Achieving Replication Consistency Using Cooperating Mobile Agents

Jiannong Cao[1], Alvin T.S. Chan[1], Jie Wu[2]

[1]Internet Computing and E-Commerce Lab
Department of Computing
Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong

[2]Dept of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431-6498
USA

Abstract

One of the problems in replication is how to coordinate the updates made to the copies maintained at different replicated servers so that data consistency is ensured. This paper presents a novel approach to designing protocols for accessing replicated data in a large-scale distributed environment such as the Internet. Unlike traditional message-passing based protocols which require expensive exchanges of messages among the replicated servers, the proposed approach uses cooperating mobile agents to synchronize the access to the replicated data at different servers. The design of such a mobile-agent enabled, fully-distributed protocol is presented and a prototypical implementation using IBM's Aglets is described. The performance of the protocol is also discussed.

1. Introduction

One of the most commonly used methods to enhance availability and reliability is to use replication, where the server providing the specified service is replicated and distributed across several sites on the network. These replicated data stores, called *replicas*, are distributed over the Internet and maintain copies of the replicated data, which can be accessed by the clients. Replication of data can improve availability because if a single replica fails, others still exist. It can also improve system performance by locating copies of the data near to their use or at a lightly loaded server.

One of the problems in using replication is to maintain data consistency between replicas, i.e., to guarantee that multiple copies of the same data behave like a single copy. Several algorithms, hereafter referred to as *data replication protocols,* have been proposed to synchronize the access to the replicated data. These protocols aim at providing an abstraction of a single copy of the data. However, different protocols may provide different consistency guarantees to applications. Some protocols provide strict single-copy semantics – any application using such a protocol is guaranteed to observe all changes to the data in the same time order as any other application using the data. They are referred to as *consistent protocols* [2, 11, 5]. Other replication protocols try to obtain better performance by using weaker consistency semantics [6], which allow replicated data objects to be temporally inconsistent but place either strict or probabilistic bounds on the divergence among replicas.

Existing data replication protocols are based on the message-passing communication model, although various communication services such as unicast message sending and receiving, multicast protocols, and remote procedure call can be used. Using message passing, conventional replication protocols are expensive because multiple local processes need to participate in sessions of passing messages and waiting for replies. Many replicated operations require several rounds of message exchange, and a replica may need to maintain appropriate context to determine its response [1]. Also, existing replication protocols are mostly designed for closely coupled distributed systems. A protocol that performs well in a local-area network may not scale to the world-wide Internet environment. This is because these protocols may incur large overhead, in both the network traffic and message-passing delay, in a wide-area network environment and they do not have provision of flexibility for adapting to system changes.

In this paper, we propose a novel approach, called *MARP* (Mobile Agent Enabled Replication Protocols), which uses cooperating mobile agents as an aid for designing replication control protocols. Mobile agents are programs that can autonomously halt execution from a host, travel across the network, and continue execution at another host [8, 9]. Cooperating mobile agents are a collection of mobile agents which come together for the purpose of exchanging information or in order to engage in cooperative task-oriented behaviors [3]. In the proposed MARP approach, mobile agents that carry requests from clients at different servers cooperate to maintain data consistency between replicas. In particular, we describe a fully distributed, consistent replication protocol using the MARP approach. The protocol is based on the well-known *Majority Consensus Voting (MCV)* scheme [11, 5].

1530-2016/01 $10.00 © 2001 IEEE

Comparing with message passing based protocols, the proposed mobile agent-enabled protocol has several advantages. First, mobile agent technology provides an approach to overcome the difficulties that hamper tight interaction between the processes. It has been found that, taking the advantages of being in the same site as the peer process and autonomously making decisions, mobile agent is especially suitable for structuring and coordinating wide-area distributed applications that require intensive remote computation and remote real-time interaction [12]. Second, using mobile agent leads to the reduction of the total amount of communications and allows us to design algorithms that make use of the most up to date system state information for decision making. This is because mobile agent can package a conversation and dispatches itself to a destination host where the interactions can take place locally. Furthermore, with mobile agent, a flexible and adaptive replication control scheme could be developed according to the current system state. Consequently such an approach achieves better performance because its intrinsic properties of automatically tolerating transit faults and dynamic changes of the network. Finally, mobile agents can support mobile computing by carrying out tasks for a mobile user temporarily disconnected from the network. After being dispatched, the mobile agents become independent of the creating process and can operate asynchronously and autonomously.

