Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

I. Shizgal

An amoeba replicated service organisation

Computer Science/Department of Algorithmics & Architecture Report CS-R8723 May

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

An Amoeba Replicated Service Organisation

Irvin Shizgal
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

A technique is described which allows replicated instances of servers on the Amoeba operating system to dynamically detect other currently active server sites, and for these to cooperate in the maintenance of replicated data.

CR Categories: C.2.4 D.4.4
Keywords & Phrases: Amoeba, multicast, 2-phase commit, distributed service

1. Introduction

A distributed algorithm is described which allows replicated instances of an Amoeba server at distinct sites to dynamically locate other currently active server sites, and for these to cooperate in the maintenance of replicated data. The technique eliminates dependence on other services such as name servers, requires no multicast support, and does not assume the availability of any other means of information sharing, such as a common file system.

1.1. Background on Amoeba

"Amoeba" is a distributed operating system designed for use on a local area network [7, 6]. It is based on the use of synchronous (i.e. unbuffered) message transactions, resembling remote procedure calls, as its main inter-process communication primitives.

1.1.1. Amoeba inter-process communication primitives

An Amoeba transaction is based on a client-server relationship. A server on the network issues a 'get_request(Port_number, Message_buffer)' system call, which blocks until a request is received. A client process issues a 'transaction(Port_number, Request_buffer, Reply_buffer)' system call which sends a message to a server waiting on the same port number, and blocks pending receipt of the reply message. (Note that the actual call arguments are slightly more complicated, but have been simplified here for purposes of illustration.) The port numbers are abstract network addresses from a very large address space not corresponding to the physical network address space. In particular, a port number does not fix the location of a server accepting messages on it, and, indeed, multiple processes at varying physical addresses may be simultaneously listening for requests on the same Amoeba port. The Amoeba transaction semantics guarantee that at most one server waiting for a request on a given port will respond to a transaction request on that port.

Report CS-R8723
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

1.1.2. Clusters and tasks

Since the primary Amoeba system calls block the caller, concurrency must be achieved by partitioning a job among a number of distinct subprocesses. To support this efficiently, Amoeba provides two types of process. The "heavy-weight" process (referred to in Amoeba as a cluster) is the level at which a virtual address space is allocated. Within a cluster is a set of "light-weight" processes (referred to as tasks) whose state consists only of a thread of execution and a transaction state. Tasks within a cluster share the same address space, and are never spread across machine boundaries. The overhead of starting and switching between tasks within a cluster is low, since no adjustment of the memory map or duplication of memory segments is done, and task switching is always done synchronously (i.e. only at system calls). The switching between clusters is done in the more traditional preemptive manner, and is a more expensive operation. Thus the light/heavy nomenclature reflects the costs of the two types of process.

1.2. A server organisational technique for Amoeba

This paper describes a technique for organising a group of distinct server instances to support a general service which keeps replicated data in each server site. The use of replicated data is intended to provide a service which can keep functioning normally in the face of a limited number of machine failures. (A failed machine is defined here as one which has failed completely, a condition indistinguishable from being unable to send or receive messages.) The intent is to provide an organisational structure which allows individual server instances, started at random times and various physical locations on the net to organise themselves dynamically into a structure capable of supplying a useful service. In the following discussion it is important to keep in mind the distinction between the service being provided, and a specific server site (or instance) which will typically be implemented by a set of closely cooperating processes or tasks on a particular machine.

The experiments with the organisation described here were done on 320xx-based workstations running UNIX† with Amoeba transaction support added to the kernel. The replicated service organisation described allows requests to be addressed to:

- any one of the servers offering the service (the basic type of access supported by the Amoeba semantics),
- a uniquely identified server instance,
- a subset of the servers cooperating to supply the service by supporting an enumeration of the unique service ports assigned to the server sites.

2. GENERAL OVERVIEW

The general scheme calls for a server to be implemented by a team of at least three processes. One process is assigned to listen to a unique port identified with the particular server instance. Another process listens to a well-known public port which allows general access to the service. A third process is required to listen for requests on an "enumeration port" whose value is known to all server instances (if not necessarily to their clients.)

