Location-Aware Cache-Coherence Protocols for Distributed Transactional Contention Management in Metric-Space Networks

Bo Zhang, Binoy Ravindran
ECE Department, Virginia Tech
Blacksburg, VA 24061, USA
{alexzbzb, binoy}@vt.edu

Abstract

Transactional Memory API utilizes contention managers to guarantee that whenever two transactions have a conflict on a resource, one of them is aborted. Compared with multiprocessor transactional memory systems, the design of distributed transactional memory systems is more challenging because of the need for distributed cache-coherence protocols and the underlying (higher) network latencies involved. The choice of the combination of the contention manager and the cache-coherence protocol is critical for the performance of distributed transactional memory systems. We show that the performance of a distributed transactional memory system on metric-space networks is $O(N_i^2)$ for N_i transactions requesting for a single object under the Greedy contention manager and an arbitrary cache-coherence protocol. To improve the performance, we propose a class of location-aware distributed cache-coherence protocols, called LAC protocols. We show that the combination of the Greedy contention manager and an efficient LAC protocol yields an $O(N \log N \cdot s)$ competitive ratio, where N is the maximum number of nodes that request the same object, and s is the number of objects. This is the first such performance bound established for distributed transactional memory contention managers. Our results yield the following design strategy: select a distributed contention manager and determine its performance without considering distributed cache-coherence protocols; then find an appropriate cache-coherence protocol to improve performance.

1. Introduction

Conventional synchronization methods for multiprocessors based on locks and condition variables are inherently non-scalable and failure-prone. Transactional synchronization is considered as an alternative programming model to manage contention in accessing shared resources. A transaction is an explicitly delimited sequence of steps that is executed atomically by a single thread. A transaction ends by either committing (i.e., it takes effect), or by aborting (i.e., it has no effect). If a transaction aborts, it is typically retried until it commits. Transactional API for multiprocessor synchronization, called *Transactional Memories*, have been proposed both for hardware and software [5–7].

Transactions read and write shared objects. Two transactions *conflict* if they access the same object and one access is a write. The transactional approach to *contention management* guarantees atomicity by making sure that whenever a conflict occurs, only one of the transactions involved can proceed.

This paper studies the contention management problem and the *cache-coherence* protocols problem in a distributed system consisting of a network of nodes that communicating with each other by message-passing links. Quantitatively, the goal is to maximize the throughput, measured by minimizing the *makespan* - i.e., the total time needed to complete a finite set of transactions.

Different models of transactions in distributed systems have been studied in the past [10]. Herlihy and Sun identify three competing transaction models: the control flow model, in which data objects are typically immobile, and computations move from node to node via remote procedure call (RPC); the data flow model, where transactions are immobile (running on a single node) and data objects move from node to node; and the hybrid model, where data objects are migrated depending on a number of heuristics such size and locality. These transactional models make different trade-offs. Past work on multiprocessor [8] suggests that the data flow model can provide better performance than the control flow model on exploiting locality, reducing communication overhead and supporting fine-grained synchronization.

Distributed transactional memory differs from multiprocessor transactional memory in two key aspects. First, multiprocessor transactional memory designs extend built-in cache-coherence protocols that are already supported in modern architectures. Distributed systems where nodes linked by communication networks typically do not come with such built-in protocols. A distributed cache-coherence protocol has to be designed. When a transaction attempts to access an object, the cache-coherence protocol must locate the current cached copy of the object, move it to the requesting node's cache, invalidating the old copy. Secondly, the communication costs for distributed cache-coherence protocols to located the copy of an object in distributed systems are orders of magnitude larger than that in multiprocessors and therefore un-negligible. The communication costs are often determined by the different locations of nodes that invoke transactions, as well as that of the performance of the cache-coherence protocol at hand. Such cost must be taken into account in the performance evaluation. Such costs directly affect the system performance e.g., system throughout; number of aborted transactions, etc.

For multiprocessors, the performance of the contention manager A is evaluated by the competitive ratio, which is the ratio of the makespan of A to the makespan of an optimal off-line clairvoyant scheduler OPT. For networked distributed systems, the specific cache-coherence protocol C has to be introduced as a new variable to affect the performance. Hence, we have to evaluate the performance of a distributed transactional memory system by taking into account its contention management algorithm as well as it supporting cache-coherence protocol.

The problem that we study in the paper is the following: what combination of contention managers and cache-coherence protocols can achieve the optimal makespan for distributed transactional memory systems? We answer this question by first selecting a contention manager, which can guarantee a provable performance with the worst possible cache-coherence protocol. In other words, we first determine the worst-case performance provided by a given contention manager. Now, what contention manager should we select? Motivated by the past work on transactional memory on multiprocessors, we consider the Greedy contention manager [4] for distributed transactional memory systems. The *pending commit* property of the Greedy manager is reserved: at any time, the transaction with the highest priority will be executed and will never be aborted by other transactions. This property is crucial for contention managers to guarantee progress.

We establish the competitive ratio of the Greedy manager with an arbitrary cache-coherence protocol for a single object. In the worst case, the competitive ratio is $O(N_i^2)$ where N_i is the number of transactions that request such object. This is due to delay characteristics of the underlying network, where different nodes may try to locate objects from different locations in the network. During this period, lower-priority transactions may locate objects earlier than higher-priority transactions — a phenomenon called *"overtaking"* [10]. Overtaking may result in high penalty on the final makespan. Overtaking is unavoidable in distributed transactional memory systems due to the network topology. Therefore, we need to design an improved cache-coherence protocol that takes into account the overtaking phenomenon and yields a better performance.

