Multicast-based Runtime System for Highly Efficient Causally Consistent Software-only DSM

Thomas Seidmann

Simultan AG, Switzerland [**]
tseidmann@simultan.ch
http://www.simultan.ch/~thomas/

Abstract. This paper introduces the application of IP multicasting for enhancing of software-only DSM systems and, at the same time, simplification of the programming model by offering a simple memory consistency model. The described algorithm is the foundation of a runtime system implemented as filesystems for the Windows NT and FreeBSD operating systems.

Keywords: distributed shared memory, DSM, memory coherence protocol, IPv4/IPv6 multicasting, causal consistency, vector logical clock

1 Introduction

Software distributed shared memory (DSM) realized on a network of workstations connected through a conventional computer network has gained a lot of attention in both research and industry. On the other hand group communication, also known as multicasting, on IP networks is getting widespread over the Internet. The new IP generation, IPv6, relies heavily on the availability of multicasting. This paper presents the idea of an application of IP multicasting for the purpose of coherence traffic in software DSMs.

2 Related Work

Speight and Bennett reported in [5] about the use of multicasting in the Brazos project. There main differences between the Brazos approach and ours are the following:

1. DSM in Brazos is concentrated on scope consistency, that means a memory consistency model using synchronization variables; our approach is to provide causal consistent DSM.
2. Brazos relies implicitly on reliable multicast, whereas we do not.

The Orca distributed system [1] uses totally ordered group communication for a MRMW write-update coherence protocol. The protocol employs a centralized component - the sequencer - to achieve totally ordered multicast on top of the potentially unreliable IP multicast. In our design this centralized component is avoided.

[**] This research was supported in parts from the Slovak University of Technology in Bratislava under Grant VEGA 1/4289/97

3 Multicast-based Coherence Protocol for Causally Consistent DSM

The main requirements for the new multicast-based DSM coherence protocol are the following:

- Delivery of causal consistent memory.
- Use of IP multicast in its present (IPv4 and IPv6) shape, that means with no guarantee of ordering and reliability.
- Avoidance of central components like managers in Brazos or sequencers in Amoeba; the protocol should be completely distributed.

Although in the following text we use the term "page" denoting hereby the granularity of memory sharing (the first implementation is being developed for a page-based DSM), the protocol itself can be used with other types of DSM as well, for example with object-based DSM.

3.1 Components of the Memory Coherence Algorithm

The proposed protocol is basically a MRMW protocol with write-update using multicast transfer of diffs between the involved processes. The causality relationship between shared memory operations is achieved by means of **vector logical clocks** [3], which represent the basic building block in the algorithm. Every process maintains for every shared page a vector logical clock value - timestamp - consisting of:

- The process's own logical (Lamport, monotonic) clock value of the last write operation upon the page.
- Other known processes' logical timestamps of write operations upon the page.

This vector logical timestamp is transmitted within every message that concerns this particular page. In the first implementation of the DSM (see below) the vector logical timestamp is represented with a fixed array of logical timestamps, meaning the group of processes using the DSM is closed and must be known in advance. In a future enhancement, in order to overcome this limitation, the representation of vector logical timestamps will be changed to an associative array consisting of pairs $(PID, value)$, where PID denotes the process' global ID composed of the node's global ID (for example IP address) concatenated with the node local PID.

Every shared page in the DSM must be uniquely identified. This page ID is simply referred to as the page number. In the runtime system described in section 4 the page number can easily be calculated from the offset within the file representing an application's shared memory space.

Every process is responsible for keeping around the last m ($m >= 1$) diffs resulting from its own write accesses for every modified page.

3.2 Initialization of the Algorithm

When a process P wants to access a shared page which is not present locally, the page's vector logical timestamp is initialized with zero values. A new page initialized with zeroes is allocated from the operating system. The further action depends on the type of access to the page.

- If the access is a **write** access, it is performed and the element in the page's logical vector timestamp denoting the local process timestamp is advanced by one and a multicast message requesting the page is sent containing the page ID, the actual vector logical timestamp and the diff freshly produced by the write operation.
- In case of a **read** operation a multicast message denoting a request for the page is sent containing the page ID and the page's actual vector logical timestamp. The read operation returns zero without blocking the caller. Note that this kind of multicast message is also sent asynchronously to any accesses to the page upon expiration of a timer as a part of the mechanism for lost messages recovery; this mechanism is described in section 3.4.

The reaction of processes which receive the message specified above is described in section 3.3

3.3 Scenarios in the Algorithm

Every process must perform some specific action upon the occurrence of two events: writing to a shared page and receipt of a multicast message with a requests and/or a diff for a page. These actions are described in the following paragraphs.

Write access to a shared page. Every process owning a copy of a shared page and performing a write access notifies all other processes of the performed operation by advancing its own logical clock for the page by one and sending out a multicast message containing the page ID, the page's current vector timestamp logical clock and the diff for the page produced by the write operation. The diff is recorded as specified in section 3.1, perhaps causing an older diff to be purged. This means that every write access must notify the runtime system, that is a context switch to kernel mode is performed, hereby increasing the cost of a write operation on a shared page.

Receipt of a multicast message with a request for a page. Any process keeping a copy of the requested page and whose vector logical timestamp of the is not smaller than the one carried with the message will reply to the message either with the missing diffs kept by the process, or a constructed diff containing the complete page if the diff isn't kept any more (the difference between the vector logical clock values is greater than m, the number of diffs kept by each process for pages modified by itself). The reply is sent in a multicast message and contains the page ID and the vector logical timestamp value of the page in addition to the diff.