The details of the structure would differ slightly in an implementation on a native Amoeba system. The low cost of introducing additional tasks under Amoeba would make it comparatively inexpensive to use additional ports even for inter-Amoeba task communication. (The reader will have noted that since the Amoeba primitives are blocking, a separate task is necessary for each port in use concurrently.) The additional benefit of a common address space would then allow communicating tasks in the same cluster to avoid the overhead of copying messages, using the operating system primitives only for signaling and synchronisation. In the Unix implementation of the enumeration scheme exactly three processes were used:

† UNIX is a Trademark of AT&T Bell Laboratories.

(1) The main server process. This process listens to its unique port. Its requests come from either its enumeration auxiliary process, its public listener auxiliary process, or from a client who knows its unique port.

(2) A public listener process. This process listens to the well-known public service port. It merely transcribes requests from the well-known public service port to its associated main server's private port, which it knows.

(3) An enumeration process. This process listens to the common enumeration port. It also knows the unique private port of its associated main server.

3. Algorithms

In this section the basic algorithms are outlined, and some potential problem areas are discussed. Section 3.1 describes the general mechanism used by clients to access the service. Section 3.2 outlines the basic self-propagating enumeration procedure. Section 3.3 extends this to support a weighted-voting distributed database scheme. Further sections go on to discuss crash recovery, generalisation of the scheme, and some possible problems.

3.1. General service access

A user wishing to use the service executes a transaction with the well-known public service port associated with the general service organisation. The Amoeba semantics guarantee that exactly one of the "public listener" processes receives the request. The public listener then passes on the request to the private port of its associated main service process, blocking until the request is completed, at which time it responds to the client by passing the reply back, completing the cycle.

3.2. The basic enumeration procedure

The basic enumeration procedure works by involving more and more of the enumeration sub-processes of server sites in a single ever-deepening transaction chain, until all have been exhausted. The initiator of the enumeration issues a transaction call on the enumeration port, which starts an Amoeba transaction with an enumeration process arbitrarily chosen from those available. The enumeration process thus selected repeats this action, selecting yet another enumeration process from those remaining. This chain grows, involving more and more of the available enumeration processes until there are no more to respond. Eventually an enumeration process's transaction call goes unanswered, causing it to assume that it is the last available such server. (The conditions under which this assumption fails are discussed in section 3.6.) At this point every one of the involved processes except the last in the chain is blocked in an Amoeba transaction call, awaiting a reply. The last in the chain, having failed to start a transaction, replies to its caller, and the reply ripples back along the chain freeing each waiting process.

An illustrative metaphor for the enumeration procedure is the childs' game of musical chairs. In each round of the game there is one chair less than the number of remaining participants, which effectively selects an arbitrary one of them, i.e. the one who is unable to find a chair when the music stops, and withdraws him from the game. If this process is repeated until no participants remain, then one has obtained an ordered enumeration of the original game participants.

To provide each server with the complete list of every server's private port, each enumeration process must accept, as an input argument of the enumeration call, a list of the private ports of all servers preceding it in the chain. It must add itself to the list, and pass the list on as the argument of its own transaction call. The last enumeration process in the chain then knows the complete list, which it can return to its caller. Each process in turn passes the complete list back along the chain as the reply to its caller's transaction. Thus at termination of the computation every enumeration process has an ordered list containing the private server port of every participant.

3.3. Enumeration used to implement a two-phase commit operation

A technique that can be used to perform an operation reliably on replicated data is to require that some minimum number of control sites agree to the operation in some sort of voting procedure before committing it. The idea is that the sum of the number of votes cast by sites agreeing to the operation must exceed some threshold before the decision to commit is taken [4,5].

The basic idea in using the enumeration procedure to implement such a scheme is to "piggy-back" the operation on an enumeration in such a way that the voting and first phase of the two-phase commit is done on the forward sweep of the enumeration, and the second commit phase is done on the return sweep.

The number of votes required to perform an operation, and the total number of sites are design decisions which, for the sake of discussion, are both taken as fixed at least for the duration of a commit operation. Each server on the forward sweep of the operation, receives as input arguments:

- a description of the transaction to be performed,
- the total number of votes required for a commitment,
- the total number of votes accumulated so far,
- the private port of the server site that initiated the operation.

