Relaxed Multicast Protocols using Concurrency Control Theory

Paul Jensen Nandit Soparkar

Electrical Engineering & Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122 USA

Contact author: Paul Jensen, pjensen@eecs.umich.edu
Track: Long presentation

Abstract

Techniques for coordinating distributed executions include process groups and transactions, each with different properties in a trade-off in terms of the degrees of consistency and performance. Transaction concurrency control theory may be used to characterize process group multicast semantics; our framework [18] which uses conflict relations among events, and the acyclicity properties of a particular precedence graph, may be used to describe the correctness of multicast orderings. In this paper, we describe protocols to realize new relaxed multicast orderings as obtained by the use of concurrency control techniques. In particular, we demonstrate efficient, time-stamp based protocols for relaxed orderings that depend on data, operation, or any similar attributes that bring about such weaker orderings. In fact, we provide a protocol for any arbitrary reflexive and symmetric relation that may be defined among the message types, and this makes our approach amenable to incorporating application semantics into multicast protocols. Using the precedence graph acyclicity framework, we can show that our protocols produce correct multicast delivery orderings. Furthermore, we discuss the efficiency of our protocols in terms of some qualitative performance metrics.

1 Introduction

With the continuing expansion of networks, and the increasing accessibility of the Internet, the number and nature of distributed applications is growing. However, due to application requirements such as consistency of shared data, performance, scalability, and failure resilience, the design of such applications is very complex in large-scale distributed environments. The use of middleware to relieve the burden for application designer has become increasingly important; some aspects of research in this regard have been explored by us (e.g., see [18, 19, 17, 30]). In this paper, we focus on efficient protocols for effecting message orderings that can be shown to provide consistent states based on our previously described techniques (i.e., in [18]). Our protocols are sound and complete in general in that they are provably correct in terms of producing only the requisite orderings, and also, they allow all possible histories that are consistent under the orderings.

There are two widely used approaches for managing data consistency in distributed environments incorporated within the middleware layer: the *process group* model [10, 7] and the *transaction* model [3]. The process group model is characterized by modeling the distributed system as a collection of processes that communicate by multicasting messages. Techniques are provided for ordering concurrent messages as well as ensuring atomicity of message delivery in the presence of process and link failures, and changes in group membership. The transaction model is characterized by operations being grouped into units called transactions, and techniques are provided for ordering concurrent transactions and ensuring their atomicity. In both models there is a need to order concurrent events, and to ensure atomicity in the presence of failures.

Some applications are better coordinated by group multicast techniques whereas others are better suited to transactions (e.g., see [4, 9]). Generally, the multicast approach is more efficient and provides adequate consistency criteria for some applications, and transactional systems are better suited to cases where a high degree of consistency criteria is needed. However, there are numerous other distributed applications which could benefit from a good meld of the performance and consistency criteria (e.g., see [4, 9]). To this end, one goal of our work has been to improve the multicast approach by proposing more efficient multicast techniques based on applications semantics and transaction models.

We have developed a framework to apply transaction concurrency control to multicast orderings. Each message multicast is viewed as a transaction, and the message are delivered at each site to the application after enforcing the appropriate ordering in a manner similar to a database scheduler ordering operations in concurrent transactions. We consider variations of standard message orderings to exhibit our approach in incorporating (hitherto difficult [9, 25]) application semantics. Our approach helps in ratifying larger classes of message orderings as being correct, thereby increasing the potential for improved performance. We review our framework in Section 2.

In this paper, we describe protocols to effect standard and certain relaxed orderings suggested by our theory, and we explain how they meet the required correctness criteria. Our protocols are developed by extending standard protocols for FIFO, causal, and total ordering in a straightforward manner. Furthermore, we have developed a technique to provide correct protocols for general relaxed ordering classes. The protocols extensions incur little overhead as we discuss based on certain qualitative metrics that we provide. Note that we do not consider issues of fault tolerance in this paper.

2 Enabling Techniques

We survey related distributed systems work done by others, and thereafter, we provide a brief synopsis of our own work from [18] in order to describe our protocols. We do not discuss the numerous relaxed transactional techniques (e.g., see [12, 16]); at the current stage, our approach is independent of, and different from, such techniques since it applies to the multicasting as opposed to transaction executions *per se*.

