POSITION PAPER: A COMPARATIVE STUDY OF CONSISTENCY IN DIFFERENT REPLICATED DOMAINS

K. Ramamritham, E. Brown, J. Dey, M. Kamath,
J. Kundu, L. Molesky, E. Nahum, C. Pedregal,
B. Purimetla, R. Sivasankaran and D. Yates

COINS Technical Report 92-36
May 1992

Position Paper: A Comparative Study of Consistency in Different Replicated Domains [1]

K. Ramamritham, E. Brown, J. Dey, M. Kamath, J. Kundu, L. Molesky, E. Nahum, C. Pedregal, B. Purimetla, R. Sivasankaran, D. Yates
Dept. of Computer Science, University of Massachusetts, Amherst, MA 01003

Replication is a ubiquitous technique employed in different parts of today's computer systems. These include:

- Memory System: The use of memory hierarchies, multiprocessor caches, and distributed shared memories imply that a data item may be found in different parts of a system's memory.
- File System: The use of file caches on the one hand, and distributed file systems on the other imply that a certain file or portion thereof may be replicated.
- Communication System: When a message is sent from one node to another and in particular, in a multicast or broadcast environment, the transmitted information is being replicated.
- Database System: Replicated data clearly occur in replicated distributed database systems.

Typically, replication is resorted to for two reasons: improved reliability/availability and improved performance. Also, what is common across these areas is the recognition that given replicated data, it is important to ensure that different copies of the data are *consistent* with each other, especially in the presence of concurrent accesses to the data. However, work in each of these areas has approached replication in ways that are seemingly unique to that area. Specifically, the fact that different terms are used to refer to consistency notions in the different areas gives the impression that they are unrelated or that there is little in common. For example, with regards to consistency, one comes across the term coherence in the context of memories, UNIX semantics in the context of files, causal ordering in the context of communication, and serializability in the context of databases.

Our goal, during the last few months, has been to conduct an in-depth investigation of the semantics of (the different terms used to refer to) consistency notions adopted in the different areas in order to accomplish the following [11]:

- Understand the similarities and differences in consistency requirements in the different areas.
- Investigate whether concepts/techniques used in one are applicable to (or have been applied to) other areas.

[1]This material is based upon work supported by the National Science Foundation under grant IRI-9109210.

Area	Unit of Consistency	Operations	Some Consistency Notions
File System	Entire file/block	Read, Write Open, Close	UNIX Semantics, Session Semantics
Memory	Cache block/page	Read, Write (global performing of)	Sequential Consistency, Processor Consistency, Release Consistency
Replicated Databases	Database object	Transaction	Serializability and its variations
Multicast Communication	Multicast message	Send, Receive, Delivery	Causal ordering, Single source ordering

Table 1: Aspects of Consistency in Different Areas

Memory	File System	Multicast Comm.	Replicated DBs
Sequential Consistency	UNIX Semantics/ Session Semantics[2]	ABCAST	Serializability
Causal Memory		CBCAST	
Processor Consistency (PRAM)		Single Source	QSR [3]
Slow Memory	NFS		PWSR

Table 2: Relationships between Consistency Notions in Different Areas

- Unify consistency issues across different areas so as to provide a common set of conceptual and implementation-oriented tools that have broader applicability.

In order to achieve the above goals, we started by asking the following questions with respect to each area.

- What is the unit of consistency (i.e., what is the granularity of data)?
- What is the unit of work at the end of which consistency is expected to hold (i.e., what is the granularity of the consistency preserving operation(s))?
- What are the different notions of consistency employed?

The answers with respect to each area are tabulated in Table 1. Table 2 shows how the different consistency notions in the various areas are related. Below we summarize, very briefly – given the page limitations – *some* of the entries found in Table 2. Specifically, we have attempted to show how consistency notions in the context of memory can be related to those in multicast communication, file systems, and databases.