We consider metric-space networks, similar to [10]. In metric-space networks, the communication delay between nodes forms a metric. For distributed systems with metric-space networks, we propose a class of distributed cache-coherence protocols with *location-aware* property, called LAC protocols. In LAC protocols, the duration for a transaction requesting node to locate the object is determined by the communication delay between the requesting node and the node that holds the object. The lower communication delay implies lower locating delay. In other words, nodes that are "closer" to the object will locate the object more quickly than nodes that are "further" from the object in the network. We show that the performance of the Greedy manager with LAC protocols is improved. We prove this worst-case competitive ratio and show that LAC is an efficient choice for the Greedy manager to improve the performance of the system.

We make the following contributions in this paper:

1. We identify that the performance of distributed transactional memory systems is determined by two factors: the contention manager and the cache-coherence protocol used. We show that, for a single object, the optimal off-line clairvoyant scheduler for a set of transactions with the ideal cache-coherence protocol visits all nodes along the shortest Hamiltonian path. This is the first such result.

2. We present a proof of the worst-case competitive ratio of the Greedy contention manager with an arbitrary cache-coherence protocol. We show that this ratio can sometimes lead to the worst choice of transaction execution.

3. We present location-aware cache-coherence protocols called LAC protocols. We show that the worst-case performance of the Greedy contention manager with an efficient LAC protocol is improved and predictable. We prove an $O(N \log N \cdot s)$ competitive ratio for the Greedy manager/LAC protocol combination, where N is the maximum number of nodes that request the same object, and s is the number of objects. To the best of our knowledge, we are not aware of any other competitive ratio result on the performance of transactional memory contention managers in distributed systems.

The rest of the paper is organized as follows: In Section 2, we discuss the related work. We present the system model and state the problem formally in Section 3. We study the performance of the Greedy manager with an arbitrary cache-coherence protocol in Section 4, and the performance for the combination of the Greedy manger and LAC protocols in Section 5. The paper concludes in Section 6.

2. Related Work

Contention managers were first proposed in [8], and a comprehensive survey on the subject is due to Scherer and Scott [17]. A major challenge in implementing a contention manager is guaranteeing progress. Using timestamp-based priorities to guarantee progress, the Greedy manager was proposed by Guerraoui *et al.* [4], where a $O(s^2)$ upper bound is given for the Greedy manager on multiprocessors with s being the number of shared objects. Attiya *et al.* [1] formulated the contention management problem as the *non-clairvoyant* job scheduling paradigm and improved the bound of the Greedy manager to $O(s)$.

Transactional memory models have received attention from many researchers, both in hardware and in software (e.g., [9, 12, 16]. These include both purely software systems, and hybrid systems where software and hardware support for transactional memory are used in conjuncture to improve performance.

Despite this large body of work, few papers in the past [2, 10, 14] investigate transactional memory for distributed systems consisting of a network of nodes. Among these efforts, Herlihy and Sun's work [10] calls our attention mostly. They present a *Ballistic* cache-coherence protocol based on hierarchical clustering for tracking and moving up-to-date copies of cached objects, and suggest a finite response time for each transaction request. In contrast, our work focuses on the makespan to finish a set of transactions whose contention is managed by the Greedy manager under certain cache-coherence protocols, and the design of such protocols in metric-space networks.

We prove that finding an optimal off-line algorithm for scheduling transactions in a graph for a single object is equivalent to the Traveling Salesman Path Problem (TSPP) [13], a problem closely related to the Traveling Salesman Problem (TSP) that replaces the constraint of a *cycle* by a *path*. It is well-known that finding such an algorithm is NP-hard. For an undirected metric-space network, where edge lengths satisfy the triangle inequality, the best known approximation ratio is $5/3$ due to Hoogeveen [11]. Among heuristic algorithms of general TSP problems, Rosenkrantz *et. al.* [15] proved a $O(\log n)$ approximation ratio for the *nearest neighbor* algorithm, which is closely related to our makespan analysis of the Greedy manager.

3. System Model

3.1. Metric-Space Network Model

We consider the metric-space network model of distributed systems proposed by Herlihy and Sun in [10].We assume a *complete undirected* graph $G = (V, E)$, where $|V| = n$. The *cost* of an edge $e(i, j)$ is measured by the communication delay of the *shortest path* between two nodes i and j. We use $d(i, j)$ to denote the cost of $e(i, j)$. Thus, $d(i, j)$ forms the metric of the graph and satisfies the triangle inequality. We scale the metric so that 1 is the smallest cost between any two nodes. All n nodes are assumed to becontained in a metric space of diameter $Diam$.

3.2. Transaction Model

We are given a set of $m \geq 1$ transactions $T_1, ..., T_m$ and a set of $s \geq 1$ objects $R_1, ..., R_s$. Since each transaction is invoked on an individual node, we use v_{T_i} to denote the node that invokes the transaction T_i, and $V_T = \{v_{T_1}, ..., v_{T_m}\}$. We use $T_i \prec T_j$ to represent that transaction T_i is issued a higher priority than T_j by the contention manager (see Section 3.3).

Each transaction is a sequence of actions, each of which is an access to a single resource. Each transaction T_j requires the use of $R_i(T_j)$ units of object R_i for one of its actions. If T_j updates R_i, i.e., a write operation, then $R_i(T_j) = 1$. If it reads R_i without updating, then $R_i(T_j) = \frac{1}{n}$, i.e., the object can be read by all nodes in the network simultaneously. When $R_i(T_j) + R_i(T_k) > 1$,T_j and T_k conflict at R_i. We use $v^0_{R_i}$ to denote the node that holds R_i at the start of the system, and $v^j_{R_i}$ to denote the j'th node that fetches R_i. We denote the set of nodes that requires the use of the same object R_i as $V_T^{R_i} := \{v_{T_j} | R_i(T_j) \geq 0, j = 1, ..., m\}$.

An execution of a transaction T_j is a sequence of *timed actions*. Generally, there are four action types that may be taken by a single transaction: *write*, *read*, *commit* and *abort*. When a transaction is started on a node, a cache-coherence protocol is invoked to locate the current copy of the object and fetch it. The transaction then starts with the action and may perform local computation (not involving access to the object) between consecutive actions. A transaction completes either with a commit or an abort. The *duration* of transaction T_j running locally (without taking into account the time of fetching objects) is denoted as τ_i.