2.1 Background

One of the early toolkits for process groups, ISIS [8, 5], provides protocols to ensure totally and causally ordered multicasts. A number of toolkits which provide similar group membership, failure, ordering and atomicity semantics have emerged for multicasting (e.g., see [29, 24]). In addition, a number of protocols for multicast (e.g., see [26, 13, 8, 14]) and group membership (e.g., see [1, 11, 15]) semantics have been proposed in the literature. In most of this work, there is little consideration given to application semantics similar to our effort in which message content may be used to influence the orderings. That is, others do not deal with the relaxed ordering semantics for multicasts that we consider, and we describe how available protocols may be extended to incorporate our approach.

Multicasts may be ordered in a distributed system in several ways. For instance, total, FIFO, and Causal orders have been described in the literature (with some variation on their precise meaning). In [31], the different definitions for total order multicast arise from the semantics of delivery agreement guarantees among the processes in the face of various faults; and these are not considered in this paper. Work related to fault tolerance would complement, and be amenable to integration with our work.

Managing the consistency of replicas in a distributed environment is also related to our research, and several schemes based on a state-machine approach [27] can be implemented using totally ordered multicasts. Replication schemes which exploit message orderings (e.g., see [6, 7, 20, 23]), and grouping of messages have been investigated as techniques for improving performance of distributed protocols. In [20], the replication scheme has events grouped into three categories, each with different ordering semantics, and each providing a different level of the trade-off between consistency and performance. Our framework can be tailored appropriately to represent the ordering semantics provided by most of these approaches. And yet, in contrast to our work as well as typical relaxed transactional approaches, the available protocols generally do not use application-level semantics to define weaker ordering semantics.

2.2 A Concurrency Control Approach to Multicast Orderings

Below, we summarize our framework (see [18] for details) to describe our protocols.

- We assume that a set of multicast messages, if effected sequentially (i.e., are serial), will represent a consistent sequence of state changes in the distributed application environment.
- We identify "conflict" relations between the multicast primitives which indicate the events which may not commute. Intuitively, two events conflict if the order of their occurrence is significant with respect to consistency considerations.
- We regard as being "equivalent" those histories of multicast primitives (events) that arise as a consequence of commuting non-conflicting events. Concurrent message multicast histories are regarded as being correct (i.e., preserving consistency) if they are equivalent to a serial history.

Our model assumes the processes execute asynchronously, and the physical messages may be delayed or re-ordered. The processes are denoted by $P = \{p_1, p_2, ...\}$, and the execution of a process is a sequence of *events*. An event is one of a *global send*, *receive*, or a *local* event. A *global send* event, *gsend*, results in a physical message being multicast to all processes, and a *receive* event, *rcv*, delivers a message to an application; the *rcv* event may occur after a delay from the physical arrival of the message. A *local event* is executed by a process *locally*, and the *global history H* is the "happened-before" relation (denoted with the symbol $\rightarrow$, see [21]) on the process events.

Arriving messages are delivered to the applications such that consistent histories are generated. A local *delivery module* at each process determines when delivery is to occur. The input to a delivery module is the sequence of messages as the arrive, and the output of the delivery module is in an appropriate order of the corresponding *rcv* events.

Consider the three well-known "incidental" orderings, which are shown to represent classes with the same names in [18]. Our notation for the send and receive events is: $gsend_i(m)$ is the broadcast of message m from site i to all other sites, and $rcv_j(m, i)$ is the delivery of message m at site j originating from site i.

- *Sender-based FIFO Order* (i.e., $gsend_i(m) \rightarrow gsend_i(m') \Rightarrow rcv_r(m, i) \rightarrow rcv_r(m', i)$):
  1. $rcv_j(m, i)$ conflicts with $rcv_j(m', i)$
  2. $gsend_i(m)$ conflicts with $gsend_i(m')$
- *Causal Order* (i.e., $gsend_i(m) \rightarrow gsend_j(m') \Rightarrow rcv_r(m, i) \rightarrow rcv_r(m', j)$):
  1. $rcv_r(m, i)$ conflicts with $rcv_r(m', j)$ iff ($gsend_i(m) \rightarrow gsend_j(m')$ or $gsend_j(m') \rightarrow gsend_i(m)$
  2. $gsend_i(m)$ conflicts with $gsend_j(m')$ iff $gsend_i(m) \rightarrow gsend_j(m')$ or $gsend_j(m') \rightarrow gsend_i(m)$)

