j^l\}$.

Claim follows.

■

Closed-nested LCM is shown in Algorithm 1. Closed-

**Algorithm 1:** closed-nested LCM

**Data**: $s_{i*}^k \in \{s_i^k\} \rightarrow$ interfered (sub)transaction.
$s_{j*}^l \in \{s_j^l\} \rightarrow$ interfering (sub)transaction.
$\psi \rightarrow$ predefined threshold $\in [0,1]$.
$\varepsilon_i^k \rightarrow$ remaining execution length of $\{s_i^k\}$
**Result**: which (sub)transaction of $s_{i*}^k$ or $s_{j*}^l$ aborts

1 **if** $p_i^k > p_j^l$ **then**
2     $s_{j*}^l$ aborts;
3 **else**
4     $c_{ij}^{kl} = len(s_j^l)/len(s_i^k)$;
5     $\alpha_{ij}^{kl} = ln(\psi)/(ln(\psi) - c_{ij}^{kl})$;
6     $\alpha = \left(len(s_i^k) - \varepsilon_i^k\right)/len(s_i^k)$;
7     **if** $\alpha \leq \alpha_{ij}^{kl}$ **then**
8         $s_{i*}^k$ aborts;
9     **else**
10         $s_{j*}^l$ aborts;
11     **end**
12 **end**

nested LCM uses the remaining length of $\{s_i^k\}$ when it is interfered, as well as $len(s_j^l)$, to decide which (sub)transaction must be aborted. If $p_i^k > p_j^l$, then $s_{j*}^l$ would be the (sub)transaction to abort because of its higher priority, and $R(s_{i*}^k)$ started before $R(s_j^l)$ (step 2). Otherwise, $c_{ij}^{kl}$ is calculated (step 4) to determine whether it is worth aborting $s_{i*}^k$ in favour of $s_{j*}^l$, because $len(s_j^l)$ is relatively small compared to the remaining execution length of $\{s_i^k\}$.

### B. Design of Closed-Nested FBLT

Design of closed-nested FBLT depends on Claim 2

**Claim 2.** *Under closed nesting FBLT, it is preferable to use one $\delta_i^k$ for all (sub)transactions $s_{i*}^k \in \{s_i^k\}$ than to use different $\delta_{i*}^k$ for each $s_{i*}^k$ (i.e., $\delta_i^k$ includes abortion of any $s_{i*}^k \in \{s_i^k\}$).*

*Proof:* Nested (sub)transactions $\{s_{i*}^k\}$ can be represented as one non-nested transaction of length $len(s_i^k)$ and accessed objects $\{\Theta_i^k\}$. Thus, upper bound on retry cost using one $\delta_i^k$ for all $s_{i*}^k \in \{s_i^k\}$ will be the same as in non-nested case. Using different $\delta_{i*}^k$ for each $s_{i*}^k \in \{s_i^k\}$ imposes extra constraints on nested transactions compared to non-nested transactions as illustrated by the following cases:

1) Let $s_i^k$ has only one child $s_{i-a}^k$. In non-nested case, $\{s_i^k\}$ is treated as one transaction with $\delta\_max_i^k$ maximum abort times. Under closed nesting FBLT, assume $s_i^k$ has

$\delta_i^k$ maximum abort times, and $s_{i-a}^k$ has $\delta_{i-a}^k$ maximum abort times. $s_{i-a}^k$ can be non-preemptive before $s_i^k$. After $s_{i-a}^k$ commits (relatively to its parent because of closed nesting), $s_i^k$ may abort, enforcing $s_{i-a}^k$ to retry again. Then it will be useless to let a child be non-preemptive while one of its ancestors is not. If $s_{i-a}^k$ is aborted $\delta_{i-a}^k$ times for each time $s_i^k$ aborts, then the total abort times of $s_i^k$ and $s_{i-a}^k$ will be $\delta_i^k\left(1+\delta_{i-a}^k\right)$. Thus, for nested transactions to reach the same abort number in non-nested case, $\delta_i^k\left(1+\delta_{i-a}^k\right)$ should equal $\delta\_max_i^k$. Thus, $\delta_i^k$ is inversely related to $\delta_{i-a}^k$. As nesting level increases, $\delta_{i*}^k$ for any $s_{i*}^k$ will be inversely related to maximum abort number of $Par(s_{i*}^k)$ and $Ch(s_{i*}^k)$. This problem is avoided by using one $\delta_i^k$ for all $s_{i*}^k \in \{s_i^k\}$.