3.3 Distributed Transactional Memory Model

We apply a data-flow model proposed in [10] to support the transactional memory API in a networked distributed system. Transactions are immobile (running at a single node) but objects move from node to node. Transaction synchronization is optimistic: a transaction commits only if no other transactions has executed a conflicting access. A *contention manager* module is responsible to mediate between conflicting accesses to avoid deadlock and livelock.

The core of this design is an efficient distributed cache-coherence protocol. A distributed transactional memory system uses a distributed *cache-coherence* protocol for operations in the distributed system. When a transaction attempts to access an object, the cache-coherence protocol must locate the current cached copy of the object, move it to the requesting node's cache, and invalidate the old copy.

We assume each node has a *transactional memory proxy* module to provide interfaces to the application and to proxies at other nodes. This module mainly perform following functions:

Data Object Management: An application informs the proxy to open an object when it starts a transaction. The proxy is responsible for fetching a copy of the object requested by the transaction, either from its local cache or from other nodes. When the transaction asks to commit, the proxy checks whether any object opened by the transaction has been modified by other transactions. If not, the proxy makes the transaction's tentative changes to the object permanent, and otherwise discard them.

Cache-Coherence Protocol Invocation: The proxy is responsible for invoking a cache-coherence protocol when needed. When a new data object is created in the local cache, the proxy calls the cache-coherence protocol to *publish* it in the network. When an object is requested by a read access and not in the local cache, the proxy calls the cache-coherence protocol to *look up* the object and fetch a read-only copy. If it is a write request, the proxy calls the cache-coherence protocol to *move* the object to its local cache.

Contention Management: When another transaction asks for an object used by an active local transaction, the proxy can either abort the local transaction and make the object available, or it can postpone a response to give the local transaction a chance to commit. The decision is made by a globally consistent contention management policy that avoids livelock and deadlock. An efficient contention management policy should guarantee progress: at any time, there exists at least one transaction that proceeds to commit without interrupt. For example, the Greedy manager guarantees that the transaction with the highest priority can be executed without interrupt by arrange a globally consistent priority policy to issue priorities to transactions.

3.4 Problem Statement

We evaluate the performance of the distributed transactional memory system by measuring its *makespan*. Given a set of transactions requiring accessing a set of objects, $makespan(A, C)$ denotes the duration that the given set of transactions are successfully executed under a contention manager A and cache-coherence protocol C.

It is well-known that optimal off-line scheduling of tasks with shared resources is NP-complete [3].While an on-line scheduling algorithm does not know a transaction's resource demands in advance, it does not always make optimal choices. An optimal clairvoyant off-line algorithm, denoted $\textsc{Opt}$, knows the sequence of accesses of the transaction to resources in each execution.

We use $makespan(\textsc{Opt})$ to denote the makespan of the optimal clairvoyant off-line scheduling algorithm with the *ideal* cache-coherence protocol. The ideal cache-coherence protocol exhibits the following property: at any time, the node that invokes a transaction can locate the requested object in the shortest time. We evaluate the combination of a contention manager A and a cache-coherence protocol C by measuring its *competitive ratio.*

Definition 1. *Competitive Ratio ($CR(A, C)$): the competitive ratio of the combination (A, C) of a contention manager A and a cache-coherence protocol C is defined as*

$$CR(A, C) = \frac{makespan(A, C)}{makespan(\textsc{Opt})}.$$

Thus, our goal is to solve the following problem: Given a Greedy contention manager (or "Greedy" for short), what kind of cache-coherence protocol should be designed to make $CR(Greedy, C)$ as small as possible?

4. Makespan of the Greedy Manager

4.1. Motivation and Challenge

The past work on transactional memory systems on multiprocessors motivates our selection of the contention manager. The major challenging in implementing a contention manager is to guarantee progress: at any time, there exists some transaction(s) which will run uninterruptedly until they commit. The Greedy contention manager proposed by Guerraoui *et. al.* [4] satisfies this property. Two non-trivial properties are proved in [4] and [1] for the Greedy manager:

- Every transaction commits within a bounded time
- The competitive ratio of Greedy is $O(s)$ for a set of s objects, and this bound is asymptotically tight.

The core idea of the Greedy manager is to use a globally consistent contention management policy that avoids both livelock and deadlock. For the Greedy manager, this policy is based on the timestamp at which each transaction starts. This policy determines the sequence of priorities of the transactions and relies only on local information, i.e., the timestamp assigned by the local clock. To make the Greedy manager work efficiently, those local clocks are needed to by synchronized. This sequence of priorities is determined at the beginning of each transaction and will not change over time. In other words, the contention management policy *serializes* the set of transactions in a decentralized manner.

At first, transactions are processed greedily whenever possible. Thus, a maximal independent set of transactions that are non-conflicting over their first-requested resources is processed each time. Secondly, when a transaction begins, it is assigned a unique *timestamp* which remains fixed across re-invocations. At any time, the running transaction with highest priority (i.e., the "oldest" timestamp) will neither wait nor be aborted by any other transaction.

The nice property of the Greedy manager for multiprocessors motivates us to investigate its performance in distributed systems. In a network environment, the Greedy manager still guarantees the progress of transactions: the priorities of transactions are assigned when they start. At any time, the transaction with the highest priority (the earliest timestamp for the Greedy manager) never waits and is never aborted by a synchronization conflict.