- *Total Order* (i.e., $rcv_r(m,i) \rightarrow rcv_r(m',j) \Rightarrow rcv_k(m,i) \rightarrow rcv_k(m',j)$):

    1. $rcv_r(m,i)$ conflicts with $rcv_r(m',j)$

Each global history is *complete* in that for every message in the history, all the events of the message are included in the history. A global history *serial*, w.r.t. to the *gsend* and *rcv* events if each individual message is contained within an interval of time as measured by a physical global clock, and distinct messages are contained in disjoint time intervals. Given a set of conflict relations M, two histories H_1 and H_2 are said to be *conflict equivalent* w.r.t. M, if they contain the same set of events and for two conflicting events o and p in H_1 and H_2, $o \rightarrow p$ in H_1 iff $o \rightarrow p$ in H_2. An *allowable* global history H is one where, for a set of conflict relations M, H is conflict equivalent to a serial history. For a given history H, the *precedence graph*, $PG(H)$, has the messages as its nodes, and an edge $m \stackrel{pg}{\rightarrow} m'$, iff m has an event that happened before, and conflicts with, an event of m'.

Equivalence Theorem [18]: *For a given set of conflict relations M, a global history H is conflict equivalent wrt M to a serial history iff $PG(H)$ is acyclic.*

Corollary [18]: *For a global history H and a set of conflict relations M, H is in the class of orderings allowed by M iff $PG(H)$ is acyclic.*

Three additional classes of relaxed *incidental orderings*, which are *relaxed FIFO*, *relaxed causal*, and *relaxed total*, are obtained by augmenting the orderings FIFO, causal, and total respectively with semantic information. We use $(type(m), type(m')) \in Con$ to represent a conflict relation on message types, where *type* is simply a function to indicate the nature of the message contents, and Con is the conflict relation on the message types. The relation Con and function *type* are unspecified, and in general, they may be derived from application semantics.

- *Relaxed FIFO Order* (i.e., $((gsend_p(m) \rightarrow gsend_p(m')) \wedge ((type(m), type(m')) \in Con)) \Rightarrow rcv_q(m,p) \rightarrow rcv_q(m',p)$):

    1. $rcv_i(m,j)$ conflicts with $rcv_i(m',j)$ iff $(type(m), type(m')) \in Con$
    2. $gsend_i(m)$ conflicts with $gsend_i(m')$ iff $(type(m), type(m')) \in Con$

- *Relaxed Causal Order* (i.e., $(gsend_i(m) \rightarrow gsend_j(m')) \wedge ((type(m), type(m')) \in Con) \Rightarrow rcv_k(m,i) \rightarrow rcv_k(m',j)$):

    1. $rcv_i(m,j)$ conflicts with $rcv_i(m',k)$ iff $((type(m), type(m')) \in Con$ and $(gsend_j(m) \rightarrow gsend_k(m')$ or $gsend_k(m') \rightarrow gsend_j(m)))$
    2. $gsend_i(m)$ conflicts with $gsend_j(m')$ iff $((type(m), type(m')) \in Con$ and $(gsend_i(m) \rightarrow gsend_j(m')$ or $gsend_j(m') \rightarrow gsend_i(m)))$

- *Relaxed Total Order* (i.e., $((rcv_i(m,j) \rightarrow rcv_i(m',k)) \wedge ((type(m), type(m')) \in Con)) \Rightarrow rcv_r(m,j) \rightarrow rcv_r(m',k))$:

  1. $rcv_i(m,j)$ conflicts with $rcv_i(m',k)$ iff $(type(m), type(m')) \in Con$.

3 Multicast Ordering Protocols

We describe protocols for the message ordering classes. To show that a protocol ensures the correct ordering of message deliveries by the Corollary, prove that a global history that evolves under the protocol results in an acyclic PG w.r.t. the corresponding message order class.