Upon receiving this package a site enumeration server can record a phase-one commit on the transaction, and add its vote to the accumulated voting weight. If the resulting voting weight meets or exceeds the required weight, then a decision to commit can be taken, the site can proceed to promote its phase-one commit, and pass the decision back down the chain. If, on the other hand, the accumulated voting weight is insufficient to commit, the site enumeration server continues to propagate the enumeration, and commits or not according to the result returned by its nested call to the 'next' site in the chain. Two important properties of the operation are (1) site enumeration servers which have a phase-one commit are blocked until the commit/abort decision is made and returned to them, and (2) a successful operation does not incur the time-out penalty (caused by a call to a non-existent server) which is implicit in the total enumeration operation, since a successful vote need not try to propagate the partial enumeration beyond the minimum necessary number of sites.

In the absence of crashes the operation of the system is simple. On the forward pass, any of the involved site enumeration servers can easily cause an abortion of the transaction by returning an 'abort' response to its caller (the previous member of the chain). The originating site is informed of the success or failure of the operation, and can be supplied with a list of all participants, if desired.

3.4. Crash recovery

At any time, either before or during a two-phase vote and commit operation, the possibility exists of one or more processors crashing.

Failures of processors occurring before the initiation of a vote are simple to deal with, since no critical information is lost. Rather, such failures either leave enough machines active to successfully perform a transaction or they do not. If there are enough machines then there is no problem. If there are too few machines available, then the last server of an attempted enumeration + vote will have its transaction request fail (where failure to find a recipient is distinguishable from a failed transaction which has started), resulting in an 'abort' result being returned to the operation initiator. System consistency is not compromised, and all replicated information remains in a known state.

The failure of a processor supporting a server site already involved in an enumeration + commit chain is more complicated. In general such an event can be expected to leave one or more processors with an unresolved phase-one commit, and zero or more processors with a completed phase-two commit. A site enumeration server somewhere in the chain whose transaction with the next enumeration server collapses, has neither the right to remove its phase-one lock nor to promote it to a full commitment. The only processor that is guaranteed to know whether or not the decision to commit was ever taken is the processor at the end of the chain that actually made the decision. Therefore, to ultimately recover

from the effects of a crash, it must be possible to locate the site at which the final decision was made and recorded, and to base the final commit/abort decisions of all sites with phase-one commit locks on this information. This implies that the replicated information can remain in an uncertain state (i.e. it is uncertain which of two versions is actually the most recent), until the deciding site can be found and consulted. If the deciding site was the one that crashed, this can delay the stabilisation of the system state arbitrarily. It is therefore highly desirable for all participants to be able to find the site at which the final decision was made.

As described so far, the site where the final decision is made is anonymous. Since this represents a difficulty in recovering from crashes, it is desirable to make the final commit/abort decision at a site that all other participating sites know the identity of. Since, due to the nature of the enumeration procedure, all participants trivially know the identity of the initiator of the operation and can record it with the phase-one lock information, it is quite easy to have the final member of the chain call back to the operation initiator (which must have established an auxiliary local task to handle this request). The initiator site must then record the decision and return. If a disruption occurs at this point every participant has a record of the identity of the deciding site, considerably easing the job of crash recovery.

3.5. Channels

The fact that the enumeration system makes use of a special port to propagate operations to multiple servers suggests the possibility of associating more than one enumeration sub-server with each server site. If each enumeration sub-server is assigned a different port, then the different ports can each be regarded as a distinct channel. This can be used as a mechanism by which the replicated information can be partitioned. Each partition is associated with its own channel. An operation to be performed on a particular partition is then simply propagated through the appropriate enumeration channel.

3.6. Interference in channels

As in other examples of shared broadcast channels, uses of these channels are subject to mutual interference. The result of more than one instance of the above-described enumeration procedure running concurrently would be a partition of the available server instances among the concurrent enumerations, in the sense that every available server instance would be part of exactly one enumeration chain. A likely result of this is that either some or all of the colliding chains will be unable to accumulate enough votes to commit the reliable operation. Although the probability of at least one chain obtaining enough votes can be improved slightly by a careful assignment of votes to servers (which might be worthwhile in some cases in spite of the slightly reduced decentralisation that results), in general an operation that fails due to an insufficient vote count must be retried [3]. The probability of such failures will depend on a number of factors:

- The total time taken for a successful enumeration + commit, which controls the width of the window of vulnerability. This will depend on the time for Amoeba requests and replies, the time of a phase one commit (which must be carried out before deciding to propagate or go to phase two), and operating system overhead.
- The frequency of propagated operations.