However, it is much more challenging to evaluate the performance for the Greedy manager in distributed systems. The most significant difference between multiprocessors and distributed systems is the cost for locating and moving objects among processors/nodes. For multiprocessors, built-in cache-coherence protocols are supported by modern architectures and such cost can be ignored compared with the makespan of the set of transactions. For networked distributed systems, the case is completely different. Some cache-coherence protocols are needed for locating and moving objects in the network. Due to the characteristics of the network environment, such cost cannot be ignored. In fact, it is quite possible that the cost for locating and moving objects constitutes the major part of the makespan if communication delays between nodes are sufficiently large. Hence, in order to evaluate the performance of the Greedy manager in distributed systems, the property of supported cache-coherence protocol used must be taken into account.

One unique phenomenon for transactions in distributed systems is the concept of *"overtaking"*. Suppose there are two nodes, v_{T_1} and v_{T_2}, which invoke transactions T_1 and T_2 that require write accesses for the same object. Assume $T_1 \prec T_2$. Due to the locations of nodes in the network, the cost for v_{T_1} to locate the current cached copy of the object may be much larger than that for v_{T_2}. Thus, v_{T_2}'s request may be ordered first and the cached copy is moved to v_{T_2} first. Then v_{T_1}'s request has to be sent to v_{T_2} since the object is moved to v_{T_2}. The *success* or *failure* of an overtaking is defined by its result:

Overtaking Success: If v_{T_1}'s request arrives at v_{T_2} after T_2's commit, then T_2 is committed before T_1.

Overtaking Failure: If v_{T_1}'s request arrives at v_{T_2} before T_2's commit, the contention manager of v_{T_2} will abort the local transaction and send the object to v_{T_1}.

Overtaking failures are unavoidable for transactions in distributed systems because of the unpredictable locations of nodes that invoke transactions or hold objects. Such failures may significantly increase the makespan for the set of transactions. Thus, we have to design efficient cache-coherence protocols to relieve the impact of overtaking failures. We will show the impact of such failures in our worst-case analysis of the transaction makespan.

4.2 $makespan(Greedy, C)$ for a Single Object

We start with the makespan for a set of transactions which require accesses for a single object R_i, denoted by $makespan_i$. It is composed of three parts:

1. Traveling Makespan ($makespan_i^d$): the total time that R_i travels in the network.

2. Execution Makespan ($makespan_i^\tau$): the duration for transactions' executions of R_i, including all successful or aborted executions; and

3. Idle Time (I_i): the time that R_i waits for a transaction request.

We first investigate the makespan for all move requests for R_i by an optimal off-line algorithm OPT with the ideal cache-coherence protocol, denoted by $makespan_i(\text{OPT})$. For the set of nodes $V_T^{R_i}$ that invoke transactions with requests for object R_i, we build a *complete* subgraph $G_i = (V_i, E_i)$ where $V_i = \{V_T^{R_i} \bigcup v_{R_i}^0\}$ and the cost of $e_i(j,k)$ is $d(j,k)$. We use $H(G_i, v_{R_i}^0, v_{T_j})$ to denote the cost of the *minimum cost Hamiltonian path* from $v_{R_i}^0$ to v_{T_j} that visits each node exactly once.Then we have the following lemma:

Lemma 1.

$$makespan_i^d(\text{OPT}) \geq \min_{v_{T_j} \in V_T^{R_i}} H(G_i, v_{R_i}^0, v_{T_j})$$

and

$$makespan_i^\tau(\text{OPT}) \geq \sum_{v_{T_j} \in V_T^{R_i}} \tau_j.$$

Proof. The execution of the given set of transactions with the minimum makespan schedules each job exactly once, which implies that R_i only have to visit each node in $V_T^{R_i}$ once. In this case, the node travels along a Hamiltonian path in G_i starting from $v_{R_i}^0$. Hence, we can lower-bound the traveling makespan by the cost of the minimum cost Hamiltonian path. On the other hand, object R_i is kept by node v_{T_j} for a duration at least τ_j for a successful commit. The execution makespan is lower-bounded by the sum of τ_j. The lemma follows. □

Now we focus on the makespan of the Greedy manager with a given cache-coherence protocol C. We postpone the selection of cache-coherence protocol to the next section. Given a subgraph G_i, we define its *priority Hamiltonian path* as follows:

Definition 2. Priority Hamiltonian Path*: the priority Hamiltonian path for a subgraph G_i is a path which starts from $v^0_{R_i}$ and visits each node from the lowest priority to the highest priority.*

Specifically, the priority Hamiltonian path is $v^0_{R_i} \rightarrow v_{T_{N_i}} \rightarrow v_{T_{N_i-1}} ... \rightarrow v_1$, where $N_i = |V^{R_i}_T|$ and $v_1 \prec v_2 \prec ... \prec v_{T_{N_i}}$. We use $H^p(G_i, v^0_{R_i})$ to denote the cost of the priority Hamiltonian path for G_i.

The following theorem gives the upper bound of $makespan_i(Greedy, C)$

Theorem 2.

$$makespan^d_i(Greedy, C) \leq 2N_i \cdot H^p(G_i, v^0_{R_i})$$

and

$$makespan^\tau_i(Greedy, C) \leq \sum_{1 \leq j \leq N_i} j \cdot \tau_j.$$

Proof. We analyze the worst-case traveling and execution makespans of the Greedy manager. At any time t of the execution, let set $A(t)$ contains nodes whose transactions have been successfully committed, and let set $B(t)$ contains nodes whose transactions have not been committed. We have $B(t) = \{b_i(t) | b_1(t) \prec b_2(t) \prec ...\}$. Hence, R_i must be held by a node $r_t \in A(t)$.