3.1 Incidental Ordering Protocols

Protocols to ensure FIFO, Causal, and Total order message delivery are well known, and as an illustration, we use our framework to prove the correctness of a simple FIFO protocol. In the protocol, the sender timestamps each message using a sequence number, and then increments the sequence number. The recipients of messages deliver only in order of the sequence numbers. To show that this protocol results in an acyclic PG_{FIFO}, assume to the contrary. An edge from message node m to m' in PG_{FIFO} implies that $gsend(m) \rightarrow gsend(m')$ by the conflict definitions, and the timestamps ensure that for any two messages, $rcv_i(m) \rightarrow rcv_i(m') \Leftrightarrow gsend(m) \rightarrow gsend(m')$. The existence of a cycle in PG_{FIFO}, $m_1 \overset{pq^*}{\rightarrow} m_2 \overset{pq}{\rightarrow} m_1$ would imply that $gsend(m_1) \rightarrow gsend(m_2) \rightarrow gsend(m_1)$, which is impossible since the happened-before relation is a partial order.

We now discuss relaxed incidental orderings. For orderings permitted with consideration given to affected data items, for each of the timestamp-based protocols for FIFO, Causal, and Total orderings, the timestamps may be augmented with the identity for the data item in question. Also, for every data item, a different timestamp counter is maintained. Events for messages which pertain to the same data item are not allowed to commute, whereas events pertaining to different data items may commute. Again, our framework may be used to prove that this protocol would work correctly. Note that while the protocols for the simple case of a single data item per message is similar to maintaining separate message groups, it is not easy to characterize the more complex cases using other available techniques.

Now consider the cases for operation-based relaxed incidental orderings. We provide protocols for all three variants of relaxed incidental orderings for a subset of the conflict relations, specifically, the case where the conflict relation Con is an equivalence relation. *Rcv* events pertaining to messages within an equivalence class are not allowed to commute, whereas other *rcv* events may do so. Observe that these protocols are essentially the same as for the case of separate data items described above. In fact, the subject of this observation is more that coincidental as we explain below.

3.2 Data-Oriented Protocols

We describe the *type* function and *Con* relation for the data-oriented orderings in order to provide protocols for each of the data-FIFO, data-causal, and data-total orderings. For these data-oriented orderings, we consider N data items $x_1, x_2, ...x_N$. A message pertains to one or more data items. The type of a message is the subset of the data items to which the message pertains. Two messages conflict if the messages pertain to at least one common data item. The data-oriented order classes are defined as the relaxed incidental order classes of Section 2.2 with the function *type* and relation *Con* defined as described, where $type(m) \subseteq \{x_1, x_2, ...x_N\}$, and $type(m)\ Con\ type(m')$ iff $type(m) \cap type(m') \neq \emptyset$.

The protocols described for each of the three classes are intended to illustrate the ordering aspect. Although the message type is described using data items, it is more general. For example, the messages could pertain to particular operations, and as long as the conflict relation of the operations can be capture using the *Con* relation we described, our protocols are valid. Also, note that for simplicity, we use broadcasts even though our approach is applicable to multicasts as well.

3.2.1 Data FIFO

We assume there are K processes $p_1, p_2, ...p_K$ and there are N data items $x_1, x_2, ...x_N$. Message type b is an N-bit binary number $b = b_1...b_N$ where bit $b_i = 1$ iff the message references data item x_i. At a given site i we define sequence numbers for the local state as follows. There are N sender sequence numbers, one for each data item, denoted by s_j where j identifies the data item. Also, there are $N \times K$ receiver sequence numbers, one for each process and data item, denoted r_{jk} where j is the process and k is the data item.

Sending a message m of type $b = b_1...b_N$ at process p_i:
    Store the type b in the message header.
    Store each sequence number s_j where $b_j = 1$.
    Increment each s_j where $b_j = 1$.
    Send the message.

Delivering a message m of type $b = b_1...b_N$ at site p_i:
    Let v be a vector of N sequence numbers.
    For each j where $b_j = 1$,
        set v_j to the sender sequence number stored in message m.
    If for all j where $b_j = 1$, $t_j = r_{ij}$
        then
            deliver m
            increment r_{ij} where $b_j = 1$
        else
            delay m until condition true.