2) Following the previous case except that $s_i^k$ becomes non-preemptive before $s_{i-a}^k$. Then either $s_i^k$ enforces $s_{i-a}^k$ to be non-preemptive, or $s_i^k$ waits for $s_{i-a}^k$ to be non-preemptive after $s_{i-a}^k$ aborts for $\delta_{i-a}^k$. If $s_i^k$ enforces $s_{i-a}^k$ to be non-preemptive, then $\delta_{i-a}^k$ is useless. Otherwise, if $s_i^k$ waits for $s_{i-a}^k$ to become non-preemptive, then $s_i^k$ is delayed by $s_{i-a}^k$. Thus, retry cost of $s_i^k$ is increased over non-nested case. This problem is avoided by using one $\delta_i^k$ for all $s_{i*}^k \in \{s_i^k\}$.

Claim follows.

■

Algorithm 2 illustrates closed-nested FBLT. Each nested transaction $s_i^k$- including its descendants- can be aborted during $T_i$ for at most $\delta_i^k$ times. $\eta_i^k$ records the number of times $\{s_i^k\}$- including its descendants- has already been aborted up to now. If $\{s_i^k\}$ and $\{s_j^l\}$ have not joined the $m$_set yet, then they are preemptive transactions. Preemptive transactions resolve conflicts using closed-nested LCM (step 2). Thus, closed-nested FBLT defaults to closed-nested LCM when no nested transaction reaches its $\delta$. If only one of the nested transactions is in the $m$_set, then the non-preemptive (sub)transaction (the one in $m$_set) aborts the other one (steps 15 to 26). $\eta_i^k$ is incremented each time any $s_{i*}^k \in \{s_i^k\}$ is aborted as long as $\eta_i^k < \delta_i^k$ (steps 5 and 18). Otherwise, the whole $\{s_i^k\}$ is added to the $m$_set and priority of any $s_{i*}^k \in \{s_i^k\}$ is increased to *m_prio* (steps 7 to 9 and 20 to 22). When the priority of $\{s_i^k\}$ is increased to *m_prio*, $\{s_i^k\}$ becomes a non-preemptive transaction. Non-preemptive nested transactions cannot be aborted by other preemptive nested transactions, nor by any other real-time job. The $m$_set can hold at most $m$ concurrent transactions because there are $m$ processors in the system. $r(s_i^k)$ records the time $\{s_i^k\}$ joined the $m$_set (steps 8 and 21). When non-preemptive (sub)transactions conflict together (step 27), the (sub)transaction with the smaller $r()$ commits first (steps 29 and 31). Thus, non-preemptive (sub)transactions are executed in FIFO order of the $m$_set.

**Algorithm 2:** The Closed-nested FBLT Algorithm

**Data**: $s_{i*}^k \in \{s_i^k\}$: interfered (sub)transaction;
$s_{j*}^l \in \{s_j^l\}$: interfering (sub)transactions;
$\delta_i^k$: the maximum number of times $\{s_i^k\}$ can be aborted during $T_i$;
$\eta_i^k$: number of times $\{s_i^k\}$ has already been aborted up to now;
$m$_set: contains at most $m$ non-preemptive nested transactions. $m$ is number of processors;
*m_prio*: priority of any nested transaction in $m$_set. *m_prio* is higher than any priority of any real-time task;
$r(s_i^k)$: time point at which $\{s_i^k\}$ joined $m$_set;
**Result**: (sub)transactions that will abort