Due to the property of the Greedy manager, the transaction requested by $b_1(t)$ can be executed immediately and will never be aborted by other transactions. However, this request can be overtook by other transactions if they are closer to $r(t)$. In the worst case, the transaction requested by $b_1(t)$ is overtook by all other transactions requested by nodes in B, and every overtaking is failed. In this case, the only possible path that R_i travels is $r(i) \rightarrow b_{|B(t)|}(t) \rightarrow b_{|B(t)|-1}(t) \rightarrow ... \rightarrow b_1(t)$. The cost of this path is composed of two parts: the cost of $r(i) \rightarrow b_{|B(t)|}(t)$ and the cost of $b_{|B(t)|}(t) \rightarrow b_{|B(t)|-1}(t) \rightarrow ... \rightarrow b_1(t)$. We can prove that each part is at most $H^p(G_i, v^0_{R_i})$ by triangle inequality (note that G_i is a metric completion graph). Hence, we know that the worst traveling cost for a transaction execution is $2H^p(G_i, v^0_{R_i})$. The first part of the theorem is provided.

The second part can be proved directly. For any transaction T_j, it can be overtook at most $N_i - j$ times. In the worst case, they are all overtaking failures. Hence, the worst execution cost for T_j's execution is $\sum_{j \leq k \leq N_i} \tau_k$. By summing them over all transactions, the second part of the theorem follows.

□

Now we have the following corollary:

Corollary 3.

$$CR^d_i(Greedy, C) = O(N_i^2) \quad (1)$$

and

$$CR^\tau_i(Greedy, C) = O(N_i). \quad (2)$$

Proof. We can directly get Equation 2 by Theorem 2. To prove Equation 1, we only have to prove

$$H^p(G_i, v^0_{R_i}) \leq N_i \cdot \min_{v_{T_j} \in V^{R_i}_T} H(G_i, v^0_{R_i}, v_{T_j}).$$

In fact, we can prove this equation by showing that the cost of the longest path between any two nodes in G_i is less than or equal to the cost of the minimum cost Hamiltonian path. Then the corollary follows. □

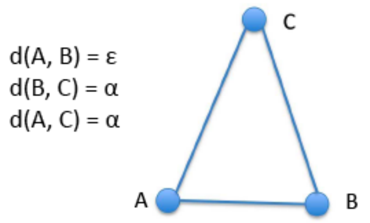

Figure 1. Example 1: a 3-node network

Example 1: in the following example, we show that for the *Ballistic protocol* [10], the upper bound in Equation 1 is asymptotically tight, i.e., the worst-case traveling makespan for a transaction execution is the cost of the *longest* Hamiltonian path.

Consider a network composed of 3 nodes A, B and C in Figure 1. Based on the Ballistic protocol, a 3-level directory hierarchy is built, shown in Figure 2. Suppose $\epsilon << \alpha$. Nodes i and j are connected at level l if and only if $d(i,j) < 2^{l+1}$. A maximal independent set of the connectivity graph is selected with members as leaders of level l. Therefore, at level 0, all nodes are in the hierarchy. At level 1, A and C are selected as leaders. At level 2, C is selected as the leader (also the root of the hierarchy).

We assume that an object is created at A. According to the Ballistic protocol, a *link path* is created: $C \rightarrow A \rightarrow A$, which is used as the directory to locate the object at A. Suppose there are two transactions T_B and T_C invoked on B and C, respectively. Specifically, we have $T_B \prec T_C$.

Now nodes B and C have to locate the object in the hierarchy by probing the link state of the leaders at each level. For node C, it doesn't have to probe at all because it has a no-null link to the object. For node B, it starts to probe the link state of the leaders at level 1. In the worst case, T_C arrives at node A earlier than T_B, and the link path is redirected as $C \rightarrow C \rightarrow C$ and the object is moved to node C. Node B probes a non-null link after the object has been moved, and T_B is sent to node C. If T_C has not been committed, then T_C is aborted and the object is sent to node B.

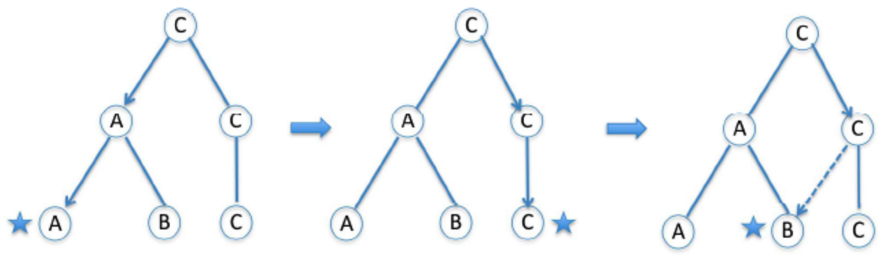

Figure 2. Example 1: link path evolvement of the directory hierarchy built by Ballistic protocol

In this case, the traveling makespan to execute T_B is $d(A, C) + d(C, B) = 2\alpha$, which is the longest Hamiltonian path starting from node A. On the other hand, the optimal traveling makespan to execute T_B and T_A is $d(A, B) + d(B, C) = \epsilon + \alpha$. Hence, the worst-case traveling makespan to execute T_B is asymptotically the number of transactions times the cost of the optimal traveling makespan to execute all transactions.

Remarks: Corollary 3 upper-bounds the competitive ratio of the traveling and execution makespans of the Greedy manager with any given cache-coherence protocol C. It describes the worst case property of the Greedy manager without taking into account the property of any specific cache-coherence protocol. In the worst case, the competitive ratio of the traveling makespan of the Greedy manager is $O(N_i^2)$ for a single object without considering any design of cache-coherence protocols. Hence, we can design a cache-coherence protocol to improve the performance of competitive ratio. A class of cache-coherence protocols designed to lower the $O(N_i^2)$ bound is introduced in the following section.

5. Location-Aware Cache-Coherence Protocol

5.1 Cache Responsiveness

To implement transactional memory in a distributed system, a distributed cache-coherence protocol is needed: when a transaction attempts to read or write an object, the cache-coherence protocol must locate the current cached copy of the object, move it to the requesting node's cache and invalidate the old copy.