3.2.2 Data Causal

The protocol for data-causal is a simple modification of the vector clock approach (see [22, 8, 2]). Again, we assume K processes, N data items, and an N-bit binary number for the data type. The idea is to maintain at each site, N vectors (one for each data item), where each vector is of length K (one integer element for each process). We denote the vector by v_{ij} where i refers to the process and j refers to the data item. The procedures for sending and delivering of messages are as follows.

Sending a message m of type $b = b_1 ... b_N$ at process p_i:
    For each j where $b_j = 1$
        Increment v_{ij}.
    Store the vector v in the message header.
    Store the process number i in the message header.
    Store the message type b in the message header.
    Send the message.

Receiving a message m of type $b = b_1 ... b_N$ at process p_i:
    Set w to the vector stored in the message.
    Set j to the sender process number from the message.
    If for all k where $b_k = 1$,
      ($w_{ik} = v_{ik} + 1$) and (for all $j \neq i$, $w_{jk} \leq v_{jk}$)
        then
            deliver m
        else
            m must be delayed until condition is true.

3.2.3 Data Total

A protocol similar to data-FIFO may be used to implement data-total — the only difference being that the sender sequence number must be a global sequence value. A way to do this is to have a token site p_k be the designated sequencer for all messages. When a process needs to broadcast a message, it first sends the message to process p_k, and then p_k broadcasts the message in data-FIFO order to the other processes. In effect, many of the the available protocols for Total ordering may be modified in a manner similar to our modification for data-FIFO and data-causal (e.g., by using a separate timestamp for each data item) to obtain data-total ordering.

There is even greater opportunity for improvement for the total ordering protocols considering that the sequencing can be distributed, e.g., for protocols that use a token holder, we can have a different token for each message class, and the tokens can reside at different sites. Total ordering protocols that distribute the sequencing must be carefully constructed to avoid deadlock situations. In this regard, there are many possibilities for relaxed total order protocols, the exploration of which is beyond the scope of this paper.

3.3 Properties of Protocols

Our protocols as described here have several useful properties. Soundness and completeness are w.r.t. a given ordering class, and all our protocols are sound since the allow only a subset of histories of the given class. A protocol is complete if it allows every history in the given order class. Completeness is pertinent with regard to allowing a greater number of realizable orderings. A protocol is non-blocking if it avoids deadlocks (i.e., it never reaches a state in which a message cannot be delivered without violating correctness).

Our data-oriented protocols are sound since they do not allow a cycle in the precedence graphs. Our data-oriented protocols are also complete (i.e., they allow all histories within the given order class). This is seen by considering any history H in the class and noting that it would have an acyclic PG(H), and we can select timestamps which follow both the precedence in PG(H) and the rules of the protocol. Furthermore, the protocols we provide are trivially non-blocking since, by incorporating appropriate conditions for progress, we can preclude indefinite waits.

Qualitative metrics of efficiency include the completeness and non-blocking criteria mentioned above. In addition, the relaxed ordering classes also imply that the protocols allow a greater degree of concurrent execution (see [18]) — as argued for transaction systems.

4 Protocols for More General Conflicts

It is desirable to provide an efficient protocol for any arbitrary Con relation to accommodate *ad hoc* message conflict relations derived from application requirements. In our work, we only consider cases where the Con relation is symmetric, since we assume the existence of this property for the definition of commutativity of events and conflict equivalence. Though we argue that it is possible to devise a protocol which is sound and complete for arbitrary symmetric Con relations, there are some difficulties for these as discussed below. We present sound and complete protocols for Con relations which are symmetric and reflexive, and discuss how the same technique leads to sound (but not necessarily complete) protocols for arbitrary Con relations.

4.1 Symmetric and Reflexive Message Conflicts

We consider protocols which will ensure relaxed FIFO, relaxed causal, and relaxed total for any arbitrary Con relation which is symmetric and reflexive. We use a particular conflict relation $AndCon_N$ for which we have protocols, in fact, the data-oriented protocols of Section 3.2. Then, given any arbitrary symmetric and reflexive relation Con, we describe how to map the message types to the elements of $AndCon_N$ such that the relations are equivalent. With the new mapping in place, we can use the protocols for the $AndCon_N$ relation (i.e., the data-oriented protocols). Finally, we illustrate how the mapping is done using an example.