1 **if** $\{s_i^k\}, \{s_j^l\} \notin m\_set$ **then**
2     Apply closed-nested LCM [?];
3     **if** $s_{i*}^k$ *is aborted* **then**
4         **if** $\eta_i^k < \delta_i^k$ **then**
5             Increment $\eta_i^k$ by 1;
6         **else**
7             Add $\{s_i^k\}$ to $m$_set;
8             Record $r(s_i^k)$;
9             Increase priority of $\{s_i^k\}$ to *m_prio*;
10         **end**
11     **else**
12         Swap $s_{i*}^k$ and $s_{j*}^l$;
13         Go to Step 3;
14     **end**
15 **else if** $\{s_j^l\} \in m\_set, \{s_i^k\} \notin m\_set$ **then**
16     Abort $s_{i*}^k$;
17     **if** $\eta_i^k < \delta_i^k$ **then**
18         Increment $\eta_i^k$ by 1;
19     **else**
20         Add $\{s_i^k\}$ to $m$_set;
21         Record $r(s_i^k)$;
22         Increase priority of $\{s_i^k\}$ to *m_prio*;
23     **end**
24 **else if** $\{s_i^k\} \in m\_set, \{s_j^l\} \notin m\_set$ **then**
25     Swap $s_{i*}^k$ and $s_{j*}^l$;
26     Go to Step 15;
27 **else**
28     **if** $r(s_i^k) < r(s_j^l)$ **then**
29         Abort $s_{j*}^l$;
30     **else**
31         Abort $s_{i*}^k$;
32     **end**
33 **end**

## VI. Retry cost with closed nesting

**Claim 3.** *Let $s_{j*}^l$ be the first descendant of $s_j^l$ that can conflict with some (sub)transaction in $\{s_i^k\}$ (i.e., $CT(s_i^k, s_j^l, 1)$). $s_{j*}^l$ begins after start of $s_j^l$ by $\nabla_{j*}^l$. Let $s_{i*}^k$ be the first descendant of $s_i^k$ that can be interfered by some (sub)transaction in $\{s_j^l\}$. $s_{i*}^k$ is the same first (sub)transaction in $\{s_i^k\}$ that can conflict with some (sub)transaction in $\{s_j^l\}$ (i.e., $CT(s_j^l, s_i^k, 1)$). $s_{i*}^k$ begins after start of $s_i^k$ by $\nabla_{i*}^k$. $s_j^l$ starts after $s_i^k$ by $\triangle$. If $\triangle < 0$, then $s_j^l$ starts before $s_i^k$. No other (sub)transaction can conflict with any (sub)transaction in $\{s_j^l\}$ or $\{s_i^k\}$. Under closed nesting, $s_i^k$ aborts and retries* *due to $s_j^l$ for*

$$RC0_{ij}^{kl}=\begin{cases}\Pi\left(len\left(s_j^l\right)-\nabla_{i*}^k+\triangle\right) & , \textit{if} \begin{array}{l}\triangle \geq \nabla_{i*}^k - len\left(s_j^l\right)\\ \triangle \leq len\left(s_i^k\right)-\nabla_{j*}^l\end{array}\\ 0 & , \textit{Otherwise}\end{cases} \tag{1}$$

*$RC0_{ij}^{kl}$ is upper bounded by $len\left(s_j^l+s_i^k\right)-\nabla_{j*}^l-\nabla_{i*}^k$.*

