

Lightweight Causal Cluster Consistency

Anders Gidenstam Boris Koldehofe Marina Papatriantafidou

Philippas Tsigas

CHALMERS | GÖTEBORG UNIVERSITY



Department of Computer Science and Engineering
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden

Göteborg, 2005



Technical Report in Computer Science and Engineering at
Chalmers University of Technology and Göteborg University

Technical Report no. 2005-09
ISSN: 1652-926X

Department of Computer Science and Engineering
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden

Göteborg, Sweden, April 2005

Lightweight Causal Cluster Consistency

Anders Gidenstam Boris Koldehofe Marina Papatriantafilou Philippas Tsigas

Abstract

Within an effort for providing a layered architecture of services supporting multi-peer collaborative applications, this paper proposes a new type of consistency management aimed for applications where a large number of processes share a large set of replicated objects. Many such applications, like peer-to-peer collaborative environments for training or entertaining purposes, platforms for distributed monitoring and tuning of networks, rely on a fast propagation of updates on objects, however they also require a notion of consistent state update. To cope with these requirements and also ensure scalability, we propose the *cluster consistency* model.

In the proposed model a dynamic set of processes, called coordinators, may concurrently propose updates to a subset of objects which form a cluster. Updates are applied in some order of interest by the coordinators of the cluster. Moreover, any interested process can receive update messages referring to replicated objects, with an option for the updates to be delivered unordered or in the same order as to the coordinators.

We also propose a two-layered architecture for providing cluster consistency. This is a general architecture that can be applied on top of the standard Internet communication layers and can offer a modular, layered set of services to the applications that need them. Further, we present a *fault-tolerant protocol* implementing causal cluster consistency with predictable reliability, running on top of decentralised probabilistic protocols supporting group communication. Our experimental study, conducted by programming and evaluating the two-layered architecture as protocols executing on top of standard Internet transport services, shows that the approach scales well, imposes an even load on the system, and provides high-probability reliability guarantees.

Keywords: *large scale group communication, consistency, collaborative environments, peer-to-peer systems, Internet application layer services*

1 Introduction

Many applications like collaborative environments allow a possibly large set of concurrently joining and leaving processes to share and interact on a set of common replicated objects. Processes deal with state changes of objects by sending update messages. Providing the infrastructure to support such applications and systems places demands for multi-peer communication, with reliability, time, consistency and scalability guarantees, especially in the presence of failures and variable connectivity of the peers in the system. Towards designing a set of service layers and lightweight distributed protocols to provide appropriate dissemination and coordination/consistency services for multi-peer collaborative applications to use, we introduce a consistency model which builds on scalable information dissemination schemes that provide probabilistic reliability guarantees. The required methods for communication and consistency must deal with the interests of users (processes) who may form groups which vary in size and behaviour.

Lightweight non-hierarchical protocols, which have good scalability potential at the cost of probabilistic guarantees on reliability have not been in the focus of earlier research in distributed computing, where the emphasis has been in proving feasible, robust solutions, rather than considering the aforementioned variations in needs and behaviour.

In this paper we look at a delivery service for updates which provides delivery in optimistic causal order, guaranteeing that an event will only be delivered if it does not causally precede an already delivered event. Events which become obsolete do not need to be delivered and may be dropped. Nevertheless, optimistic causal order algorithms aim at minimising the number of lost events. In order to detect which events that are missing many algorithms rely on the use of vector clocks. These, however, grow linearly with the group size, implying a limitation on the size of process groups with respect to scalability.

We propose a consistency management method denoted by *causal cluster consistency*, providing optimistic causal delivery of update messages to a large set of processes. Causal Cluster Consistency takes into account that for many applications the number of processes which are interested in performing updates can be low compared to the overall number of processes which are interested in receiving updates and maintaining replicas of the respective objects. Therefore, the number of processes which are entitled to perform updates at the same time is restricted to n , which corresponds to the maximum size of the vector clocks used. However, the set of processes entitled to perform updates may change dynamically.

Our proposed approach is in line with and inspired from recent approaches in multipeer information dissemination [6, 9, 10], where the aim is at what is called *predictable reliability*, guaranteeing that each event is delivered to all non-faulty destinations with a high-probability guarantee. We present a two-layer architecture implementing cluster consistency that can make use of lightweight communication algorithms which can in turn run using standard Internet transport services. Our method is also designed to tolerate a bounded number of process failures, by using a combined push-and-pull (recovery) method. We also present an implementation and experimental evaluation of the proposed method and its potential with respect to reliability and scalability, by building on recently evolved large-scale and lightweight probabilistic group communication protocols. Our implementation and evaluation have been carried out in a real network, and also in competition with concurrent network traffic by other users.

Structure of the paper. In Section 2 we present background information and related work on causal ordering and lightweight probabilistic group communication. In Section 3 notation and definitions are given. Section 4 introduces a layered architecture for achieving causal delivery and the two-layered protocol implementing it. Section 5 discusses the implementation and experimental evaluation of the proposed protocol running on top of standard Internet transport services. The paper concludes with a discussion of the contribution and future work.