The rest of this paper is organized as follows. In Section Two, we describe the preliminaries including the terminology, system model, and assumptions. Section Three presents the design of the data replication protocol using mobile agents. Section Four describes a prototypic implementation of the proposed protocol using IBM's Aglets, and discusses the performance of the protocol. Finally, Section Five concludes the paper.

2. System Model and the MACR Framework

Copies of the replicated data are held at a number of replicas, which is part of a replicated server and consists of some storage and a process that maintains the data copy. Replicas receive *read* and *write* requests from clients for reading or updating the data. These operations are coordinated among the replicas using a data replication protocol that provides the client with the illusion of a single data object. Replicas also perform operations such as failure recovery, creation of new replicas and background information transfer.

We consider distributed computing in a wide-area network environment such as the Internet with the support of mobile agents. Mobile agents have their own identity and behavior and are capable of navigating through the underlying network, performing various tasks at the sites they visit, and communicating with other mobile agents. In the context of this paper, replicas are abstracted into a finite set of N mobile agent-enabled, cooperating processes. The processes can use both message passing and mobile agents in their computation. Mobile agents are employed for carrying out replication consistency coordination of the computations. Figure 1 illustrates the system architecture.

For each request received, a mobile agent is dispatched which is responsible of obtaining the permission to perform the requested operation. The mobile agent carries the request while it travels through the replicated servers. After obtaining the permission, it informs the replicated servers to process the request. We assume that mobile agents are capable of interacting with the stationary server processes for reading and writing specific data on the remote hosts they visited.

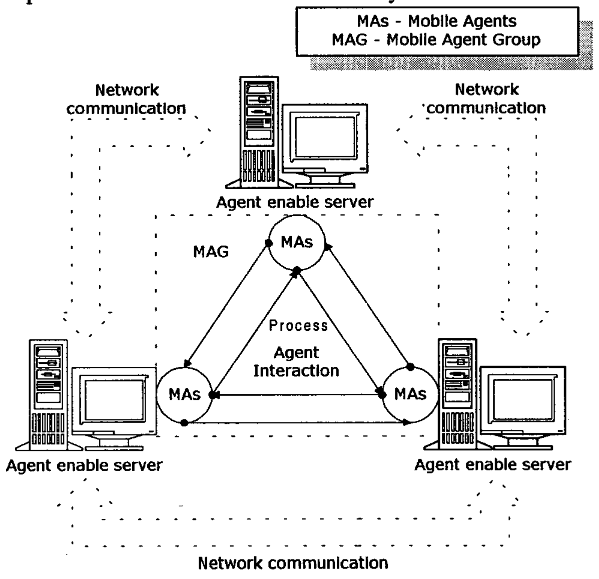

Figure 1: System architecture for mobile agent-enabled distributed computation

The network has asynchronous and reliable logical communication channels whose transmission delays are unpredictable but finite. A process can fail and recover. It fails according to the fail-stop model, that is, when a process fails, it immediately halts all processing without exhibiting malicious behavior. When a process fails, all other processes are informed of the failure in a finite time. If a mobile agent cannot migrate to a replicated server host after certain amount of time, the protocol assumes that the replica process at the host has temporarily failed. After certain number of such unsuccessful attempts, the protocol declares the replica unavailable and does not attempt to visit that replica until the next round of request.

The scale and characteristics of the Internet-based environment complicate protocol performance. For example, it has been reported that the Internet has long, variable communication latency, frequent short transient failures but rare long transient failures, etc.[6]. Data replication protocols designed to work in this environment should interact with replicas in a controlled fashion. The protocols should be sensitive to the communication latency of replicas, and should tend to communicate with nearby replicas rather than distant ones. The protocols should also address the problems associated with transient failures. Most existing protocols are not designed with such features.