[2]If we consider *Open* = *Read* and *Close* = *Write* and the entire file as a single memory location.

[3]In the PRAM model the writes issued by a single processor are seen in the same order everywhere. Similarly in quasi-serializability (QSR) [2], all local transactions and global subtransactions executed at a single site are serializable.

Our motivation is to point out that in fact there are many ideas used in one area that are applicable to others and that such cross-fertilization possibilities should be recognized and exploited for overall improvement in system design and performance. In what follows, we are going to confine ourselves to consistency issues and not discuss related issues, such as atomicity, recovery, and durability of the data.

Relationships between Consistency Notions. We begin by relating memory consistency notions to consistency in multicasting environments. In a distributed/multiprocessor system, multiple processes may need to cooperate, e.g., manage replicated data, monitor one another's state and so forth; they do so by forming process groups and communicating among themselves by means of group multicast or broadcast. In a multicast environment, a process sends a message to all the processes in the group – this can be viewed as a write update performed on a single variable (cache block or word) shared among different processors. Thus, the members of a particular multicast group can be thought of as processors that share a particular variable. We can also relate the primitive operations in both the areas. The primitives in a communication system are *send*, *receive* and *deliver* [1], while those for a memory system are *read* and *write*. The send and receive events together correspond to the write event, while delivery can be thought of as a read. The point in time at which consistency is maintained for a communication (memory) system is immediately prior to a delivery (read), after a sequence of zero or more sends and receives (writes).

With these analogies, it becomes easier to compare the consistency models in the context of communication systems with their counterparts in memory systems. In a communication system, the ABCAST protocol or the notion of *total causal ordering*, as introduced in [1], has the following properties:

1. If a process sends a message m_1 before m_2, then all the processes receiving both m_1 and m_2 should have m_1 delivered before m_2.

2. Messages in the system are partially ordered with respect to *potential causality* (or just *causality* - see [1] for a complete definition). In property 1, m_1 (potentially) causes m_2. For a multicast transmission, the send of message m_i, causes all receives (and deliveries) of m_i.

3. If at any process m_1 is delivered before m_2, then all the processes which receive both m_1 and m_2 should have m_1 delivered before m_2.

These three conditions impose a total order (across overlapping multicast groups) in which messages can be delivered, where a send and receive cause a delivery. This is equivalent to the notion of sequential consistency [8] in a memory system where a total order is imposed on all reads and writes, and all memory accesses from each processor appear to execute in program order.

The CBCAST protocol [1] onlyt imposes the first two conditions on multicasts. This resembles *causal memory* [6] that requires all processors to agree on the order of causally related accesses, but allows concurrent events that are not causally related to be observed in different orders.

Single source ordering [3] is a weaker form of CBCAST, since it does not contain the notion of causality, only requiring that the first of the three properties stated above be satisfied. This

is closely related to the notion of PRAM (Pipelined RAM [10]) and *processor consistent* [5] memory systems, where the only requirement is that all processors agree on the order of all observed writes by a single processor.

It is important to point out several differences between consistency models for communication and memory systems. For multicast communication, the unit of work after which consistency is expected to hold is the multicast transmission. For memory, the analogous unit of work is the globally performed read or write. There is no analogous notion of weak ordering [4] in communication. Also the notions of *multiple source ordering* and *multiple group ordering* [3] do not have suitable analogies in the context of memory consistency.

We can make some analogies between memory consistency techniques and those in distributed file systems by examining the semantics of sharing in a distributed file system. Two common protocols are UNIX semantics and and Session semantics [9]. UNIX semantics guarantee sequential consistency by enforcing coherent memory (files) and assuming the file accesses of a program are executed in program order. Session semantics, on the other hand, has no direct memory consistency analogue. The problem is that all reads and writes are done to the local copy and no writes are made globally visible until the file is closed. However, if we equate the open of a file with a main memory read and the close of a modified file with a main memory write (essentially treating the entire file as a single memory location), it is possible to draw some analogies. Session semantics now provides sequential consistency across the file system as long as the file server services open and close requests in FIFO order and multiple requests from a given client to the file server arrive at the server in the same order in which they were issued from the client. The granularity of consistency is the entire file and consistency is maintained at file open and close operations.