2 Background

Many systems like collaborative environments (e.g. [18, 12, 8]) allow multiple processes to perform updates on shared replicated objects. In order to perform well for many processes, such systems rely on protocols which provide scalable group communication, support maintenance of membership information as well as fast dissemination of updates (*events*) in the system. Applications building on such systems would benefit from an event delivery service that satisfies the causal order relation, i.e. satisfies the “happened before” relation as described in [15]. A lot of work has been carried out achieving reliable causal delivery in the occurrence of faults [7, 5, 22, 14]. However, the schemes to recover lost messages can lead to long latencies for events, while often short delivery latencies are needed. In particular the latency in large groups can become large since, in the worst case, a causal reliable delivery service needs to add timestamp information to every event, whose size grows

quadratically with the size of the group.

Relaxed requirements, such as optimistic causal ordering as in [4, 23] can be suitable for systems where events are associated with deadlines. While the causal order semantics require that an event is only delivered after all causally preceding events have been delivered, optimistic causal order only ensures that no events are delivered which causally precede an already delivered event. However, optimistic causal delivery algorithms aim at minimising the number of lost messages.

Recent approaches for information dissemination use lightweight probabilistic group communication protocols [6, 9, 10, 13, 20, 3]. These protocols allow groups to scale to many processes by providing reliability expressed with high probability. In [20] it is shown that probabilistic group communication protocols can perform well also in the context of collaborative environments. However, per se these approaches do not provide any ordering guarantees.

Vector clocks [17] allow processes to determine the precise causal relation between pairs of events in the system. Further, one can detect missing events and their origin. Since the size of the timestamps grow linearly with the number of processes, to ensure scalability, one may need to introduce some bound on the growing parameter.

Also of relevance and inspiration to this work is the recent research on peer-to-peer systems and in particular the methods of such structures to share information in the system (cf. e.g. [25, 2, 21, 24, 26]), as well as a recent position paper for atomic data access on CAN-based data management [16].

3 Notation and problem statement

Let $G = \{p_1, p_2, \dots\}$ denote a group of processes with ongoing joining and leaving of processes and a set of replicated objects $B = \{b_1, b_2, \dots\}$. Processes maintain replicas of objects they are interested in. Let B be partitioned into disjoint clusters C_1, C_2, \dots with $\cup_i C_i \subseteq B$. Further, let C denote a cluster and p a process in G , then we write also $p \in C$ if p is interested in objects of C . *Causal Cluster Consistency* allows any processes in C to maintain the state of replicated objects in C by applying updates in optimistic causal order. However, at most n processes (n is assumed to be known to all processes in C) may propose updates to objects in C at the same time. Processes which may propose updates are called *coordinators* of C . Let $Core_C$ denote the set of coordinators of C . The set of coordinators can change dynamically over time. Throughout the paper we will use the term *events* when referring to update messages sent or received by processes in a cluster.

The propagation of events is done by multicast communication. It is not assumed that all processes of a cluster will receive an event which was multicast, nor does the multicast need to provide any ordering by itself. Any lightweight probabilistic group communication protocol as appears in the literature [9, 10, 13] would be suitable. We refer to such protocols as *PrCast*. PrCast is assumed to provide the following properties:

- An event is delivered to all destinations with high probability.
- Decentralised and lightweight group membership, i.e. a process can join and leave a multicast group in a decentralised way and processes do not need to know all members of the group.

Within each cluster we apply vector timestamps of the type used in [1]. Let the coordinator processes in $Core_C$ be assigned to unique identifiers in $\{1, \dots, n\}$ (a process which is assigned to an identifier is also said to *own* this identifier). Then, a time stamp t is a vector whose entry $t[j]$ corresponds to the $t[j]$ th event send by a process that *owns* index j or a process that *owned* index j before (this is because processes may leave and new processes may join $Core_C$). A vector time stamp

t_1 is said to be smaller than vector time stamp t_2 if $\forall i \in \{1, \dots, n\} t_1[i] \leq t_2[i]$ and $\exists i \in \{1, \dots, n\}$ such that $t_1[i] < t_2[i]$. In this case we write $t_1 < t_2$.

For any multicast event e , we write t_e for the corresponding timestamp of e . Let e_1 and e_2 denote two multicast events in C , then e_1 causally precedes e_2 if $t_{e_1} < t_{e_2}$, while e_1 and e_2 are said to be concurrent if neither $t_{e_1} < t_{e_2}$ nor $t_{e_2} < t_{e_1}$. Further we denote the index owned by the creator of event e as $index(e)$ and the event id of event e as $\langle index(e), t_e[index(e)] \rangle$.

Throughout the paper it is assumed that each process p maintains for each cluster C a *cluster-consistency-tailored logical vector clock* (for brevity also referred to as *cct-vector clock*) denoted by $clock_p^C$. A cct-vector clock is defined to consist of a vector time stamp and a sequence number. We write T_p^C when referring to the timestamp and seq_p^C when referring to sequence number of $clock_p^C$. T_p^C is the timestamp of the latest delivered event while seq_p^C is the sequence number of the last multicast event performed by p . In Section 4 when describing the implementation of causal cluster consistency, we explain how these values are used. Note, whenever we look at a single cluster C at a time, we write for simplicity $clock_p$, T_p , and seq_p instead of $clock_p^C$, T_p^C , and seq_p^C respectively.

4 Layered architecture for optimistic causal delivery

This section proposes and studies a layered protocol for achieving optimistic causal delivery. Here we assume that coordinators of a cluster are assigned to vector entries and that the coordinators of a cluster know each other. To satisfy these requirements we choose a decentralised and fault-tolerant cluster-management protocol [11] which can map a process to a unique identifier in the cct-vector clock in a decentralised way and can inform all processes in $Core_C$ about this mapping.