3. A Mobile Agent Enabled Replication Data Algorithm

3.1 Overview

Depending on whether synchronization of operations at all replicas is performed before or after an update makes an access to data objects, consistent replication protocols can be divided into two classes [10]. Protocols that perform synchronization before accessing data objects are called *pessimistic* algorithms, while protocols that perform synchronization after accessing data objects are called *optimistic algorithms*. The Available Copy (AC) protocol [2], also known as the "write-all read-once" protocol, is an example of optimistic algorithms. Update operations must be applied at all available replicas. If all available replicas participated in the last update, an application can read from any replica and observe the update.

The AC protocol is vulnerable to communication partitions. However, it is useful in a system where access is read-dominated, which is the case in Internet-based environments, where most access requests are querying about information. An improvement of the AC protocol in terms of performance and handling realistic network failures is the voting protocols which are optimistic protocols [11, 5]. Each replica is assigned with one or more votes. Each read operation must collect a read quorum of at least r votes from replicas before it can proceed. Similarly, each write operation must collect a write quorum of at least w votes. To ensure consistency, $r+w$ must be greater than the total number of votes assigned to all replicas so that there is a *non-empty intersection* of copies between every pair of read and write operations. In this way, the user will have a consistent view of the current state of the system.

In this paper, we use mobile agents as an aid to design a fully distributed protocol for achieving consensus among members of a quorum. In our design, for simplicity, we consider the scheme that a quorum of replicas of an object is simply any majority of its copies [5]. A write operation is performed on a write quorum of replicas but a read operation may be executed on an arbitrary copy. This design consideration is based on the assumptions that read operations dominate the requests for accessing the replicated data and should be made fast. Also, it is acceptable that quires executed on a replica are not guaranteed to give an up-to-date answer.

With the MARP approach, the protocol is written from the point of view of the navigating mobile agents, rather than from the point of view of a stationary process communicating with other such processes [4]. The basic ideas of the proposed protocol is as follows:

- For updating a data item, a mobile agent is dispatched which travels across the network to obtain consensus from a majority of replicas. It is possible that multiple mobile agents from different replicated servers request to update a replica at the same time.
- At each server visited, the mobile agent requests a lock by appending its identifier at the end of the locking list maintained by the replica server. Permission of update is granted when the mobile agent is ranked the top in the locking lists of a majority of servers.
- The majority quorum achieved is used to determine both the most up-to-date version of a replica and the order of performing requested operations at a replicated server. Once a mobile agent obtains the lock, it checks the time of last update of all the quorum members and uses the most recent copy. It then broadcasts a message to all the replicas to request the update of the replica. Having collected acknowledgement from a majority number of servers, the mobile agent multicasts a COMMIT message to these servers and then releases the lock.

3.2 Data Structures

We have a network of N replicated servers. Each server is assigned a unique identifier, which is an integer ranging from 1 to N. Requests received from the client will be stored on each individual replica server S_i. After a pre-defined number of requests have been received or periodically, a mobile agent will be created and dispatched by S_i for processing the requests.

Each replicated server S_i maintains two data structures. One is called *Locking List (LL)*, used to store the locking information for each visiting mobile agent. LL is sorted according to the time the entries are created. The other is called *Updated List (UL)*, a list of identifiers of the mobile agents that have already obtained the lock and performed the actual update.

In addition, we assume that each server has a routing table containing the cost of the transferring a mobile agent form the local server to another server in the network. This information, together with others, can be used by a visiting mobile agent to determine the replicated server to visit next.

When a mobile agent is created, it is assigned a unique identifier consisting of the host-name of the replicated server where the mobile agent is created plus the local creation time. Each mobile agent maintains the following data structures:

- *Request List (RL)*: a list of requests carried by the mobile agent.
- *Un-visited Servers List (USL)*: a list of servers which have not been visited by this mobile agent. Initially, this list contains all the replicated servers in the system and is sorted by the cost of travelling from the current location. Recall that each replicated server is responsible of calculating and providing this information to the mobile agent.
- *Updated Agents List (UAL)*: a list of mobile agents that have already finished their request processing. This list is obtained by merging the UL maintained at each of the replicated servers.
- *Locking Table (LT)*: a table of locking information obtained from all visited severs. After a mobile agent arrives at a replicated server, it will get the LL of the server, and add it to its Locking Table.