*Proof:*

$s_{j*}^l$ is the first (sub)transaction in $\{s_j^l\}$ that conflicts with any (sub)transaction in $\{s_i^k\}$. Thus, before $s_{j*}^l$ begins, $\{s_i^k\}$ detects no conflict with $\{s_j^l\}$, and $\{s_i^k\}$ and $\{s_j^l\}$ can run concurrently. Due to closed nesting, effects of $s_{j*}^l$ appear only after $s_j^l$ commits. $s_{j*}^l$ is the first descendant of $s_j^l$ that can conflict with any (sub)transaction in $\{s_i^k\}$. So, there is no $s_{j*+1}^l \neq s_{j*}^l$ such that $\left(\Theta_{j*+1}^l \cap \{\Theta_i^k\} \neq \emptyset\right) \wedge \left(\nabla_{j*+1}^l < \nabla_{j*}^l\right)$. Thus, during $len(s_j^l)-\nabla_{j*}^l$, $s_i^k$ aborts and retries due to $s_{j*}^l$ and/or later sub-transactions in $\{s_j^l\}$. $s_{i*}^k$ is the first (sub)transaction in $\{s_i^k\}$ that can be interfered by any (sub)transaction in $\{s_j^l\}$. Due to closed nesting, parent (sub)transaction does not abort due to child abortion. Thus, the first descendant of $s_i^k$ that needs to abort and retry is $s_{i*}^k$. For $\{s_j^l\}$ to conflict with $\{s_i^k\}$, $s_j^l$ should start no less than $\nabla_{i*}^k - len\left(s_j^l\right)$ relative to start of $s_i^k$. Otherwise, $\{s_j^l\}$ would have finished before $s_{i*}^k$ starts and there will be no conflict between $\{s_j^l\}$ and $\{s_i^k\}$. $s_{j*}^l$ must start before end of $s_i^k$. Otherwise, $s_i^k$ would have finished before $s_{j*}^l$ starts, and there will be no conflict between $\{s_i^k\}$ and $\{s_j^l\}$. The worst conflict scenario between $\{s_i^k\}$ and $\{s_j^l\}$ occurs when $s_{j*}^l$ starts just before $s_i^k$ commits. Thus, $s_i^k$ aborts and retries for at most $len(s_j^l)-\nabla_{j*}^l+len(s_i^k)-\nabla_{i*}^k$. If $\{s_i^k\}$ and $\{s_j^l\}$ are overlapping, then the overlapping of conflicting sub-transactions in $\{s_i^k\}$ and $\{s_j^l\}$ should be subtracted from the maximum retry cost, otherwise the overlapping part will be summed twice. The conflicting overlapping part of $\{s_j^l\}$ during $\{s_i^k\}$ is $len(s_i^k)-\triangle-\nabla_{j*}^l$. Retry cost must be positive. Otherwise, there is no conflict. That is why $\Pi(x)$ is used. Claim follows.

∎

**Claim 4.** *Under closed nested FBLT, any (sub)transaction $s_{i*}^k \in \{s_i^k\}$ uses closed nested LCM to resolve conflicts before $s_i^k$ becomes non-preemptive. Under closed nested LCM, $\{s_i^k\}$ aborts and retries due to only one interference of higher priority $\{s_j^l\}$ by at most*

$$RC1_{ij}^{kl}=\begin{cases}\Pi\left(len\left(s_j^l\right)-\nabla_{i*}^k+\triangle\right) & , \begin{array}{l}\triangle \geq \nabla_{i*}^k - len\left(s_j^l\right)\\ \triangle \leq min\begin{pmatrix} len\left(s_i^k\right)-\nabla_{j*}^l\\ \alpha_{ij}^{kl} len\left(s_i^k\right)\end{pmatrix}\end{array}\\ 0 & , \textit{Otherwise}\end{cases} \tag{2}$$

*where $\nabla_{j*}^l$ is the time interval between starts of $s_j^l$ and $CT(s_i^k,s_j^l,1)$. $\nabla_{i*}^k$ is the time interval between starts of $s_i^k$ and $CT(s_j^l,s_i^k,1)$.*