Section 4.1 introduces the layered approach while Section 4.2 presents and analyses the method for ensuring fault-tolerance and enhancing the throughput of the protocol.

4.1 Protocol description

The first of the two layers uses *PrCast* in order to multicast events inside the cluster (cf. pseudo-code description Algorithm 1). The second layer implements the optimistic causal delivery service. The consistency protocol is inspired by the causal consistency protocol by Ahamad et. al. [1] and is adapted and enhanced to provide the optimistic delivery service of the cluster consistency model and the recovery procedure for for events that may be missed due to *PrCast*.

Each process in a cluster interested in observing events in optimistic causal order (which is always true for a coordinator), maintains a queue of events denoted by H_p^C . For any arriving event e one can determine from T_p^C and the event's timestamp t_e whether there exist any events which (i) causally precede e , (ii) have not been delivered, and (iii) are still deliverable according to the optimistic causal order property. More precisely we define this set of not yet delivered deliverable events as

$$to_deliver_before(e) = \{e' \mid t_{e'} < t_e \wedge \neg(t_{e'} < T_p^C)\}$$

and their event ids, which can be used for recovery, can be calculated as follows

$$to_deliver_before_ids(e) = \{\langle i, j \rangle \mid (\forall i \neq index(e) . T_p^C[i] < j \leq t_e[i]) \vee (i = index(e) \wedge T_p^C[i] < j < t_e[i])\}.$$

If there exist any such events, e will be enqueued in H_p^C for a time interval until e may become obsolete (at the end of this interval, process p may need to “pull” missing events —see below). Otherwise, p delivers e to the application.

When a process p delivers an event e referring to cluster C , the logical cct-vector clock $clock_p^C$ is updated by setting $\forall i T_p^C[i] = \max(t_e[i], T_p^C[i])$. Process p also checks whether any events in H_p or recovered events now can be dequeued and delivered.

When a coordinator p in $Core_C$, owning the j th vector entry, multicasts an event it will update $clock_p^C$ by incrementing seq_p^C by one. Then, p creates a vector timestamp t with $t[i] = T_C^p[i]$ for $i \neq j$ and $t[j] = seq_p^C$.

Since PrCast delivers events with high probability p , there may be processes which will need to recover events. In our protocol this is done via a pull scheme, where the process which is missing an event e is requesting it from other(s). A process p invokes the recovery procedure when some event e in H_p risks to become obsolete. Then p issues recovery messages for the missing events that precede e . The time before e becomes obsolete depends on the delay of e , i.e the amount of time passed since the start of the dissemination, and is assumed to be larger than the duration of a PrCast (which is estimated by the number of hops that an event needs before it has reached all destinations with w.h.p. using PrCast) and the time it takes to send a recovery message and receive an acknowledgement. Before $e \in H_p$ becomes obsolete, p delivers e , all causally preceding events in H_p , and causally preceding recovered events by respecting their causal relation.

A simple way to do the recovery is by contacting the sender of each event. For this purpose the sender can maintain a *recovery buffer* in which it stores each event until no more recovery messages are expected (this is for example the case if $\forall i t_e[i] < T_p^C[i]$).

A method that enhances the throughput and the fault tolerance of the recovery is given in the following subsection (4.2), accompanied by an analysis of the guarantees and the required resources.

Properties of the protocol. The PrCast protocol provides a delivery service which guarantees that an event will reach all its destinations with high probability, i.e. PrCast can achieve high message stability. However, if recovery of events needs to be performed, the number of processes which did not receive the event is expected to be low. This means that a process is only expected to receive a low number of recovery message when it multicasts an event. If the processes do not encounter process failures, link failures or timing failures, reliable point to point communication succeeds in recovering all causally preceding missing events, and thus provide causal order without any message loss. The following lemma is straightforward, following the analysis in [1].

Lemma 1 *An execution using the two-layer protocol guarantees causal delivery of all events disseminated to a cluster if neither processes nor links are slow or fail.*

4.2 Event recovery procedure, fault tolerance and throughput

As mentioned above, since PrCast delivers events with high probability, there may be processes which will need to recover e . In the version described in the previous subsection, the recovery is done via the sender p of the event, who maintains a recovery buffer of the events it has sent. Besides this being a simple scheme, it can also give a point of reference for the recovery rate of events.

The throughput and fault tolerance of the protocol can be increased by introducing redundancy. In this case processes are required to keep a history of some of the observed events for a while. A process sending a *RECOVER* message needs only to contact a fixed number of other processes in order to receive the respective update. Further, the recovery of failing processes could make use of such redundancy. For obvious reasons it is desirable to bound the size of this buffer. Below we make an analysis to get the recovery buffer size and number of processes to contact so that the recovery succeeds with high probability.

Algorithm 1 Two-Layer protocol

VAR

H_p : set of received events that can not be delivered yet
 R : set of recovered events that can not be delivered yet
 B : fixed size recovery buffer with FIFO replacement.