Definition 3. Locating Cost *($\delta^C(i, j)$): in a given graph G that models a metric-space network, the locating cost is the time needed for a transaction running on a node i for successfully locating an object held by node j (so that it can read or write the object), under a cache-coherence protocol C.*

Note that the locating cost doesn't include the cost to move the object since we assume the object is moved via the shortest path after being located. The locating cost is network-specific, depending on the topology of G and the locations of i and j in the network. On the other hand, it also depends on the specific cache-coherence protocol C.

The cache-coherence protocol has to be responsive so that every transaction commits within a bounded time. We prove that for the Greedy manager, a cache-coherence protocol is responsive if and only if $\delta^C(i, j)$ is bounded for any G that models a metric-space network.

Let $V_T^{R_i}(T_j) = \{v_{T_k} | v_{T_k} \prec v_{T_j}, v_{T_k}, v_{T_j} \in V_T^{R_i}\}$ for any graph G. Let $\Delta^C[V_T^{R_i}(T_j)] = \max_{v_{R_i} \in V_T^{R_i}} \delta^C(i, j)$ and $D[V_T^{R_i}(T_j)] = \max_{v_{R_i} \in V_T^{R_i}(T_j)} d(v_{R_i}, v_{T_j})$. We have the following theorem.

Theorem 4. *A transaction T_j's request for object R_i with the Greedy manager and cache-coherence protocol C is satisfied within time*

$$|V_T^{R_i}(T_j)| \cdot \{\Delta^C[V_T^{R_i}(T_j)] + D[V_T^{R_i}(T_j)] + \tau_j\}.$$

Proof. The worst case of response time for T_j's move request of object R_i happens when T_j's request overtakes each of the transaction that has a higher priority. Then the object is moved to v_{T_j} and the transaction is aborted just before its commit. Thus, the *penalty time* for an overtaking failure is $\delta^C(i, j) + d(v_{R_i}, v_{T_j}) + \tau_j$, where $v_{R_i} \in V_T^{R_i}(T_j)$. The overtaking failure can happen at most $|V_T^{R_i}(T_j)|$ times until all transactions that have higher priority than T_j commit. The lemma follows.

□

Theorem 4 shows that for a set of objects, the responsiveness for a cache-coherence protocol is determined by its locating cost. Formally, we use *stretch* as a metric to evaluate the responsiveness of a cache-coherece protocol.

Definition 4. *$Stretch(C)$: the stretch of a cache-coherence protocol C is the maximum ratio of the locating cost to the communication delay:*

$$Stretch(C) = \max_{i,j \in V} \frac{\delta^C(i, j)}{d(i, j)}.$$

Apparently, the ideal cache-coherence protocol's stretch is 1. In the practical design of a cache-coherence protocol, we need to lower its stretch as much as possible. For example, the Ballistic protocol provides a constant stretch, which is asymptotically optimal .

5.2. Location-Aware Cache-Coherence Protocols

We now define a class of cache-coherence protocols which satisfy the following property:

Definition 5. Location-Aware Cache-Coherence Protocol*: In a given network G that models a metric-space network, if for any two edges $e(i_1, j_1)$ and $e(i_1, j_1)$ such that $d(i_1, j_1) \geq d(i_1, j_1)$, there exists a cache-coherence protocol C which guarantees that $\delta^C(i_1, j_1) \geq \delta^C(i_2, j_2)$, then C is location-aware. The class of such protocols are called location-aware cache-coherence protocols or LAC protocols.*

By using a LAC protocol, we can significantly improve the competitive ratio of traveling makespan of the Greedy manager, when compared with Equation 1. The following theorem gives the upper bound of $CR_i^d(Greedy, LAC)$.

Theorem 5.

$$CR_i^d(Greedy, LAC) = O(N_i \log N_i)$$

Proof. We first prove that the traveling path of the worst-case execution for the Greedy manager to finish a transaction T_j is equivalent to the *nearest neighbor path* from $v_{R_i}^0$ that visits all nodes with lower priorities than T_j.

Definition 6. *Nearest Neighbor Path: In a graph G, the nearest neighbor path is constructed as follows [15]:*

1. Starts with an arbitrary node.

2. Find the node not yet on the path which is closest to the node last added and add the edge connecting these two nodes to the path.

3. Repeat Step 2 until all nodes have been added to the path.

The Greedy manager guarantees that, at any time, the highest-priority transaction can execute uninterrupted. If we use a sequence $\{v_{R_i}^1 \prec, ..., \prec v_{R_i}^{N_i}\}$ to denote these nodes in the priority-order, then in the worst case, the object may travel in the reverse order before arriving at $v_{R_i}^1$. Each transaction with priority p is aborted just before it commits by the transaction with priority $p-1$. Thus, R_i travels along the path $v_{R_i}^0 \rightarrow v_{R_i}^{N_i} \rightarrow ... \rightarrow v_{R_i}^2 \rightarrow v_{R_i}^1$. In this path, transaction invoked by by $v_{R_i^j}$ is overtaken by all transactions with priorities lower than j, implying

$$d(v_{R_i}^0, v_{R_i}^{N_i}) < d(v_{R_i}^0, v_{R_i}^k), 1 \leq k \leq N_i - 1$$

and

$$d(v_{R_i}^j, v_{R_i}^{j-1}) < d(v_{R_i}^j, v_{R_i}^k), 1 \leq k \leq j - 2.$$

Clearly, the worst-case traveling path of R_i for a successful commit of the transaction invoked by $v_{R_i}^j$ is the nearest neighbor path in G_i^j starting from $v_{R_i^{j-1}}$, where G_i^j is a subgraph of G_i obtained by removing $\{v_{R_i}^0, ..., v_{R_i}^{j-2}\}$ in G_i and $G_i^1 = G_i$.