Receipt of a multicast message with a diff for a page. This kind of messages are ordered according to the vector logical timestamp carried by them causing the causality relationship between accesses to the page to be respected. The use of vector logical clocks for determination of causal order of events has been described in the ISIS project (Cornell) [2]. Causally related write accesses are performed in their causal order, concurrent (not causally related) writes are performed in arbitrary order. For the purposes of ordering these messages a queue must be maintained by every process for every shared page it keeps locally. A diff to the page is ready to be applied when all diffs causally preceding it have already been applied. When a diff is applied to the contents of the page, the vector logical timestamp of the page is advanced by merging its original value with the timestamp of the diff, thus possibly making other diffs ready to be applied.

The queue if diffs yet to be applied reveals messages missed by the process, which have been seen by at least one process in the causal chain of accesses to the page. These missing diffs, which prevent a diff to be applied, are requested and replied with point-to-point messages.

3.4 Handling of Lost Messages

The previous paragraph showed us that the algorithm can cope with lost messages (both containing requests and/or diffs) as long as at least one of the other processes in the chain of causal accesses to a page has received the missing write update. However there is still a problem when this condition is not met, which is solved by a timer associated with each shared page. This timer is restarted every time a write update (a diff) for the page is received, and upon expiration of this timer a request message containing the page ID and its current vector logical timestamp is sent in a multicast message.

3.5 Summary of the Memory Coherence Protocol Algorithm

It can be shown, that the number of messages in case of no loss can be reduced by the use of multicasting compared to point-to-point communication from $O(N^2)$ to $O(N)$, which is the biggest advantage of the proposed protocol. The simple memory consistency offered by the application of the protocol can be viewed as a gain which has its prize in the rather expensive write operation – each one must be interrupted in order to formulate and send a write-update multicast packet. In case of a filesystem-form implementation of the runtime system (see next section) this means only one context switch between user and kernel mode, however.

4 Results

To examine and test the proposed DSM cache coherence protocol we've implemented it into the CVM system [4] and deployed it on a network of UNIX workstations communicating over an IPv6 network, both local and the 6bone [1]. Note that IPv6 itself is not a prerequisite, but merely a chosen characteristic of the used testbed. In the case of 6bone communication two sites were available, both with multicast routing enabled. The functionality of the protocol was verified on two well-known scientific applications, Barnes Hut (from the SPLASH suite) and FFT-3D.

The coherence-related communication is reduced compared to the sequential consistency mechanism contained in CVM in case of 5 processes by a factor in range between 4 and 5 in case of no loss of messages. Under conditions of 10% loss this factor drop down to 3 due to additional point-to-point messages.

The DSM protocol is being implemented in the form of specialized filesystems for the Windows NT and FreeBSD operating systems. The idea is appealing since it allows the use existing APIs for accessing the DSM instead of adding new ones. Both operating system families contain APIs for the filesystem-backed shared memory paradigm, namely the `MapViewOfFileEx()` family under Windows NT and `mmap()` family under the BSD dialects. The implemented filesystems have two significant features:

- The data contained by them are never serialized to stable storage, except by the applications themselves.
- Although the filesystems can be accessed with "normal" file access APIs, this is not the intended use.

The filesystem-form of implementations has the advantage of reducing the overhead of a write operation on a shared page, since all work related to memory coherence can be done in kernel mode and thus require only a single transition between user and kernel mode and back.

Clearly, various files opened via these filesystems correspond to different applications using the DSM. Every application is supposed to create a separate group for multicast communication.

5 Conclusions and Future Work

The results we have experienced so far make us believe of a right direction in our research. There are a couple of tasks we plan to take on in near future:

- **Formal specification and verification:** We are currently dealing with formal verification of the proposed (and other) protocols by means of the Z specification language.
- **Simulation of the memory coherence algorithm** under FreeBSD-current, which contains a framework called "dummynet" primarily intended for network simulations.
- **Overhead determination** of various effects of the proposed algorithm: group communication, additional context switches etc.
- **Open group of involved processes** as specified in section 3.1.

[1] experimental IPv6 network layered on top of the Internet

- **Group membership:** One application is using one group in the sense the Internet Group Membership Protocol (IGMP – in IPv4) or Multicast Listener Discovery (MLD – in IPv6). This behavior could be changed to use more groups and thus keep processes from receiving unnecessary interrupts. We try to elaborate some algorithm for this. An extreme solution that would be a separate group for each shared page.
- **Fault tolerance:** The current version of the DSM coherence protocol doesn't cope with the failure of nodes; we investigate this topic.
- **Security:** Since the some of the current IPsec protocols are not suitable for group communication (for example the ESP header), we investigate the means of achieving privacy for the transfer of coherence-based traffic.

References

1. Henri E. Bal, Raoul Bhoedjang, Rutger Hofman, Ceriel Jacobs, Koen Langendoen, Tim Ruehl, and M. Frans Kaashoek. Performance evaluation of the orca shared object system. *ACM Trans. on Computer Systems*, February 1998.
2. K. P. Birman. The process group approach to reliable distributed computing. *Commun. of the ACM*, 36:36–54, December 1993.
3. Randy Chow and Theodore Johnson. *Distributed Operating Systems & Algorithms.* Addison Wesley Longman, Inc., 1997.
4. Pete Keleher. Cvm: The coherent virtual machine. Technical report, University of Maryland, July 1997.
5. W. E. Speight and J. K. Bennett. Using multicast and multithreading to reduce communication in software dsm systems. In *Proc. of the 4th IEEE Symp. on High-Performance Computer Architecture (HPCA-4)*, pages 312–323, February 1998.