On p creates e

/ Create timestamp t_e */*
 $t_e := T_p^C$; $t_e[p] := seq_p^C$; $seq_p^C := seq_p^C + 1$
PrCast($\langle e, t_e \rangle$)
Insert e into recovery buffer B

On p receives $\langle e, t_e \rangle$

Insert e into recovery buffer B
if e can be delivered **then**
 deliver(e)
 for all $e' \in H_p \cup R$ that can be delivered
 deliver(e')

else

if e is not delivered or obsolete **then**
 delay(e , time_to_terminate)

end if**end if****On** timeout(e , time_to_terminate)

for all $eid \in to_deliver_before_ids(e)$ not in $H_p \cup R$ and eid not already under recovery
 send($\langle RECOVER, eid \rangle$) to source(eid) or to k arbitrary processes in cluster
delay(e , time_to_recover)

On timeout(e , time_to_recover)

for all $e' \in to_deliver_before(e) \cap (H_p \cup R)$ that can be delivered
 deliver(e')
 deliver(e)
for all $e' \in H_p$ that can be delivered
 deliver(e')

On p receives $\langle RECOVER, source(e'), eid \rangle$

if p has e with identifier eid in its buffer **then**
 respond($\langle ACKRECOVER, e, e_t \rangle$)
end if

On p receives $\langle ACKRECOVER, e, e_t \rangle$

Insert e into recovery buffer
if e can be delivered **then**
 deliver(e)
 for all $e' \in R \cup H_p$ that can be delivered
 deliver(e')

else

if e is not delivered or obsolete **then**
 $R := R \cup \{e\}$

end if**end if****On** deliver(e)

/ Update T_p^C */*
 $\forall i T_p^C[i] := \max(t_e[i], T_p^C[i])$
Remove e from R and H_p
Deliver e to the application

Following [13], first we describe a model suitable to determine the probability for availability of events that are deliverable and may need recovery in an arbitrary system consisting of a cluster C of n processes that communicate using the Two-Layer protocol. Let \mathcal{C} denote this system and T denote

the time determined by the number of rounds an event stays at most in \mathcal{C} . We observe the relation of the buffer system to a single-server queueing system, where new events are admitted to the queue as a random process. However, unlike common queueing systems, the service time (time needed for all processes in \mathcal{C} to get the event using the layered protocol) in this model depends on the arrival times of events. The service time is such that every event stays at least as long in the queue as it needs to stay in the buffer of \mathcal{C} in order to guarantee delivery/recovery (i.e. whether the queue is stable is not an issue here). Below we estimate the probability that the length of the queue exceeds the choice of the length for the recovery buffer of \mathcal{C} .

If a_i denotes the arrival time of an event e_i , the “server” processes each event at time $s_i = a_i + T$. Observe that if the length of the buffer in \mathcal{C} is greater than the maximum length of the queue within the time interval $[a_i, s_i]$ then \mathcal{C} can safely deliver e_i .

Consider $[t_a, t_s]$ denoting an interval of length T and the random variable $X_{i,j}$ denoting the event that at time $t_a + i$ process j inserts a new event in the system. Further, assume that all $X_{i,j}$ occur independently, and that $\Pr[X_{i,j} = 1] = p$ and $\Pr[X_{i,j} = 0] = 1 - p$. The number of admitted events in the system can be represented by the random variable $X := \sum_{j=1}^n \sum_{i=1}^T X_{i,j}$, hence the random process describing the arrival rate of new events is a binomial distribution and the expected number of events in the queue in an arbitrary time interval $[t_a, t_s]$ equals

$$\mathbf{E}[X] = npT.$$

Clearly, the length of the recovery buffer must be at least as large as $\mathbf{E}[X]$, or we are expected to encounter a large number of events that cannot be recovered.

Now we bound, using Chernoff bound [13, 19], the buffer size so that the probability for events that need recovery be not present in the recovery buffers of arbitrary processes becomes low.

Theorem 1 *Let e be an event admitted to a system \mathcal{C} executing the two-layered protocol, where each event is required to stay in \mathcal{C} for T rounds. Each of the n processes in the system admits a new event to \mathcal{C} in a round with probability p . Then \mathcal{C} can guarantee the availability of e in the recovery buffer of an arbitrary process with probability strictly greater than*

$$1 - \left(\frac{e}{4}\right)^{npT}$$

if the size of the buffer is chosen greater than or equal to $2npT$.

Proof: Following the Chernoff bound for binomial distributions, for any $\delta > 0$ it is the case that

$$\Pr[X > (1 + \delta)npT] < \left(\frac{e^\delta}{(1 + \delta)^{\delta+1}}\right)^{npT}.$$

By choosing $\delta = 1$, the result follows. □

It is also possible more generally assume that each process i admits a new event to the system with probability p_i . If

$$\mu = \sum_{i=1}^n p_i$$

a buffer size of length $2\mu m$ guarantees availability of an event with a probability strictly greater than $1 - \left(\frac{e}{4}\right)^{\mu T}$. In the experimental evaluation in the following section we used the formula as described in Theorem 1 in order to estimate the safe buffer sizes.

For estimating T , we can use the estimation of time of a PrCast, e.g. as in [13]. Let PrCastTime denote this time. An event e is likely to be needed in \mathcal{C} for (i) PrCastTime rounds (to be delivered to all processes with high probability); (ii) plus PrCastTime rounds, if it is missed by some process to be detected as missing by the receipt of a causally related event (note that this is relevant under high load, since in low loads PrCast algorithms are even more reliable); (iii) plus the time interval $time_to_terminate + time_to_recover$ that the algorithm takes before and after issuing recovery messages.