Perhaps the most well known distributed file system is NFS. The granularity of consistency here is the file block, and consistency is maintained on a client by client basis when the client queries the server for a consistency check. The period of consistency checking is arbitrary, making the point at which consistency is guaranteed nondeterministic. This allows the effects of writes (at the block level) as seen by other clients to be arbitrarily delayed. Assuming that block writes by a client are performed at the server in the same order, then the order of writes by a given client to the same location (block) as seen by all other clients is maintained, but the order of writes to different locations is not. This results in a consistency notion similar to Slow Memory [6].

Turning to databases, the sequential consistency correctness criterion in memory systems is analogous to serializability in databases, if we consider each memory read and write (which typically would translate into multiple writes to different cache blocks, due to replication) as a transaction. Similarly Slow Memory is analogous to predicatewise serializability (PWSR) [7] where each data item is a consistency unit. Also the motivation behind weak ordering in memory is similar to that of ϵ-serializability in the database area.

Conclusions. Our goal in this work has been to understand the issue of consistency and its maintenance in different components of computer systems. Our motivation was to identify common threads across areas and to thereby see if the ideas and techniques have broader applicability than is obvious on the surface. We believe that we have been successful in this endeavor and have been able to discern many interesting and useful similarities and differences across the areas. We plan to continue our investigations, especially with a view to understanding the performance implications of our initial findings.

References

[1] Kenneth Birman, André Schiper and Pat Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Trans. on Computer Systems* 9(3):272-314, Aug. 1991.

[2] Weimin Du and Ahmed K. Elmagarmid. Quasi Serializability: a Correctness Criterion for Global Concurrency Control in InterBase. *Proceedings Fifteenth International Conference on Very Large Data Bases*, pp. 347-55, Amsterdam, The Netherlands, Aug. 1989.

[3] Hector Garcia-Molina and Annemarie Spauster. Ordered and Reliable Multicast Communication. *ACM Trans. on Computer Systems* 9(3): 242-71, Aug. 1991.

[4] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta and John Hennessy. Memory Consistency and Event Ordering in Scalable Shared–Memory Multiprocessors. *Proceedings 17th Annual International Symposium on Computer Architecture*, pp. 15-26, May 1990.

[5] James R. Goodman. Cache Consistency and Sequential Consistency. Technical Report 61, SCI Committee, Mar. 1989.

[6] Phillip W. Hutto and Mustaque Ahamad. Slow Memory: Weakening Consistency to Enhance Concurrency in Distributed Shared Memories. *Proceedings IEEE 10th International Conference on Distributed Computing Systems*, pp. 302-09, May-June 1990.

[7] Henry F. Korth and Gregory D. Speegle. Formal Model of Correctness Without Serializability. *Proceedings ACM SIGMOD International Conference on Management of Data* pp. 379-86, June 1988.

[8] Leslie Lamport. How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs. *IEEE Trans. on Computers*, C-28(9): 690-91, Sept. 1979.

[9] Eliezer Levy and Abraham Silberschatz. Distributed File Systems: Concepts and Examples. *ACM Computing Surveys* 22(4):321-74, Dec. 1990.

[10] R. J. Lipton and J. S. Sandberg. PRAM: A Scalable Shared Memory. Technical Report CS-TR-180-88, Princeton University, Department of Computer Science, Sept. 1988.

[11] K. Ramamritham, E. Brown, J. Dey, M. Kamath, J. Kundu, L. Molesky, E. Nahum, C. Pedregal, B. Purimetla, R. Sivasankaran, D. Yates, "A Comparative Study of Consistency in Different Replicated Domains", (in preparation).