Define $AndCon_N$ as a relation on N-bit binary numbers, where two numbers are related if there is at least one position where both numbers have a digit equal to 1 (i.e.,

$AndCon_N$={(b, b') : b AND $b' \neq 0$ }). For $AndCon_N$, we can directly apply the data-oriented protocols described in Section 3.2.

We map the given message types (i.e., the range values of *type*) to binary numbers such that the relation Con is equivalent to the relation $AndCon_N$. Let G be an undirected graph where the nodes are the range values of *type*, and the edges are defined by the relation Con such that there is an undirected edge between v and v' iff $(v, v') \in Con$ (i.e., implies $(v', v) \in Con$). Find a set of cliques in G, $clique_1, clique_2, ...clique_k$, which cover all the edges in G, (in the worst case, the cliques would correspond to the edges themselves). Now, for each node v (which corresponds to a message type) in G, define a function *btype* which maps nodes to k-bit binary numbers such that $btype(v) = b_1...b_k$ and $b_i = 1$ iff v is in $clique_i$. Now, with the relation $AndCon_N$ where $N = k$, and the types of messages defined as a composition $btype \circ type$, the relations are equivalent. That is, for any two messages m and m', $((type(m), type(m')) \in Con$ iff $(btype(type(m)), btype(type(m')) \in AndCon_N)$, which we prove as follows.

Proof:

($\Rightarrow$)
Assume $(type(m), type(m')) \in Con$.
Let $v = type(m)$ and $v' = type(m')$.
Then G has an undirected edge between v and v'.
There is a clique, say $clique_i$, which includes v and v' to cover this edge.
The i^{th} bit of both $btype(v)$ and $btype(v')$ is 1, and therefore
$(btype(v)$ AND $btype(v') \neq 0)$, implying
$(btype(v), btype(v')) \in AndCon_N$.

($\Leftarrow$)
Assume $(btype(type(m)), btype(type(m'))) \in AndCon_N$.
Let $v = type(m)$ and $v' = type(m')$.
By $AndCon_N$ definition, $(btype(v)$ AND $btype(v') \neq 0)$.
$btype(v)$ and $btype(v')$ have at least one common bit, say the i^{th} bit, which is 1.
By the mapping definition, v and v' are both in $clique_i$, implying
G has an undirected edge between v and v'.
Therefore, $(v, v') \in Con$.

Therefore, by using the type function $btype \circ type$, we can directly apply the protocols from Section 3.2 to any arbitrary symmetric and reflexive Con relation.

In Figure 1 we show a simple example for mapping the message types of the given Con relation to the binary numbers of the $AndCon_N$ relation. The Con relation is represented by the undirected graph (for simplicity, we do not show the reflexive edges). The graph consists of five message types (numbered 1–5) and five edges. All five edges can be covered with three cliques (as shown encircled by dotted lines) which are labeled 100, 010, and 001. As shown in the figure, each of the five message types is mapped to a new type; a binary number constructed by setting a bit for each clique that the message type is in. For example,

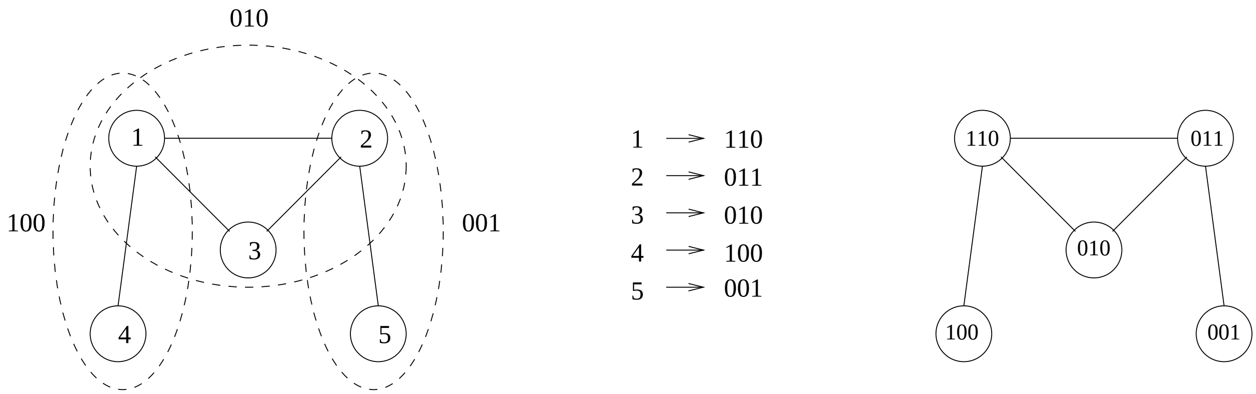

Figure 1: Mapping message types of symmetric, reflexive conflicts for data-oriented protocols.

type 1 is in cliques labeled 100 and 010, so the new type becomes 110. Now, the $AndCon_3$ relation, along with the new types, will lead to the expected message conflicts, as shown to the right in the figure.