Furthermore, considering that processes may fail by crashing, the protocol can help by having a process that needs to recover event(s) contact a number of other processes to guarantee recovery with high probability.

Consider that processes may fail independently with probability p_f . Let X_f be the random variable denoting the number of faulty processes in the system. Then

$$\mathbf{E}[X_f] = p_f n.$$

By applying the Chernoff bound as in theorem 1 we get:

Lemma 2 *If, in a system of n processes where each one may fail independently with probability p_f , we consider an arbitrary process subset of size greater than or equal to $2np_f$, with probability strictly greater than*

$$1 - \left(\frac{e}{4}\right)^{np_f}$$

there will be at least one non-failed process in the subset.

This implies that a for a process requesting recovery from $R = 2p_f n$ processes ensures w.h.p. that there will be at least one non-faulty to reply.

Corollary 1 *In a system of n processes where each one may fail independently with probability $p_f \leq k/(2n)$ for fixed k , an arbitrary process that needs to recover events according to the Two-Layer protocol, will get a reply with high probability if it requests recovery from k processes.*

Notice further that by requesting recovery only once and not needing to propagate the recovery messages is good from the point of view that in cases of high loss due to networking problems we do not flood the network with recovery messages. Compared to the method of recovery from the originator of an event, this method may need k times more messages during each recovery. However, this comes with the advantage of tolerating failures and process departures, as well as distributing the load of the recovery in the system.

Regarding the events replacement in the recovery buffer, the simplest option is to use FIFO replacement. Another option is to use an aging scheme based e.g. on the number of steps that the event has made since it was generated. As was shown with in [13], an aging scheme results to even improved performance from the reliability point of view. However, to employ such a scheme here it seems we might need to sacrifice the transparency between the two layers —but it is of course possible, if we assume this information is passed from the underlying dissemination service to the consistency service. Instead, it is also worth noticing that using a dissemination algorithm such as the ETTB [13], which uses an aging method to remove events from process buffers and which guarantees very good message stability, implies that the reliability is improved via the fact that fewer processes may need to ask for recovery of events.

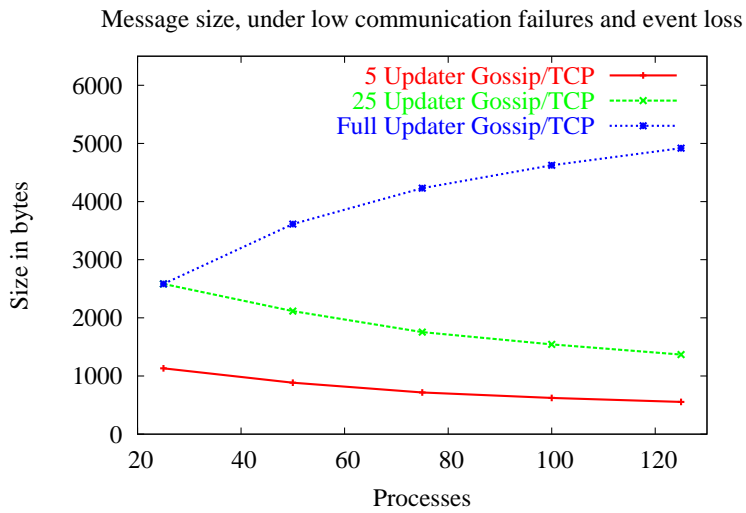
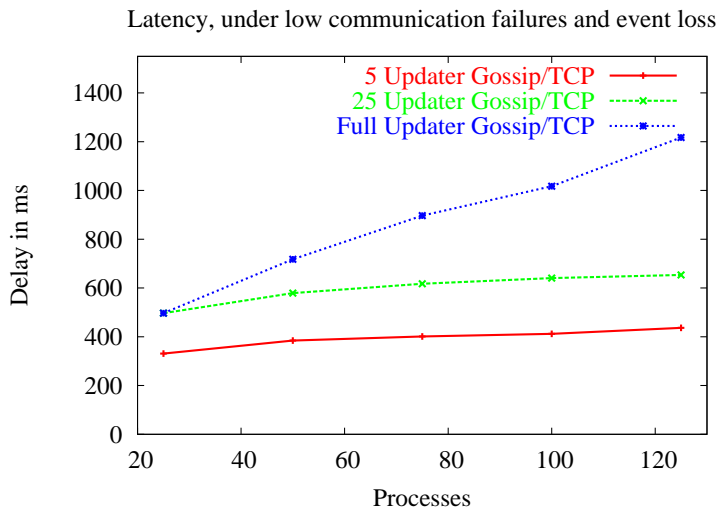
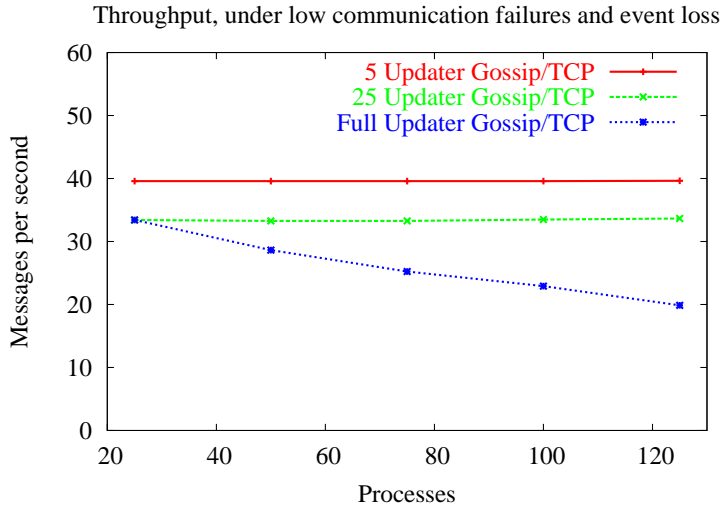