3.3 The Replication Control Algorithm

For updating a data item, a mobile agent travels across the network to obtain consensus from a majority of replicas. Upon arrival at a replicated server, the mobile agent makes a request to the replica to obtain the lock. The replica creates a new entry for the mobile agent and appends it to the end of its LL. Permission of update is granted when the mobile agent's request appears on the top of LLs of a majority of servers. As shown later, in our algorithm, each mobile agent calculates the priority independently.

When a mobile agent finished its update, locks from this agent will be removed from all locking lists at the replicated server sites. Other mobile agents will then be able to change their priorities in their locking tables and decide which mobile agent will obtain the lock next. Mobile agents can exchange their locking information by leaving the information at the servers they visited. This information may be used by a mobile agent to determine which replicated server to visit next.

Algorithm 1 and Algorithm 2 show the operations performed by each mobile agent and each replicated server in the system, respectively. *N* is the number of all replicas in the system. The algorithm is defined in terms of performing a single request. However, it can be extended so that mobile agents can determine not only the first mobile agent who will obtain the lock next, but also the second agent, the third agent, etc.

The calculation of priority is done in a fully distributed manner by individual mobile agents. On visiting a replicated server, a mobile agent learns about which mobile agents have higher ranks than it does in the server's LL. It will carry the information with it when it travels from site to site, thus accumulating the locking information in its LT. After it accumulates enough information, the mobile agent knows which mobile agent has the highest priority to request the lock. If several mobile agents achieve an equal number of ranks and no further processing is possible (i.e., $S + (N-M*S) < N/2$, where *M* is the number of mobile agents that achieve equal ranks at *S* servers, and *N* is the total number of replicas), the tie is resolved by using the mobile agents' identifiers.

```
Algorithm 1: Operations performed by a mobile agent
Initialization:
Initialize all the data structures;
Found := False;
Top_Count := 0;

While Found <> True Do
  Begin
    Find the server to visit next from USL using routing
    information;
    On arrival at the server site
      Request lock and update data structures using server-
      provided information;
      If at top of LL Then Top_count ++;
      // Calculate priority using LT
      If Top_Count > N / 2 Then
          assign itself the highest priority;
      If M mobile agents achieve equal ranks at S servers
        and (S + (N-M*S)) < N/2 Then
          user mobile agent ID to resolve the tie.
      If Highest Priority Then
        broadcast UPDATE message to all replicated
        servers;
        wait until more than N/2 acknowledgements have
        been received;
        broadcast COMMIT message to all replicated
        servers;
        Found : = TRUE;
  End;
  dispose;
```

```
Algorithm 2: Operations performed by a replicated server
Initialization:
Initialize all the data structures;
Upon arrival of a mobile agent:
  If Request_for_Lock Then
    create an entry for the mobile agent and append it to
LL;
  // infromation sharing
  update data structures using information provided by the
  mobile agent;
Upon receipt of a UPDATE message:
  perform the requested update and update data structures;
  send an acknowledgement to the mobile agent;
Upon receipt of a COMMIT message:
  commit the requested update processing;
```

The main concern to be discussed in the remaining of this section is to show how the algorithms ensure that only one mobile agent is allowed to perform the actual updating at a time. This can be shown by proving the following two theorems.

Theorem 1: All participating mobile agents agree on who is ranked the top in the locking list at each individual replicated server.

Proof: This is straightforward. A mobile agent learns about the Locking List of a replicated server by either visiting the server site by itself or from other mobile agents which have visited the server. In both cases, the information about which mobile agent is on the top of the LL will be preserved.

Theorem 2: There is only one highest priority mobile agent in the system at any time.

Proof: This can be derived partially from Theorem 1. Since mobile agents have the same view on who is on top of the LL at a server, if a mobile agent MA_i achieves a top rank at a majority of servers, it is agreed by all mobile agents that MA_i will have the highest priority. In the case where a tie exists, every individual mobile agent will calculate the priority following the same rule, i.e. by using the mobile agent IDs, to resolve the tie. The rule guarantees that only one mobile agent will win the competition.

