Contents

5.1 Distributed Concurrency Control 2
5.1.1 Centralised Concurrency Control 2
5.1.2 Distributed Concurrency control 3
5.1.3 Transaction Management 4
5.1.4 Previous work 5
5.1.5 Time Stamp Ordering 5
5.1.6 Programmer productivity 6

Bibliography **8**

5.1 Distributed Concurrency Control

Transactional Memory systems require the application program to interact with a centralised transaction manager but this interaction makes programming difficult and restricts scalability. This section proposes using distributed transaction management to ensure the correct concurrent execution of Memory Transactions. Distributed transaction management makes concurrent programming easier and concurrent systems more scalable.

The main run-time component of a Transactional Memory system is the transaction manager which ensures the correct concurrent execution of Memory Transactions. Correctness is usually taken to mean that the result of the concurrent execution is equivalent to the result obtained by executing the transactions in some serial order. A transaction manager ensures serialisability by enforcing a concurrency control protocol and the choice of protocol dictates the design of the transaction manager.

Section 5.1.3 introduces Transaction management.

The main contribution of this section is the observation that the correct concurrent execution of Memory Transactions can be ensured without centralised transaction management. This section focuses on ensuring the serialisable execution of functions acting on an Immutable Data Structure.

5.1.1 Centralised Concurrency Control

Centralised transaction management restricts the scalability of a concurrent system as some part of the management processing is necessarily serialised. As the number of concurrent processors increases the time spent within the serialised part grows and eventually dominates the execution time of the concurrent system. Amdahl's law imposes restrictions on the scalability of a system with centralised transaction management.

A concurrent application communicates with the transaction manager to signal that it is ready to commit a transaction and the transaction manager then responds. This two way communication cannot be easily hidden by abstraction. The orchestration of communication with the transaction manager makes concurrent programming difficult.

Centralised transaction management makes it difficult for programmers to use

Memory Transactions in existing programs. To make use of Memory Transactions a programmer must adapt a program to fit into a transaction processing framework. This is an obstacle to the integration of Memory Transactions into existing software and it is a barrier to the adoption of Transactional Memory.

The solution to these difficulties should ensure the serialisability of concurrent Memory Transactions without requiring a centralised transaction manager.

5.1.2 Distributed Concurrency control

Distributed transaction management is scalable because it does not require a centralised mechanism to enforce concurrency control. It makes concurrent programming easier because programmers do not need to coordinate the application's interaction with a centralised system and it makes the use of Memory Transactions in existing applications easier by alleviating the need to integrate a concurrent application into a centralised transaction management framework.

A distributed transaction manager can make the decision whether to commit or abort a transaction independent of operations taking place on other processors because a distributed concurrency control protocol requires only information local to a processor. It does not depend on any information about concurrently active transactions so in a distributed system it is not necessary to orchestrate the interaction of transactions on multiple processors. Each processor can implement transaction management independently.

A distributed transaction manager can make the decision whether to commit or abort a transaction using only local information about the transactions that affect an object. It does not depend on information about accesses to any other objects so in a distributed system each transaction manager can maintain information about the objects that it manages and go about making its decisions independent of the action of other transaction managers. Each object can implement transaction management independently.

A distributed transaction manager does not attempt to serialise access to multiple objects. Groups of objects that require mutually consistent access are logically connected and should be combined into a single object for the purposes of concurrency control.

A fully distributed concurrency control protocol requires no communication between transaction managers whatsoever as it can be implemented on a per processor per object basis.

5.1.3 Transaction Management

Database systems divide transaction management into three distinct tasks: concurrency control, contention management and scheduling. Concurrency control is the task of ensuring correct concurrent execution by enforcing serialisability. Contention management is the task of guaranteeing progress. Scheduling is the task of load-balancing the execution between processors. We make a distinction between these tasks and consider each independently. However, Transactional Memory systems tend not to treat these aspects of transaction management as distinct. Consequently, transaction management in Transactional Memory systems tends to be difficult to characterise.

A transaction manager ensures that concurrent execution is correct by ensuring that it is equivalent to a serial execution. Determining whether a concurrent execution is serialisable is a NP-Complete problem [Pap79]. A transaction manager enforces a concurrency control protocol which ensures that all conforming transaction schedules are serialisable.

A transaction manager applies the rules of the concurrency control protocol to determine whether a transaction can commit or not. A concurrency control protocol can be viewed as a set of invariants and a binary function which ensures them. In the Transactional Memory literature the action of this function is referred to as validation.

A concurrency control protocol can be enforced either pessimistically, by a scheduler which checks that each operation conforms to the invariants of the concurrency control protocol before it is executed, or optimistically, by a certifier that enforces the concurrency control protocol when a transaction commits. The Transactional Memory literature refers to pessimistic concurrency control as eager validation and optimistic concurrency control as lazy validation. Many Transactional Memory systems employ mixed protocols detecting some types of conflict eagerly and others lazily.

A concurrency control protocol considers conflicting read and write operations acting on variables. These conflicts can be either between a read and a write or between two writes. Different concurrency control protocols can be applied independently to each type of conflict. A concurrency control protocol considers conflicts between these operations without regard to the values of the variables. Transactional Memory systems can be roughly divided into those which regard the variables as objects and those which regard them as memory words.