Figure 1: Throughput and latency with increasing number of cluster members

5 Experimental evaluation

The experiments described in this section study the effect of clustering with respect to scalability. Moreover, we evaluate the reliability and throughput effects of using the optimistic causality layer of the Two-Layer protocol by comparing the gain in message/event loss and throughput of the optimistic causality layer. Throughout the section we refer to a message/event as lost if it was not received or could not be delivered without violating optimistic causal order. We will also refer to the optimistic causality layer as the causality layer for brevity.

The evaluation of the Two-Layer protocol is based on data received from a network experiment on up to 125 computers at Chalmers University of Technology. The computers were running the Linux or Solaris 9 operating systems. The hardware varied among the machines (Sun Ultra10 and Sun Blade workstations, PC's with Pentium4 processors, multiprocessor PC's, all with memory ranging from 128MB up to 4048MB). The computers were distributed over a few different subnetworks of the university network. Depending on the network load a IP-ping message of 4KB size had an average round-trip-time between 1ms and 5ms. As we did not have exclusive access to the computers and the network, the experiment had to coexist with other users concurrently running applications which potentially might have made intensive use of the network.

We have implemented the layered architecture in an object oriented, modular manner in C++. The implementation of the causality layer follows the description in Sections 4.1 and 4.2 and can be used with several group communication objects within our framework. The PrCast implementation we use follows the outline of the ETTB-dissemination algorithm described in [13] together with the membership algorithm of lpbcast [9]. The message transport can use either TCP or UDP. For connection oriented communication a timeout can be specified to ensure that a communication round has approximately the same duration for all processes.

Our first experiment evaluates how the number of group coordinators affect throughput, latency and message sizes. The results are based on a test application which can act either as a coordinator or as an ordinary group member. The test application runs in rounds of the same duration as the PrCast protocol does. When the process acts as an updater it produces a new event with probability p . In this experiment the product of updaters and p was kept constant (at 6). The configuration parameters of PrCast in this experiment were set to satisfy the goal of each event reaching up to 250 processes with high probability. This was set so, in order to eliminate event loss at the dissemination layer, so as to be able to focus on the performance measures of the consistency layer. The fanout was 4, while the termination time for an event was 5 hops. PrCast was allowed to keep track of all members to avoid side effects of the membership scheme in this experimental study. The maximum number of events which can be transported in one gossip message was limited to 20 events, while the number of group members transported was limited to 10. The size of the history buffer was set to 40 events, which is according to [13] high enough to prevent with high probability the same event from being delivered to the causality layer multiple times. The gossip messages were sent using connection oriented communication. This allowed us to tune the duration of each gossip round so that all experiments had approximately the same rate of failures for communication links. In each round a process sends its gossip messages concurrently by using multi-threading, so the round time could be adapted to be a fixed-time interval longer (10ms) than the expected maximum message delay.

Figure 1 compares three instances of the Two-Layer protocol. In the full-updater experiment all processes acted as coordinators, while in the 5-updater and the 25-updater experiments the number of coordinators was restricted to 5 and 25 respectively. The protocols were tuned by adjusting the connection timeout such that the average connection loss rate did not exceed 0.2%. The causality