Theorem 3 establishes the lower and upper bounds on the number of migrations by the winning mobile agent.

Theorem 3: The winning mobile agent needs to migrate at least $(N+1)/2$ and at most N times in order to know the result.

Proof: The winning mobile agent wins the competition by either achieving the top rank at more than $N/2$ replicated servers or by having a higher rank in its ID than all other mobile agents that have achieved a equal number of top ranks in LLs of replicated servers. In either case, the winning mobile agent will know the winning result by visiting at least $((N+1)/2$ and at most N sites.

4. Prototypical Implementation and Evaluation

The mobile agent enabled replication management framework MARP and the specific algorithms proposed in this paper have been implemented on the IBM Aglet mobile agent platform [7] over a network of SUN workstations running Solaris. The Aglet is a Java-based mobile agent framework. The IBM Aglet platform also provides an aglet viewer called *Tahiti* and aglet servers, which are powerful machines that can host large number of aglets.

Experiments have been carried out to demonstrate the effectiveness of the proposed MARP framework and the algorithms. An interface has been developed to set up the experiments and to visualize the execution of checkpointing and rollback algorithms. An exponential random number generator was used to generate requests. In all experiments, for each server, 60-150 requests were generated at different rates. Experiments were set to measure the latency of update operations, and the percentages of requests whose locks are obtained by visiting a certain numbers of servers. The following metrics are used:

(a) *ALT*: the average time required by a mobile agent to obtain the lock, which equals the average number of server sites visited by a mobile agent times the average multiply by a mobile agent spent at a server;
(b) *ATT*: the average total time required by a mobile agent to process an update request. This total latency includes the message passing delay for sending the UPDATE and COMMIT messages.
(c) *PRK*: the percentage of requests whose lock is obtained by visiting K number of servers.

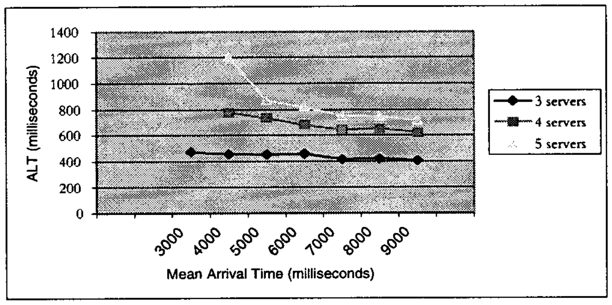

Figure 2: Average time for obtaining the lock by a mobile agent

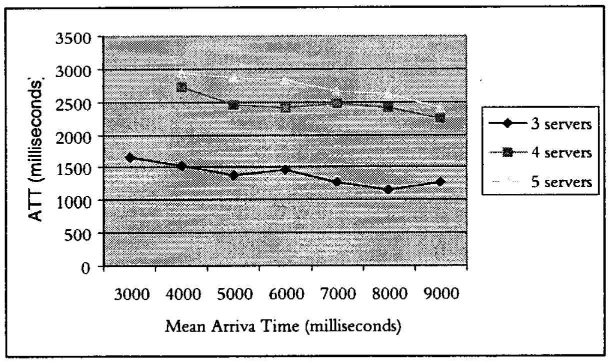

Figure 3: Average time for completing a request

Figures 2 and 3 show the results of ALT and ATT, respectively, obtained by using 3-5 replicated servers with different request generation rates. By comparing the figures, we can see that the message passing latency is the predominant factor determining the latency of operations on the replicated data. As the number of servers increase, this trend is more obvious. We believe that message passing would incur larger overhead if the experiments were conducted in a wide-area network such as the Internet. From the figures it can also be observed that as the mean arrival time increases both the ALT and ATT decrease.

Figure 4 shows the results of PRK with a configuration of 5 replicated servers. As it can be observed, for a higher request generation rate with inter-arrival time less than 45 milliseconds, for most requests, mobile agents need to visit all of the 5 servers in order to obtain the lock. However, as the generation rate drops, most requests can be granted the lock by having their mobile agents visit only 3 servers (*(N+1)/2*).