We use $NN(G, v_i)$ to denote the traveling cost of the nearest neighbor path in graph G starting from v_i. We can easily prove the following equation by directly applying Theorem 1 from [15].

$$\frac{NN(G_i^j, v_{R_i}^{j-1})}{\min_{v_{R_i}^k \in G_i^j} H(G_i, v_{R_i}^{j-1}, v_{R_i}^k)} \leq \lceil \log(N_i - j + 1) \rceil + 1 \quad (3)$$

Theorem 1 from [15] studies the competitive ratio for the nearest *tour* in a given graph, which is a circuit on the graph that contains each node exactly once. Hence, we can prove Equation 3 by the triangle inequality for metric-space networks. We can apply Equation 3 to upper-bound $makespan_i^d(Greedy, LAC)$:

$$makespan_i^d(Greedy, LAC) \leq \sum_{1 \leq j \leq N_i} NN(G_i^j, v_{R_i}^{j-1})$$

$$\leq \sum_{1 \leq j \leq N_i} \min_{v_{R_i}^k \in G_i^j} H(G_i^j, v_{R_i}^{j-1}, v_{R_i}^k) \cdot (\lceil \log(N_i - j + 1) \rceil + 1).$$

Note that

$$\min_{v_{T_j} \in V_T^{R_i}} H(G_i, v_{R_i}^0, v_{T_j}) \geq \min_{v_{R_i}^k \in G_i^j} H(G_i^j, v_{R_i}^{j-1}, v_{R_i}^k).$$

Combined with Lemma 1, we derive the competitive ratio for traveling makespan of the Greedy manager with a LAC protocol as:

$$CR_i^d(Greedy, LAC) = \frac{makespan_i^d(Greedy, LAC)}{makespan_i^d(\text{OPT})}$$

$$\leq \sum_{1 \leq j \leq N_i} (\lceil \log(N_i - j + 1) \rceil + 1) \leq \log(N_i!) + N_i.$$

The theorem follows. □

Example 2: We now revisit the scenario in Example 1 by applying LAC protocols. Note that $T_B \prec T_C$ and $d(A, B) < d(A, C)$. Due to the location-aware property, T_B will arrive at A earlier than T_C. Hence, the traveling makespan to execute T_B and T_C is $d(A, B) + d(B, C)$, which is optimal in this case.

Now we change the condition of $T_B \prec T_C$ to $T_C \prec T_B$. In this scenario, the upper bound of Theorem 5 is asymptotically tight. T_C may be overtook by T_B and the worst case traveling makespan to execute T_C is $d(A,B)+d(B,C)$, which is the nearest neighbor path starting from A.

Remarks: the upper bounds presented in Corollary 3 also applies to LAC protocols. However, for LAC protocols, the traveling makespan becomes the worst case only when the priority path is the nearest neighbor path.

5.3 $makespan(Greedy, LAC)$ for Multiple Objects

Theorem 5 and Corollary 3 give the makespan upper bound of the Greedy manager for each individual object R_i. In other words, they give the bounds of the traveling and execution makespans when the number of objects $s = 1$. Based on this, we can further derive the competitive ratio of the Greedy manager with a LAC protocol for the general case. Let $N = \max_{1 \le i \le s} N_i$, i.e., N is the maximum number of nodes that requesting for the same object. Now,

Theorem 6. *The competitive ratio CR(Greedy, LAC) is*

$$O(\max[N \cdot Stretch(LAC), N \log N \cdot s)].$$

Proof. We first prove that the total idle time in the optimal schedule is at least $N \cdot Stretch(LAC) \cdot Diam$ times the total idle time of the Greedy manager with LAC protocols, shown as Equation 4.

$$I(Greedy, LAC) \le N \cdot Stretch(LAC) \cdot Diam \cdot I(\textsc{Opt}) \tag{4}$$

If at time t, the system becomes idle for the Greedy manager, there are two possible reasons:

1. A set of transactions S is invoked before t have been committed and the system is waiting for new transactions. There exist an optimal schedule that completes S at time at most t, is idle till the next transaction released, and possibly has additional idle intervals during $[0, t]$. In this case, the idle time of the Greedy manager is less than that of $\textsc{Opt}$.

2. A set of transactions S are invoked, but the system is idle since objects haven't been located. In the worst case, it takes $N_i \cdot \delta^{LAC}(i,j)$ time for R_i to wait for invoked requests. On the other hand, it only takes $d(i,j)$ time to execute all transactions in the optimal schedule with the ideal cache-coherence protocol. The system won't stop after the first object has been located.

The total idle time is the sum of these two parts. Hence, we can prove Equation 4 by introducing the stretch of LAC.

Now we derive the bounds of $makespan^d$ and $makespan^\tau$ in the optimal schedule. Consider the set of write actions of all transactions. If $s+1$ transactions or more are running concurrently, the pigeonhole principle implies that at least two of them are accessing the same object. Thus, at most s writing transactions are running concurrently during time intervals that are not idle under $\textsc{Opt}$. Thus, $makespan^\tau(\textsc{Opt}))$ satisfies:

$$makespan^\tau(\textsc{Opt}) \ge \frac{\sum_{i=1}^m \tau_i}{s}.$$

In the optimal schedule, s writing transactions run concurrently, implying each object R_i travels independently. From Lemma 1, $makespan^d(\textsc{Opt})$ satisfies:

$$makespan^d(\textsc{Opt}) \ge \max_{1 \le i \le s} \min_{v_{T_j} \in V_T^{R_i}} H(G_i, v^0_{R_i}, v_{T_j}).$$

Hence, we bound the makespan of the optimal schedule by

$$makespan(\textsc{Opt}) \ge I(\textsc{Opt}) + \frac{\sum_{i=1}^m \tau_i}{s} + \max_{1 \le i \le s} \min_{v_{T_j} \in V_T^{R_i}} H(G_i, v^0_{R_i}, v_{T_j}).$$