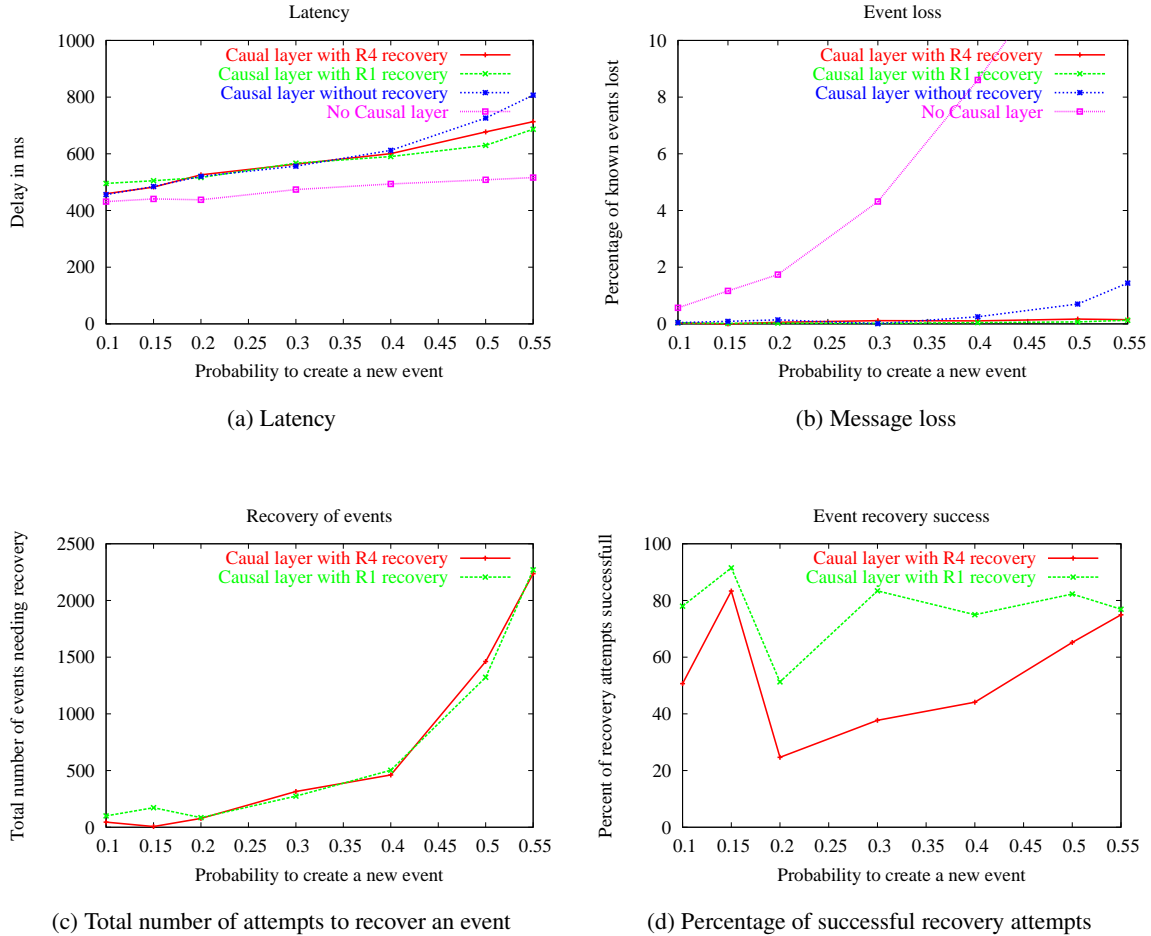


Figure 2: Event latency, loss and recovery behaviour under varying load with and without the causality layer.

layer used the recovery method described in Section 4.1. The data shows the impact of the size of the vector clock on the overall message size. For the protocols using a constant number of coordinators message sizes even decreased slightly with growing group size since the dissemination distributes the load of forwarding events better for large group sizes, i.e. for large groups a smaller percentage of processes performs work on an event during the initial gossip rounds. However, for the full updater protocol messages grow larger with the number of coordinators. This behaviour directly influences the observed results with respect to latency and throughput. For growing group size the protocols using a constant set of coordinators experience only a logarithmic increase in message delay and throughput remains constant. The message delay for the full updater protocol increases linearly while throughput decreases.

The second experiment studies the effects induced by the causality layer and the recovery scheme used in the Two-Layer protocol. Figure 2 compares the gossip protocol, the Two-Layer protocol with and without recovery. The recovery is done in two ways: (i) from the originator, as described in Section 4.1 (curves marked by "R1 recovery") and (ii) from arbitrary processes, as described in Section 4.2 (curves marked by "R4 recovery", as k equals 4, similar to the fan-out in the dissemination

protocol). The recovery buffer size is following the analysis in section 4.2, with the timeout-periods set to the number of rounds of the PrCast. Here, unlike the first experiment, the number of processes and updaters was fixed to 25; instead we evaluated the system for varying values of p , to be able to study the reliability of the consistency layer under varying load in the system. As we read the diagrams from left to right, larger values for p imply increased load in the system, having at the rightmost side approximately $n/2$ new events being multicasted in each round. As the load increases, the dissemination layer reorders events quite a lot and, to the rightmost side, it starts to experience message loss due to buffer overflows, thus putting the causality layer protocols under stress.

The measurements in Figure 2 show that the causality layer reduces the amount of lost (ordered) events, the difference being all the more significant when the number of events disseminated in the system is high. By using the recovery scheme almost all events could be delivered in optimistic causal order, both in R1 and R4 recovery. With increasing event probability latency grows only slowly, thus manifesting scalability. The causality layer adds a small overhead by delaying events in order to respect the causal order. The recovery scheme does not add significant overhead with respect to latency, however it significantly reduces the number of lost events. For some instances it was even possible to observe a faster event delivery when using the recovery scheme. One reason is that by recovering the missing events causally subsequent events in H_p do not need to be held until they time out before being delivered.

Figure 2(c) shows the total number of attempts to recover events in the system and Figure 2(d) shows the success rate for these recovery attempts. As expected the number of recovery attempts increase as the load on the system increases, with very few events needing to be recovered when the load is low. There are three likely causes for a event recovery to fail: (i) the reply arrives too late; (ii) the process(es) asked do not have the event; and (iii) the reply or request(s) messages were lost. The unexpectedly low success rate for recoveries using the R4 method during low load could be because the PrCast might reach very few processes if gossip messages are lost early in the propagation of an event. Further, one should keep in mind that as the system load is low the number of missing events and recovery attempts is small. However, when the load and the number of recovery attempts increase the success rate starts to converge towards the predicted outcome.

6 Discussion and future work

We have proposed lightweight causal cluster consistency, a hierarchical layer-based structure for multi-peer collaborative applications. This is a general architecture, can be applied on top of the standard Internet transport-layer services, and can offer a layered set of services to the applications that need them.