A transaction certifier requires a record of the read and write operations on variables and the transactions that issued them. The association between variables and transactions can be maintained by placing a transaction identifier within each affected object. It can also be maintained by associating a transaction with a list of addresses or object identifiers representing its read and write set. A certifier also requires meta-data, such as time stamps, relating to the operations on each variable.

The interaction between weakly isolated transactions is complex so concurrency control is simplified by strong isolation. The validation process is made simpler if it is known that all the values read by a transaction were written by transactions that have already committed.

5.1.4 Previous work

Bernstein comprehensively describes concurrency control and transaction management in a book entitled 'Concurrency Control and Recovery in Database Systems' [BHG87]. Özsu describes distributed transaction management and distributed concurrency control in database systems [ÖV99].

Kotselidis develops the idea of distributing Memory Transactions across a computing cluster [KAJ+07]. Hammond describes the TCC protocol which is a centralised broadcast based concurrency control protocol enforced by a centralised transaction manager [HCW+04]. Kotselidis describes a centralised broadcast concurrency control protocol based on the TCC protocol which ensures the serialisability of transactions both within a Chip Multi-Processor and across the cluster. However, in a computing cluster the latency and bandwidth restrictions of Inter-Processor Communication are more severe and the problems created by centralised transaction management are more apparent than in a Chip Multi-Processor. Kotselidis found that the centralised nature of transaction management made concurrent programming difficult and restricted the scalability of the system [KAJ+08]. These problems were not easily overcome despite a significant engineering effort.

5.1.5 Time Stamp Ordering

There are several distributed concurrency control protocols described in the literature and each can be applied independently to different types of conflict. Both the Time Stamp Ordering protocol and Reed's Multi-version Time Stamp Ordering protocol can be implemented without blocking so a distributed transaction manager can enforce either concurrency control protocol [BHG87] [Ree79].

Pessimistic concurrency control requires fine-grained memory serialisation and a strongly coherent memory model. As the number of processors on a Chip Multi-Processor increases the overhead of implementing fine-grained memory serialisation in hardware increases [HP06]. The Transactional Memory literature therefore makes a strong case for optimistic concurrency control [HLR10].

The Time Stamp Ordering concurrency control protocol can be enforced optimistically by a Time Stamp Ordering certifier which associates each transaction with a unique monotonically increasing time stamp. The certifier maintains a set containing the variables read and written by a transaction and also associates each variable with the time stamp of the transaction that wrote the variable and the highest time stamp of any transaction to have read the variable. When a transaction commits the certifier examines the read and write time stamps of all of the variables affected by the transaction and if the operations conform to the protocol then the transaction can commit, otherwise it must be aborted.

5.1.6 Programmer productivity

Ease of problem diagnosis is an important contributor to overall programmer productivity. It is often very difficult to diagnose problems in a concurrent system where concurrency control is enforced by a locking protocol because it can be difficult to determine which transaction wrote a particular value to a variable. When Time Stamp Ordering is used as a concurrency control protocol transactions appear to occur in the order of their starting time stamps. The order in which transactions are executed can be recorded and this aids the diagnosis of any problems that occur when a transactional system is executing concurrently. The order of the memory operations at the time the problem occurred can be determined using from the read and write time stamps associated with variables so it is possible to diagnose a problem from a core dump taken at the moment in time that a problem occurred.

Ease of problem reproduction is an important contributor to overall programmer productivity. It is often very difficult to reproduce a problem in a concurrent system where concurrency control is enforced by a locking protocol because the serial order, to which the execution should be equivalent, may be unknown. When

Time Stamp Ordering is used as the concurrency control protocol the serial order is given by the order of the transaction time stamps so it is possible to reproduce problems by executing the transactions serially in the order given by their time stamps.

Bibliography

[BHG87] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[HCW$^+$04] Lance Hammond, Brian D. Carlstrom, Vicky Wong, Ben Hertzberg, Mike Chen, Christos Kozyrakis, and Kunle Olukotun. Programming with transactional coherence and consistency (tcc). In *ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, pages 1–13. ACM Press, October 2004.

[HLR10] Tim Harris, James R. Larus, and Ravi Rajwar. *Transactional Memory, 2nd edition.* Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2010.

[HP06] John L. Hennessy and David A. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[KAJ$^+$07] Christos Kotselidis, Mohammad Ansari, Kim Jarvis, Mikel Luján, Chris Kirkham, and Ian Watson. Designing a distributed software transactional memory system. In *ACACES '07: 3rd International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems*, July 2007.

[KAJ$^+$08] Christos Kotselidis, Mohammad Ansari, Kim Jarvis, Mikel Luján, Chris Kirkham, and Ian Watson. Distm: A software transactional memory framework for clusters. In *ICPP '08: Proceedings of the 37th IEEE International Conference on Parallel Processing.* IEEE Computer Society Press, September 2008.

[ÖV99] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, Second Edition*. Prentice-Hall, 1999.

[Pap79] Christos H. Papadimitriou. The serializability of concurrent database updates. *J. ACM*, 26:631–653, October 1979.

[Ree79] David P. Reed. Implementing atomic actions on decentralized data. In *Proceedings of the seventh ACM symposium on Operating systems principles*, SOSP '79, pages 163–, New York, NY, USA, 1979. ACM.