4.2 Symmetric Message Conflicts

We can provide sound protocols for any arbitrary Con relation. This is done by using the same technique presented in Section 4.1. We use the given Con relation to form a new relation Con_{sr} by adding appropriate additional relationships to make Con_{sr} symmetric and reflexive. Then, we apply the mapping technique from Section 4.1, and finally, we use an appropriate data-oriented protocol. Such a protocol will be correct since it ensures a strict subset of histories defined by the conflict class of Con (since Con_{sr} covers all the edges of Con).

Although we believe that protocols which are sound and complete exist in principle for arbitrary symmetric message conflict Con relations, the existence of practical protocols is not evident. In the following we describe a protocol for relaxed causal ordering with symmetric message conflict relations, and argue for its correctness. The key idea of the relaxed causal ordering protocol is to make the causal ordering protocol cognizant of the conflict relations. Now, causal delivery can be effected, in principle, by piggy-backing on each message the causal history of that message [28]. To ensure relaxed causal ordering, the delivery of a message is delayed only if a message in its causal history has not been delivered (i.e., causal ordering) *and* the receive events at that site for the two messages conflict. The argument to show that this protocol results in an acyclic PG is similar to the case for FIFO: Assume to the contrary there is a cycle $m_1 \overset{pq}{\to}{}^* m_2 \overset{pq}{\to} m_1$ where $gsend_i(m_1) \to gsend_j(m_2)$ and $rcv_k(m_1, i) \to rcv_k(m_2, j)$. The protocol would not allow receives processed in this order, and $rcv_k(m_1, i)$ would be delayed until after $rcv_k(m_2, j)$. Therefore, the protocol would ensure an acyclic PG.

Providing sound and complete protocols for general symmetric message conflict relations can be regarded as being similar to *serializability graph testing* (see [3]) for database transactions. In our case, we test the precedence graph, where a message is delivered only if it does not create a cycle in the precedence graph. The shortcoming with such an approach is that it incurs significant overhead as it is necessary to exchange histories, and possibly conflict information, which may be quite large.

5 Conclusions

Our previously reported framework [18] is useful for unifying two widely used models for distributed computing: process groups and transactions. Our framework allows us to consider new classes of message event histories based on relaxed ordering semantics, and in this paper, we described the associated multicast delivery protocols. Our protocols are sound and complete for a variety of conflict relations. Furthermore, we provided protocols the relaxed orderings with fairly general conflict relations to accommodate semantics which could be derived from the applications. We demonstrated how the protocols achieve the required correctness criteria and discussed how they may lead to improved performance.

References

[1] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Membership Algorithms for Multicast Communication Groups. In *Proc. 6th Intl. Workshop on Distributed Algorithms (WDAG-6), (LNCS 647)*, pages 292–312, Haifa, Israel, November 1992.

[2] O. Babaoglu and K. Marzullo. *Distributed Systems*, chapter Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms. Addison-Wesley, 1993.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA, 1987.

[4] K. Birman. A Response to Cheriton and Skeen's Criticism of Causal and Totally Ordered Communication. *ACM Operating System Review*, 28(1):11–21, January 1994.

[5] K. P. Birman. The Process Group Approach to Reliable Distributed Computing. *Comm. of the ACM*, 36(12):37–53, December 1993.

[6] K. P. Birman and T. A. Joseph. Low-Cost Management of Replicated Data in Fault-Tolerant Distributed Systems. *ACM Trans. on Computer Systems*, 4(1):54–70, February 1986.

[7] K. P. Birman and T. A. Joseph. Reliable Communication in the Presence of Failures. *ACM Trans. on Computer Systems*, 5(1):47–76, February 1987.

