Fault Tolerant Shared-Object Management System With Dynamic Replication Control Strategy

Juan Carlos Leonardo Takaichi Takaichi
Kyushu Institute of Technology
680-4 Kawazu, Iizuka, 820-8502 JAPAN
leonardo,takaichi@mickey.ai.kyutech.ac.jp

Shuji Narazaki
Nagasaki University
1-14 Bunkyo-machi Nagasaki 852-8521
narazaki@cs.cis.nagasaki-u.ac.jp

Abstract

This paper is based on a dynamic replication control strategy for minimizing communications costs. In dynamic environments where the access pattern to share resources can not be predicted statically, it is required to monitor such parameter during the whole lifetime of the system so as to adapt it to new requirements. The shared-object management system is implemented in a centralized manner in which a master processor deals with the serialization of invocations. On one hand, we attempt to provide fault tolerance as a way to adjust the system parameters to work only with a set of correct processors so as to enhance system functionality. On the other hand, we attempt to furnish availability by masking the failure of the master processor. A new master processor is elected that resumes the master processor processing. Our shared-object management system modularity is realized through a meta level implementation.

1. Introduction

A distributed system is a set of processors connected by communication links. Basically, a processor comprises a processing unit and memory, and communicate each other by exchanging messages. Apart from the message primitive, other mechanisms coexist such as RPC, the shared memory model and ORB (Object request broker) architecture that offer a high level of abstraction. Communication is a fundamental tool to coordinate the activities in a distributed algorithm–the composition of all local algorithms and to share resources-data, software or hardware.

One aspect that renders systems the characteristic of being distributed is shared-state, that is, a global state shared among processors as an effort to maintain system functionality in spite of processor failures and to keep the correctness of a distributed algorithm specification as well. Needless to say a lack of synchronization leads to erroneous computations and then an ill-behavior of a distributed system especially in an asynchronous distributed system where the message delivery time is variable due to the unpredictable message delay. Hereafter, we use the term distributed system meaning an asynchronous distributed system.

In general, systems that require a great deal of messages to be exchanged in order to keep shared-state degrades performance to the extent that distribution no longer yields the expected benefits. This is primarily due to the communication channel is an expensive resource. However, resource sharing, an inherent property of distributed systems, may lead to an increase in communication costs to keep correctness of data when data resources are copied in different processors and propagation of operations are required.

The major technique used nowadays to enhance system fault tolerance and reduce communication costs to some extent is replication. Replication allows local access to a resource and provides the system with an aggregate computing power, that is, the overall computation from all processors taken as a whole. For example, data queries can be performed in parallel. However, a data consistency protocol is required to prevent processors from using stale copies. Roughly speaking a process that modifies data in a particular processor, it is required to propagate the update as a necessary requirement to keep correctness of data. Actual gains in replication in terms of communication cost minimization depend on the number of replicas , the placement of those replicas and the nature of operations. For instance, a system with a high rate of read accesses is advised to have a large number of copies, on the contrary a system with a high rate of updates a small number of copies suffices to prevent an overuse of the communication channel since the update protocol incurs in an extra communication overhead to keep up to date the replicas. Besides, the placement of replicas plays an important role so as to obtain high performance in

0-7695-0571-6/00 $10.00 © 2000 IEEE
7th International Conference on Parallel and Distributed Systems: Workshops (ICPADS'00 Workshops)

distributed systems. It is worthless to place a replica in a processor that has very low rate access or crashes regularly. The dynamic replication control strategy aims at minimizing communication costs by monitoring the access pattern and change to a new configuration on the fly.

Furthermore, replication also offers better availability since even though one or some replicas are unavailable clients can still get a service from the up replicas. Availability can be thought of the accessibility of services and a system is highly available if it is operational in a time interval. It is seems that the larger the number of replicas the more available is the system. However, it can rise communication costs because of strict consistency maintenance. Since failures of processors can cause the partial or total lost of system functionality, a level of availability need to be supported. Processors holding important data should not fail so often or have backups that assumes the responsibility in case of a crash.

We propose a fault tolerant shared-object system that allows the failure of processors and enhances availability by masking the failure of the master processor to a new processor. Failed processors are excluded from the set of processors by a failure detection protocol. Additionally, our shared-object system is implemented in modular way to avoid the mixture of algorithmic code and allocation code. A meta level model is harnessed so as to achieve modularity.