*Proof:*

Under closed nested LCM, any (sub)transaction $s_{i*}^k \in \{s_i^k\}$ aborts and retries due to $s_{j*}^l \in \{s_j^l\}$ if $s_j^l$ begins before $s_i^k$ reaches $\alpha_{ij}^{kl}$ of its execution length (i.e., $\alpha_{ij}^{kl} len(s_i^k)$). Following Claim 3, retry cost of $\{s_i^k\}$ due to interference of $\{s_j^l\}$ can be calculated by (1) conditioning that $\triangle \leq \alpha_{ij}^{kl} len\left(s_i^k\right)$. Lower bound of $\triangle$ given in (1) should be less than $\alpha_{ij}^{kl}$ (i.e., $\nabla_{i*}^k - len\left(s_j^l\right) \leq \alpha_{ij}^{kl}$). Otherwise, $s_j^l$ starts after $s_i^k$ reaches $\alpha_{ij}^{kl}$ of its execution length, and $\{s_i^k\}$ will not abort due to $\{s_j^l\}$. Claim follows.

∎

**Claim 5.** *Under closed nested FBLT, any (sub)transaction $s_{i*}^k \in \{s_i^k\}$ uses closed nested LCM to resolve conflicts before $s_i^k$ becomes non-preemptive. Under closed nested LCM, $\{s_j^l\}$ aborts and retries due to lower priority $\{s_i^k\}$ if $s_i^k$ has passed $\alpha_{ij}^{kl}$ of its execution length. Under closed nested LCM, $\{s_j^l\}$ aborts and retries due to lower priority $\{s_i^k\}$ by at most*

$$RC2_{ji}^{lk}=\begin{cases}\Pi\left(len\left(s_i^k\right)-\nabla_{j*}^l-\triangle\right) & , \begin{array}{l}\triangle \geq \alpha_{ij}^{kl} len\left(s_i^k\right)\\ \triangle \leq len\left(s_i^k\right)-\nabla_{j*}^l\end{array}\\ 0 & , \textit{Otherwise}\end{cases} \tag{3}$$

*where $\nabla_{j*}^l$ is the time interval between starts of $s_j^l$ and $CT(s_i^k,s_j^l,1)$.*

*Proof:*

Proof is the same as proof of Claim 3 except for:

$s_{i*}^k$ can overlap with $s_{j*}^l$ for $\triangle + len\left(s_j^l\right)-\nabla_{i*}^k$. Thus, $\{s_j^l\}$ can abort and retry due to $\{s_i^k\}$ for $len\left(s_j^l+s_i^k\right)-\nabla_{j*}^l-\nabla_{i*}^k-\triangle-len\left(s_j^l\right)+\nabla_{i*}^k = len\left(s_i^k\right)-\nabla_{j*}^l-\triangle$. $s_j^l$ must start not less than $\alpha_{ij}^{kl} len\left(s_i^k\right)$, and $\alpha_{ij}^{kl}\left(s_i^k\right) \leq len\left(s_i^k\right)-\nabla_{j*}^l$. Otherwise, $\{s_j^l\}$ will not abort and retry due to $\{s_i^k\}$ by definition of LCM. Claim follows.

∎

**Claim 6.** *Let* $\omega 1_i^j = \begin{cases}\left\lceil \frac{T_i}{T_j}\right\rceil & , \textit{G-EDF}\\ \left\lceil \frac{T_i}{T_j}\right\rceil+1 & , \textit{G-RMA}\end{cases}$

$\omega 2_i = \begin{cases} 1 & , \textit{G-EDF} \\ 2 & , \textit{G-RMA} \end{cases}$. *Let*

$$RC3_i^k = \sum_{\forall \tau_j \in \gamma_i^{k^{ex}}} \omega 1_i^j \sum_{\substack{\forall \{s_j^l\} \\ \{\Theta_j^l\} \cap \Theta_i^{k^{ex}} \neq \emptyset}} \max_{\substack{\forall \{s_x^y\}, \forall \tau_x \\ p_i^k \le p_x^y < p_j^l}} \left(RC1_{ix}^{ky}\right) + \omega 2_i \sum_{\forall \tau_j \in \gamma_i^k} \sum_{\substack{\forall \{s_j^l\} \\ \{\Theta_j^l\} \cap \Theta_i^{k^{ex}} \neq \emptyset}} \max\left(RC2_{ij}^{kl}\right)$$