Note that, whenever the Greedy manager is not idle, at least one of the transactions that are processed will be completed. However, from Theorem 2 we know that it may be overtook by all transactions with lower priorities, and therefore the penalty time cannot be ignored. Use the same argument of Theorem 2, we have

$$makespan^\tau(Greedy, LAC) \le \sum_{i=1}^s \sum_{k=1}^{N_i} k \cdot \tau_k.$$

The traveling makespan of transaction T_j is the sum of the traveling makespan of each object that T_j involves. With the result of Theorem 5, we have

$$makespan^d(Greedy, LAC) \le \sum_{i=1}^s \sum_{j=1}^{N_i} NN(G_i^j, v_{R_i}^{j-1})$$

$$\le s \cdot \sum_{j=1}^{N} \min_{v^k_{R_i} \in G_i^j} H(G_i^j, v_{R_i}^{j-1}, v_{R_i}^k) \cdot (\lceil \log(N-j+1) \rceil + 1)$$

$$\le s \cdot (\lceil \log(N!) \rceil + N_i) \cdot \max_{1 \le i \le s} \min_{v_{T_j} \in V_T^{R_i}} H(G_i, v^0_{R_i}, v_{T_j}).$$

Hence, the makespan of the Greedy manger with a LAC protocol satisfies:

$$makespan(Greedy, LAC) \le I(Greedy, LAC) + \sum_{i=1}^s \sum_{k=1}^{N_i} k \cdot \tau_k$$

$$+ s \cdot (\lceil \log(N!) \rceil + N) \cdot \max_{1 \le i \le s} \min_{v_{T_j} \in V_T^{R_i}} H(G_i, v^0_{R_i}, v_{T_j}).$$

The theorem follows. □

Remarks: Theorem 6 generalizes the performance of makespan of (Greedy,LAC). In order to lower this upper bound as much as possible, we need to design a LAC protocol with $Stretch(LAC) \leq s \log N$. In this case, the competitive ratio for (Greedy, LAC) is $O(N \log N \cdot s)$. Compared with the $O(s)$ bound for multiprocessors, we conclude that the competitive ratio of makespan of the Greedy manager degraded for distributed systems. As we stated in Section 4, the penalty of overtaking failures is the main reason for the performance degradation.

6. Conclusion and Open Questions

Compared with transactional memory systems for multiprocessors, the design of such systems for distributed systems is more challenging because of cache-coherence protocols and the underlying (higher) network latencies involved. We investigate how the combination of contention managers and cache-coherence protocols affect the performance of distributed transactional memory systems. We show that the performance of a distributed transactional memory system with a metric-space network is far from optimal, under the Greedy contention manager and an arbitrary cache-coherence protocol due to the characteristics of the network topology. Hence, we propose a location-aware property for cache-coherence protocols to take into account the relative positions of nodes in the network. We show that the combination of the Greedy contention manager and an efficient LAC protocol yields a better worst-case competitive ratio for a set of transactions. This results of the paper thus facilitate the following strategy for designing distributed transactional memory systems: select a contention manager and determine its performance without considering cache-coherence protocols; then find an appropriate cache-coherence protocol to improve performance.

Many open problems are worth further exploration. We do not know if there exists other combinations of contention managers and cache-coherence protocols that provide similar or better performance. For example, is there any other property of cache-coherence protocols that can improve the performance of the Greedy manager? Are there other contention managers that can provide a better performance with LAC protocols? What principles can be used by designers (of distributed transactional memory systems) to make such choices? These open problems are very interesting and important for the design of distributed transactional memory systems.

References

[1] H. Attiya, L. Epstein, H. Shachnai, and T. Tamir. Transactional contention management as a non-clairvoyant scheduling problem. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 308–315, 2006.

[2] R. L. Boccino, V. S. Adve, and B. L. Chamberlain. Software transactional memory for large scale clusters. In *PPoPP'08*, pages 247–258, 2008.

[3] M. R. Garey and R. L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187–200, 1975.

[4] R. Guerraoui, M. Herlihy, and B. Pochon. Toward a theory of transactional contention managers. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 258–264, 2005.

[5] L. Hammond, V. Wong, M. Chen, B. Hertzberg, B. D. Carlstrom, J. D. Davis, M. K. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun. Transactional memory coherence and consistency. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, June 2004.

[6] T. Harris and K. Fraser. Language support for lightweighted transactions. In *Proceedings of the 18th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, pages 388–402, 2003.

[7] M. Herlihy, V. Luchangco, and M. Moir. Obstruction-free synchronization: Double-ended queues as an example. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS)*, pages 522–529, 2003.

[8] M. Herlihy, V. Luchangco, M. Moir, and I. William N. Scherer. Software transactional memory for dynamic-sized data structures. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 92–101, 2003.

[9] M. Herlihy and J. E. B. Moss. Transactional memory: Architectural support for lock-free data structures. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, 1993.

[10] M. Herlihy and Y. Sun. Distributed transactional memory for metric-space networks. *Distributed Computing*, 20(3):195–208, 2007.

[11] J.A.Hoogeveen. Analysis of christofides' heuristic: some paths are more difficult than cycles. *Operations Research Letters*, 10:291–291, 1991.

[12] T. F. Knight. An architecture for most functional languages. In *Proceedings of ACM Lisp and Functional Programming Conference*, pages 500–519, August 1986.

[13] F. Lam and A. Newman. Traveling salesman path problems. *Math. Program.*, 113(1), 2008.

[14] K. Manassiev, M. Mihailescu, and C. Amza. Exploiting distributed version concurrency in a transactional memory cluster. In *PPoPP'06*, pages 198–208, March 2006.

[15] D. J. Rosenkrantz, R. E. Stearns, P. M. Lewis, and II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

[16] N. Shavit and D. Touitou. Software transactional memory. In *Proceedings of the 22nd Annual Symposium on Principles of Ditributed Computing (PODC)*, pages 204–213, 1995.

[17] I. William N. Scherer and M. L. Scott. Advanced contention management for dynamic software transactional memory. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 240–248, 2005.