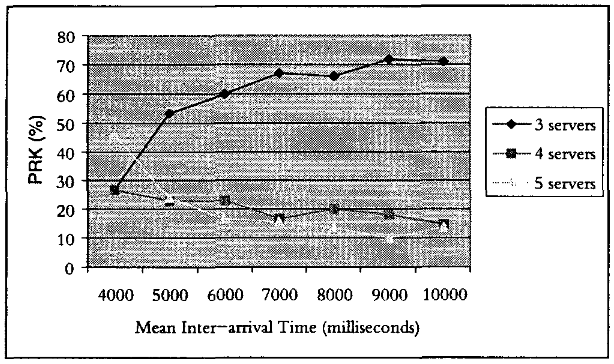

Figure 4: Percentages of request whose lock is obtained by visiting K(K = 3, 4, 5) servers

5. Conclusions

In this paper, we proposed the MARP approach to designing replication control protocols. We demostrate the feasbility of MARP by designing a consistent replication control protocol using mobile agents as an aid. Among others, the protocol has the following features:

- It is fully distributed and scalable.
- It avoids heavy message transmission required by conventional replication control protocols for achieving the quorum. A low message overhead leads to the improvement in the response time of an operation.
- It is order preserving: all updates are performed in exactly the same order at all the replicas.

We verified the correctness of the protocol and evaluated its effectiveness and performance. The results of the experiments showed that the proposed algorithms are effective with good performance.

The protocol described uses a strategy that yields good performance for an object that has a high read-to-update ratio, since a read operation needs only to access the local copy of the object. Updates must be performed at all copies but, since update is not frequent, the overhead for update is restrained. It is worth of notice that the MARP approach is a generic method, which can be used to implement different kinds of replication control algorithms. The mobile agents encapsulate the data replication protocols and a flexible and adaptive replication scheme could be developed according to the current state of the system.

Acknowledgements

This research is partially supported by Hong Kong Polytechnic Univ. under HK PolyU Research Grant G-T200. The authors wish to thank C.K. Cheng for his assistance in implementing the prototype of the proposed framework and algorithms.

References

[1] H.E. Bal, M.F. Kaashock and A.S. Tanenbaum, "Orca: A Language for Parallel Programming of Distributed Systems", *IEEE Trans. Software Engineering*, 18, 3 (Mar.), 1992. pp. 190-205.

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

[3] J. Cao, G.H. Chan, W. Jia, and T. Dillon, "Checkpointing and Rollback of Wide-Area Distributed Applications Using Mobile Agents", to appear in *Proc. IEEE 2001 International Parallel and Distributed Processing Symposium (IPDPS2001)*, April 2001, San Francisco, USA.

[4] M. Fukuda, L. F. Bic, M. B. Dillencourt, and F. Merchant, "MESSANGERS: Distributed Computing Using Mobile Autonomous Objects", *Information Sciences*, 1997.

[5] D.J. Gifford, "Weighted Voting for Replicatied Data", *Proc. 7th ACM Symp. On Operating Systems Principles*, Pacific Grove, California, Dec. 1979, pp. 150-162.

[6] R.A. Golding, "Accessing Replicated Data in Large-scale Distributed System", *MSc. Thesis, Dept of Computer and Information Sciences, Univ. of California, Santa Cruz.* June 1991.

[7] D. B. Lange and M. Oshima, "Programming and Deploying Java Mobile Agents with Aglets", Addison Wesley, 1998.

[8] D. B. Lange and M. Oshima, "Seven Good Reasons for Mobile Agents", *Communication of the ACM*, Vol. 42, No. 3, March 1999. pp. 88-89.

[9] V. A. Pham and A. Karmouch, "Mobile Software Agents: An Overview", *IEEE Communications Magazine*, July 1988. pp. 26-37.

[10] M. Singhal, "Update transport: A new Technique for Update Synchronization in Replicated Database Systems", *IEEE Transactions on Software Engineering*, Vol. 16, No. 12, Dec. 1990. pp.1325-1336.

[11] R.H. Thomas, "A Majority Consensus Approach to Concurrency Control", *ACM Transactions on Database Systems*, 4, 1979. Pp.180-209.

[12] C. Xu and D. Tao, "Building Distributed Applications with Aglet", http://www.cs.duke.edu/chong/aglet