*where $RC1_{ix}^{ky}$ is calculated by (2) and $RC2_{ij}^{kl}$ is calculated by (3). Under closed nested FBLT, the maximum retry cost of any transactional family $\{s_i^k\} \in \tau_i^x$ before $s_i^k$ becomes non-preemptive due to other conflicting (sub)transactions is upper bounded by*

$$RC4_i^k = \begin{cases} RC3_i^k & , \textit{if } \left\lceil \frac{RC3_i^k}{len\left(s_i^k\right)} \right\rceil < \delta_i^k \\ \delta_i^k len\left(s_i^k\right) & , \textit{Otherwise} \end{cases} \tag{5}$$

*Proof:*

$\omega 1_i^j$ is maximum number of higher priority jobs $\tau_j^h$ that can be released during $T_i$. $\omega 2_i$ is number of lower priority jobs $\tau_j^l$ that can be released during $T_i$. Under G-EDF, only one instance of each $\tau_j$ can be of lower priority than current job $\tau_i^f$. So, remaining jobs of $\tau_j$ is the maximum number of higher priority jobs released during $T_i$ (i.e., $\left\lceil \frac{T_i}{T_j} \right\rceil$). Under G-RMA, all jobs of $\tau_j$ are of higher priority than any job of $\tau_i$ if $p_j > p_i$ (i.e., $\left\lceil \frac{T_i}{T_j} \right\rceil + 1$). Also, all jobs of $\tau_j$ are of lower priority than any job of $\tau_i$ if $p_j < p_i$ (i.e., $\left\lceil \frac{T_i}{T_j} \right\rceil + 1$). Under G-RMA with implicit deadlines, $T_j > T_i$ if $p_j < p_i$. Thus, maximum number of lower priority jobs $\tau_j^l$ that can be released during $T_i$ is 2.

Under closed nested FBLT, any (sub)transaction $\{s_i^k\}$ uses closed nested LCM to resolve conflicts before $\{s_i^k\}$ is aborted $\delta_i^k$ times. If maximum abort number of $\{s_i^k\}$ is less than $\delta_i^k$, then retry cost of $\{s_i^k\}$ is calculated by (4). Equation (4) is derived from proof of Claim 5 (Claim 8) in [**?**] for G-EDF (G-RMA) respectively, except for two points: 1) Claims 5 and 8 in [**?**] calculates retry cost for all transactions in any job $\tau_i^x$, while (4) calculates retry cost for only one $\{s_i^k\}$. Thus, Claims 4 and 5 are used to calculate (4). 2) In Claim 5 in [**?**], each higher priority transaction $s_i^k$ can be aborted only once by any lower priority transaction. In closed nested LCM, due to multiple objects per transaction and nested transactions, each $\{s_i^k\}$ can be aborted by all directly and transitively conflicting lower priority transactions. By definition of closed nested FBLT, $\{s_i^k\}$ retries for at most $RC3_i^k$ before $\{s_i^k\}$ becomes non-preemptive if maximum abort number of $\{s_i^k\}$ is less than $\delta_i^k$ (i.e., $\left\lceil \frac{RC3_i^k}{len\left(s_i^k\right)} \right\rceil < \delta_i^k$). Otherwise, $\{s_i^k\}$ aborts and retries for at most $\delta_i^k len\left(s_i^k\right)$ before $\{s_i^k\}$ becomes non-preemptive. Claim follows.