Causal cluster consistency conforms to a dynamic interest model for processes on replicated objects. Processes can observe updates which correspond to their interest in optimistic causal order. Moreover, a dynamic set of processes may concurrently propose updates to the state of the replicated objects. Our proposed approach for reliable ordered message delivery pairs well with recent approaches in multi-peer information dissemination, where the aim is at predictable reliability, i.e. that each event is delivered (in the required order here) to all non-faulty destinations with a high-probability guarantee.

We also presented a two-layer protocol for causal cluster consistency running on top of decentralised probabilistic protocols supporting group communication. Our experimental study, conducted by programming and evaluating the proposed architecture as a two-layered protocol that uses standard Internet transport communication, shows that the approach scales well, imposes an even load on the

system, and provides high-probability reliability guarantees.

Further work includes complementing this service architecture with other consistency models such as total order delivery with respect to objects. Considering object ownership and caching is another topic that is worth studying.

References

- [1] M. Ahamad, G. Neiger, P. Kohli, J. E. Burns, and P. W. Hutto. Casual memory: Definitions, implementation and programming. *Distributed Computing*, 9:37–49, 1995.
- [2] L. O. Alima, A. Ghodsi, P. Brand, and S. Haridi. Multicast in DKS(N; k; f) overlay networks. In *Proceedings of the 7th International Conference on Principles of Distributed Systems (OPODIS '03)*, volume 3144 of *LNCS*, pages 83–95. Springer-Verlag, 2003.
- [3] S. Baehni, P. T. Eugster, and R. Guerraoui. Data-aware multicast. In *Proceedings of the 5th IEEE International Conference on Dependable Systems and Networks (DSN 2004)*, pages 233–242, 2004.
- [4] R. Baldoni, R. Prakash, M. Raynal, and M. Singhal. Efficient Δ -causal broadcasting. *International Journal of Computer Systems Science and Engineering*, 13(5):263–269, 1998.
- [5] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, Aug. 1991.
- [6] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, May 1999.
- [7] K. P. Birman and T. A. Joseph. Reliable communication in the presence of failure. *ACM Transactions on Computer Systems*, 5(1):47–76, Feb. 1987.
- [8] C. Carlsson and O. Hagsand. DIVE - a multi-user virtual reality system. In *Proceedings of the IEEE Annual International Symposium*, pages 394–400, Seattle, Sept. 1993.
- [9] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2001)*, pages 443–452, July 2001.
- [10] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Scamp: Peer-to-peer lightweight membership service for large-scale group communication. In *Proceedings of the Third International COST264 Workshop*, volume 2233 of *LNCS*, pages 44–55. Springer-Verlag, 2001.
- [11] A. Gidenstam, B. Koldehofe, M. Papatrantafileou, and P. Tsigas. Dynamic and fault-tolerant cluster management. Technical Report 2005-10, Computer Science and Engineering, Chalmers University of Technology, Apr. 2005.
- [12] C. Greenhalgh and S. Benford. A multicast network architecture for large scale collaborative virtual environments. In *Proceedings of the Second European Conference on Multimedia Applications, Services and Techniques - ECMAST'97*, volume 1242 of *LNCS*, pages 113–128. Springer-Verlag, 1997.
- [13] B. Koldehofe. Buffer management in probabilistic peer-to-peer communication protocols. In *Proceedings of the 22nd Symposium on Reliable Distributed Systems (SRDS '03)*, pages 76–85. IEEE, Oct. 2003.
- [14] A. D. Kshemkalyani and M. Singhal. Necessary and sufficient conditions on information for causal message ordering and their optimal implementation. *Distributed Computing*, 11:91–111, 1998.
- [15] L. Lamport. Time, clocks, and the ordering of events in a distributed system. In *Communications of the ACM*, volume 7 of 21, pages 558–565, July 1978.
- [16] N. Lynch, D. Malkhi, and D. Ratajczak. Atomic data access in distributed hash tables. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, volume 2429 of *LNCS*, pages 295–305. Springer-Verlag, 2002.
- [17] F. Mattern. Virtual time and global states of distributed systems. In *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226, 1989.
- [18] D. C. Miller and J. A. Thorpe. SIMNET: the advent of simulator networking. In *Proceedings of the IEEE*, volume 8 of 83, pages 1114–1123, Aug. 1995.
- [19] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, June 1995.
- [20] J. Pereira, L. Rodrigues, M. Monteiro, and A.-M. Kermarrec. NEEM: Network-friendly epidemic multicast. In *Proceedings of the 22nd Symposium on Reliable Distributed Systems (SRDS '03)*, pages 15–24. IEEE, Oct. 2003.

- [21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM Computer Communication Review*, volume 31, pages 161–172, 2001.
- [22] M. Raynal, A. Schiper, and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Information Processing Letters*, 39(6):343–350, Sept. 1991.
- [23] L. Rodrigues, R. Baldoni, E. Anceaume, and M. Raynal. Deadline-constrained causal order. In *Proceedings of the Third IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2000)*, Mar. 2000.
- [24] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 2218 of *LNCS*. Springer-Verlag, Nov. 2001.
- [25] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM 2001 Conference*, pages 149–160, New York, Aug. 2001. ACM Press.
- [26] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, and A. D. Joseph. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53, 2004.