[8] K. P. Birman, A. Schiper, and P. Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Trans. on Computer Systems*, 9(3):272–314, August 1991.

[9] D. R. Cheriton and D. Skeen. Understanding the Limitations of Causally and Totally Ordered Communication. In *Proc. of the 14th ACM Symp. on Operating Systems Principles*, pages 44–57, Asheville, NC, December 1993.

[10] D. R. Cheriton and W. Zwaenepoel. Distributed Process Groups in the V Kernel. *ACM Trans. on Computer Systems*, 3(2):77–107, May 1985.

[11] F. Cristian. Reaching Agreement on Processor Group Membership in Synchronous Distributed Systems. *Distributed Computing*, 4:175–187, 1991.

[12] A. K. Elmagarmid. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1992.

[13] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proc. of the ACM SIGCOMM Symp.*, pages 342–356, Cambridge, MA, August 1995.

[14] A. Gopal, R. Strong, S. Toueg, and F. Cristian. Early-Delivery Atomic Broadcast. In *Proc. of the 9th. Annual Symp. on Principles of Distributed Computing*, pages 297–310, Quebec City, Quebec, 1990.

[15] F. Jahanian, S. Fakhouri, and R. Rajkumar. Processor Group Membership Protocols: Specification, Design, and Implementation. In *Proc. of Symposium on Reliable Distributed Systems*, Princeton, NJ, October 1993.

[16] S. Jajodia and L. Kerschberg. *Advanced Transaction Models and Architectures.* Kluwer Academic Publishers, Norwell, MA, 1997.

[17] P. Jensen and N. Soparkar. Real-time concurrency control in groupware. In *Proceedings of the Engineering Systems Design and Analysis Conference*, Montpellier, France, July 1996. Also available as: Tech. Report: CSE-TR-265-95 from The University of Michigan, EECS Dept. Ann Arbor, MI, USA.

[18] P. Jensen, N. Soparkar, and A. Mathur. Characterizing multicast orderings using concurrency control theory. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, Baltimore, Maryland, May 1997.

[19] P. Jensen, N. Soparkar, and M. Tayara. Towards Distributed Real-Time Concurrency and Coordination Control. In *Advanced Transaction Models and Architectures*, chapter 12. Kluwer Academic, 1997.

[20] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Providing High Availability Using Lazy Replication. *ACM Trans. on Computer Systems*, 10(4):360–391, November 1992.

[21] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Comm. of the ACM*, 21(7):558–565, July 1978.

[22] F. Mattern. Virtual Time and Global States of Distributed Systems. In *Proc. of the Intl. Workshop on Parallel and Distributed Algorithms*, pages 215–226, Gers, France, October 1988.

[23] S. Mishra, L. L. Peterson, and R. D. Schlichting. Implementing Fault-Tolerant Replicated Objects Using Psync. In *Proc. of IEEE 8th. Symp. on Reliable Distributed Systems*, pages 42–52, Seattle, WA, October 1989.

[24] L. Moser, P. Melliar-Smith, D. Agarwal, R. Budhia, and C. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):54–63, April 1996.

[25] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-To-End Arguments in System Design. *ACM Trans. on Computer Systems*, 2(4):277–288, November 1984.

[26] A. Schiper and A. Sandoz. Uniform Reliable Multicast in a Virtually Synchronous Environment. In *Proc. of the 13th. Intl. Conf. on Distributed Computing Systems*, Pittsburgh, PA, 1993.

[27] F. B. Schneider. Implementing Fault-Tolerant Services using the State-Machine Approach. *ACM Computing Surveys*, 22, December 1990.

[28] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: in search of the holy grail. *Distributed Computing*, 7:149–174, 1994.

[29] R. van Renesse, K. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.

[30] C. Wallace, P. Jensen, and N. Soparkar. Supervisory control in workflow scheduling. In *Proceedings of the Int'l Workshop on Advanced Transaction Models and Architectures*, Goa, India, August 1996.

[31] U. G. Wilhelm and A. Schiper. A Hierarchy of Totally Ordered Multicasts. In *Proceedings of the IEEE Intl. Symposium on Reliable Distributed Systems (SRDS-14)*, Bad Neuenahr, Germany, September 1995.