The statistical behaviour of such broadcast packet channels is a well understood phenomenon [2]. (The ALOHA network is the classic example [1].) The critical parameter is the expected number of packet arrivals per packet transmission time. As this number exceeds one, the probability of collision increases rapidly, and along with it the expected time to send a packet. (In the case under discussion, a packet transmission time corresponds to the enumeration time, the the successful sending of a packet corresponds to the successful commit of an update to the replicated data.) The probability of collision is therefore sensitive to the number of server sites (and consequently the replication factor) in the system, since this number directly affects the time required for an enumeration to complete.

One possible way to control this factor would be to make use of multiple channels to reduce the likelyhood of collision, for example by partitioning the database, using one channel per partition. One can

assign partitions to distinct sets of servers, with little or no overlap (virtually different service organisations for each partition), or one can keep close to 100% overlap (i.e. every server instance has all partitions represented on it, but partitions are manipulated via different channels.) The channel mechanism can also be used to advantage by assigning different subsets of operations, between which maximum overlap is desired, to different channels.

3.7. Experimental implementation

The basic enumeration algorithm was implemented under the Unix operating system with kernel modifications to support the Amoeba transaction primitives. The implementation language was an experimental version of Prolog with built-in support for the Amoeba transaction primitives.

The main server process for a site forked child processes to play the part of its public listener and auxiliary enumeration sub-servers. This resulted in a rather enthusiastic consumption of resources, however, due to the combination of the lack of lightweight processes in Unix and the tendency of Prolog to consume memory resources.

The typical structure of Amoeba servers mapped surprisingly easily into the logic programming idiom. This use of Prolog differs from most research efforts on parallel Prolog systems in several ways.

(1) Process creation is only done explicitly (by execution of a special built-in predicate which does a *fork()* system call.)

(2) Communication (and implicit synchronisation) takes place between processes with no common ancestry (and, theoretically, between processes written in Prolog and other languages, such as 'C'.)

This experience suggests that further experiments with the use of Prolog as a prototyping and development tool in this unconventional medium would be worthwhile.

4. Conclusion

Although it cannot be recommended as a general reliable multicast algorithm, the enumeration procedure has some very useful properties:

- It is simple enough to implement quickly and correctly.
- It allows the set of active server instances involved in the support of a general network service to change spontaneously without requiring the explicit maintenance of a "multicast group".
- There is sufficient flexibility in the choice of various implementation details to allow extensions and modifications without complicating the basic structure to the point where it becomes unmanageable.
- Its properties depend on the basic semantics of the Amoeba primitives and do not depend on the presence of other services, beyond the obvious need to provide some means for clients to discover the permanent public port of the service.

On the negative side:

- Although all distributed database update procedures have some vulnerability to hardware failures, the serialisation implicit in this procedure perhaps leaves a larger window of vulnerability than it is possible to achieve.
- The technique would not be appropriate for services which must support a very high density of requests per unit time.
- The technique would likely not scale up well to large systems with high replication factors.

As long as its strengths and weaknesses are properly understood, this technique is a useful part of the Amoeba service designer's tool box.

5. Acknowledgements

The assistance of those who commented on this paper is appreciated. In particular Sape Mullender's important observations concerning crash recovery were most valuable.

References

1. Abramson, Norman, (1970). The ALOHA system- Another alternative for computer communications, *Proceedings of Fall Joint Computer Conference*, 37, 281-285.
2. Abramson, Norman, (Jan. 1977). The Throughput of Packet Broadcasting Channels, *IEEE Transcations on Communications*, COM-25.1, 117-128.
3. Barbara, D. and Garcia-Molina, H., (Aug 1986). The Vulnerability of Vote Assignments, *ACM Transactions on Computer Systems*, 4.3.
4. Daniels, Dean and Spector, Alfred Z., (1983). An Algorithm for Replicated Directories, *Proceedings of the 2nd PODC Conference*, ACM.
5. Gifford, David K., (Dec 1979). Weighted Voting for Replicated Data, pp. 150-162 in *Proceedings of the 7th Symposium on Operating Systems Principles*, ACM.
6. Mullender, Sape J. (1985). Principles of Distributed Operating System Design, *PhD Thesis*, Vrije Universiteit, Amsterdam.
7. Mullender, Sape J. and Tanenbaum, Andrew S., (1986.). The Design of a Capability-Based Distributed Operating System, *The Computer Journal*, 29.4.