■

**Claim 7.** *Let $\{s_i^k\}$ be a nested transaction under closed nested FBLT. Let $RC5_{ij}^{kl} = RC0_{ij}^{kl}$ as defined in Claim 3 except that $\nabla_{i*}^k - len\left(s_j^l\right) \le \triangle \le min\left(0, len\left(s_i^k\right) - \nabla_{j*}^l\right)$. Let $\chi_i^k$ be set of nested transactions $\{s_j^l\}$ that can conflict directly or transitively with $\{s_i^k\}$ arranged in non-increasing order of $RC0_{ij}^{kl}$. Each $\{s_j^l\} \in \chi_i^k$ belongs to a distinct task $\tau_j$. $\chi_i^k(a)$ is the $a^{th}$ nested transaction in $\chi_i^k$. $\chi_i^k = \left\{\{s_j^l\} | \left(\Theta_i^{k^{ex}} \cap \{\Theta_j^l\} \neq \emptyset\right) \wedge \left(RC5_i^k(a) \ge RC5_i^k(a+1)\right)\right\}$ where $RC5_i^k(a) = RC5_{ij}^{kl} | \chi_i^k(a) = \{s_j^l\}$. The total retry cost of any job $\tau_i^x$ during $T_i$ under closed nested FBLT due to 1) directly and transitively conflicting transactions with any $\{s_i^k\}$. 2) release of higher priority jobs is upper bounded by*

$$RC_i = \sum_{\forall s_i^k} \begin{cases} RC3_i^k & \left\lceil \frac{RC3_i^k}{len\left(s_i^k\right)} \right\rceil < \delta_i^k \\ \delta_i^k len\left(s_i^k\right) + \sum_{\forall \chi_i^k(a), a=1}^{a \le m-1} RC5_{ij}^{kl}(a) & , \textit{otherwise} \end{cases} + RC_{re}(T_i) \tag{6}$$

*$RC3_i^k$ is calculated by Claim 6. $RC_{re}(T_i)$ is the retry cost resulting from release of higher priority jobs which preempt $\tau_i^x$. $RC_{re}(T_i)$ is calculated by as defined by Claim 1 in [**?**].*

*Proof:*

Non-preemptive (sub)transaction $\{s_i^k\}$ resolves conflicts based on the time $\{s_i^k\}$ becomes non-preemptive. Thus, non-preemptive $\{s_i^k\}$ can be interfered at most by $m-1$ nested transactions that precede $\{s_i^k\}$ in the *m_set* as defined in closed nested FBLT. As defined by closed-nested FBLT, nested transactions in the *m*_set are arranged in FIFO order. Thus, if $\{s_j^l\}$ precedes $\{s_i^k\}$ in *m*_set, then $\{s_i^k\}$ must have started as a non-preemptive transaction not before non-preemptive $\{s_j^l\}$. So, $RC0_{ij}^{kl}$ is modified to $RC5_{ij}^{kl}$ to indicate the proper time interval for start of $s_j^l$ relative to $s_i^k$. The $m-1$ nested transactions preceding $\{s_i^k\}$ result in maximum retry cost to $\{s_i^k\}$ (i.e., $\sum_{\forall \chi_i^k(a), a=1}^{a \le m-1} RC5_{ij}^{kl}(a)$). Based on the previous notion and Claims 6, 3 and Claim 1 in [**?**], Claim follows.

■

Any newly released task $\tau_i^x$ can be blocked by $m$ lower priority non-preemptive nested transactions. $\tau_i^x$ has to wait at most for the whole length of a non-preemptive nested transaction. Thus, $D_i$ is independent of nesting. Blocking time of $\tau_i^x$ ($D_i$) due to the longest $m$ lower priority non-preemptive nested transaction is calculated by Claim 3 in [**?**]. Claim 2 in [**?**] is used to calculate response time under closed nested FBLT where $RC_{to}(T_i)$ is calculated by (6).

## VII. Closed Nested vs. Non-Nested FBLT

**Claim 8.** *Schedulability of closed-nested FBLT is better or equal to non-nested FBLT's if the conflicting* *(sub)transactions in each $\{s_i^k\}$ begin lately relative to start of $s_i^k$.*

*Proof:*

Let upper bound on retry cost of any task $\tau_i^x$ during $T_i$ under non-nested FBLT be denoted as $RC_i^{nn}$. $RC_i^{nn}$ is calculated by Claim 1 in [**?**]. Let upper bound on retry cost of any task $\tau_i^x$ during $T_i$ under closed-nested FBTL be denoted as $RC_i^{cn}$. $RC_i^{cn}$ is calculated by (6). Let $D_i$ be the upper bound on blocking time of any newly released task $\tau_i^x$ during $T_i$ due to lower priority jobs. Any newly released task $\tau_i^x$ can suffer $D_i$ blocking time if there are $m$ non-preemptive executing transactions. Thus, $D_i$ is the same for both closed-nested and non-nested FBLT. $D_i$ is calculated by Claim 2 in [**?**] for both closed-nested and non-nested FBLT. For closed-nested FBLT schedulability to be better than schedulability of non-nested FBLT:

$$\sum_{\forall \tau_i} \frac{c_i + RC_i^{cn} + D_i}{T_i} \leq \sum_{\forall \tau_i} \frac{c_i + RC_i^{nn} + D_i}{T_i} \tag{7}$$

$\because$ $D_i$ and $c_i$ are the same for each $\tau_i$ under closed-nested and non-nested FBLT, then (7) holds if:

$$\forall \tau_i, RC_i^{cn} \leq RC_i^{nn}$$

$$\therefore \delta_i^k len\left(s_i^k\right) + \sum_{\forall \chi_i^k(a), a=1}^{a \leq m-1} RC5_{ij}^{kl}(a) \leq \delta_i^k len\left(s_i^k\right) + \sum_{\forall s_{iz}^k \in \Upsilon_i^k} len\left(s_{iz}^k\right)$$

$$\therefore \sum_{\forall \chi_i^k(a), a=1}^{a \leq m-1} RC5_{ij}^{kl}(a) \leq \sum_{\forall s_{iz}^k \in \Upsilon_i^k} len\left(s_{iz}^k\right) \tag{8}$$

where $\Upsilon_i^k$ is the set of at most $m-1$ longest transactions conflicting directly or transitively with $s_i^k$ as defined in Claim 1 in [**?**]. If $\{s_j^l\} = RC5_{ij}^{kl}(a)$, then by definition of $RC5_{ij}^{kl}$, $\triangle = len(s_i^k) - \nabla_{j*}^l$ if $len(s_i^k) - \nabla_{j*}^l < 0$. So, $max\left(RC5_{ij}^{kl}(a)\right) = \Pi\left(len(s_j^l) - \nabla_{i*}^k\right)$. $\therefore$ by substitution in (8)

$$\therefore \sum_{\forall \{s_j^l\} = \chi_i^k(a), a=1}^{a \leq m-1} \Pi\left(len(s_j^l) - \nabla_{i*}^k\right) \leq \sum_{\forall s_{iz}^k \in \Upsilon_i^k} len\left(s_{iz}^k\right) \tag{9}$$

(9) holds as $\nabla_{i*}^k$ increases. Claim follows. ∎

## VIII. Conclusion

Past research on real-time CMs focused on non-nested transactions. Nested transactions can be flat, closed and open. In this paper, we analysed effect of closed nesting over FBLT CM. Analysis shows that retry cost, hence schedulability, can be reduced if conflicting (sub)transactions start lately relative to their roots. Some CMs make no use of nesting due to behaviour of that CM (e.g, under PNF, all non-preemptive transactions are non-conflicting). Experimental evaluation of closed-nested FBLT, compared to non-nested FBLT, will be done in future work. Also, open nesting will be analysed to reveal whether retry cost and schedulability can be more improved than closed and non-nested FBLT.