2. Replication Techniques

Replication improves performance and availability to some extent. First, there is no need for expensive remote accesses and an aggregate computing power from all processors is achieved by task parallelism. Second, despite processor failures, system functionality may not be lost provided that a replica is hold in another processor. However, in practice, the benefits of replication are hardly realized since the correctness of data have to be maintained, that is, a change in one replica has to be visible to all replicas. Mutual consistency enforcement causes an increase in communication costs to the extent that replication no longer yields a performance enhancement since the consistency protocol requires a great deal of messages. Indeed, the number of replicas is an important factor to obtain benefits from replication.

Additionally, the placement of replicas play an important role due to the processor workload is variable during the whole execution of the system. From this, replication can be classified as static and dynamic. Static replication determines the replica placement during initialization time. Replicas last in the established processor during the whole execution. This tends to degrade performance in open and dynamic environments. A trivial example is the client-server architecture. It is provided with a single copy that is store in the server and clients access it remotely (See Figure 1.a). Indeed, read accesses are very expensive, carrying as a consequence high communication costs. In the all-servers model, read accesses are cheap but update accesses are very expensive in terms of the number of messages to be exchanged to achieve mutual consistency (See Figure 1.b). The virtual shared memory approach comprise all processor's memory into a global memory. Processors store data at a determined memory space that is known by a group of processors with which the data is to be shared. Processors requiring a read access, fetch the data by using a global address. Although the data storage is decentralized, the system still suffer from expensive read accesses (See Figure 1.c). The previous approaches rely on a strict coherence mechanism to preclude processors accessing an out-dated object, that is, a read operation returns the value written by the most recent write operation.

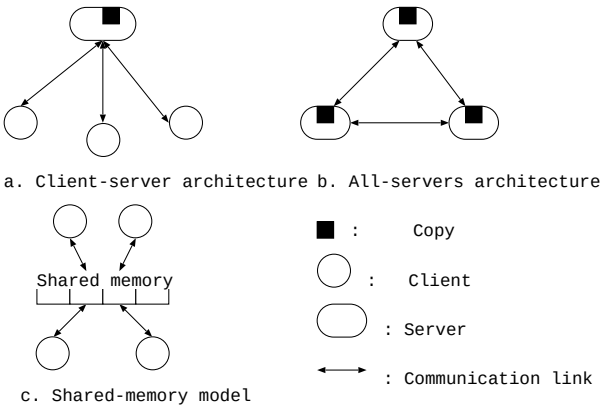


Figure 1. Static replication approaches

Another example is the Munin system [1] that comprises both the shared memory model and distributed memory model and aims at providing a type-specific memory coherence and loose consistency. Replication is achieved by means of a type-specific memory coherence. The system identifies the shared memory access patterns of shared data objects and determines the most appropriate coherence mechanism. However, the Munin system is required to know the anticipated access patterns of shared data objects at programming language level. The dynamic changes of workloads on shared data objects are left out in this approach.

Due to the steady changes in distributed environments, distributed applications that uses static replication could suffer from failing to provide optimum performance. Hence, access patterns needs to be assessed during the whole execution of an application and proceed to take a better specification that adjusts gradually to the current characteristics of the environment.

The previous approaches concern replication in terms of

0-7695-0571-6/00 $10.00 © 2000 IEEE
7th International Conference on Parallel and Distributed Systems: Workshops (ICPADS'00 Workshops)

the distribution of copies. However, failures need a careful attention in distributed systems. Processors exposed to failures caused by human errors, software bugs, power failures, etc needs careful design to prevent the system from losing functionality. For instance, consider a system in which a processor holding a replica crashes for some reasons and an update operation is performed. If the protocol is not design to deal with crash processors, it has to block the operation until the processor has recovered from the failure and resume the operation. The issue gets worse when a critical processors crashes. The system may become useless during a fraction of time. In order to cope with failures, a failure detection protocol is required to be operational when the system fails to contact a replica.

In the read-one write-all available approach, an operation is no longer required to ensure updates on all copies but only in the available copies. In this scheme, failed processors are not allowed to become available again until they recover by copying the current value of the shared-object. Otherwise, the recovering processor can allow stale reads and hence non one-copy-serializable executions. In case that a particular processor is down there will be no response and the coordinator will time-out. A validation protocol is realized to make sure that the coordinator has no make any mistake regarding the status of a failed process. In this approach, replicas are statically assigned and dynamic creation and removal of copies is not possible. The validation protocol ensures correctness but at the expense of increased communication costs [3].

In the primary-backup replication approach [2] [4] , a write operation is performed at the primary copy and propagated to the backups. A read operation is carried out only at the primary copy. When the primary fails, a backup, is elected either by a fixed line of succession or by an election protocol. It is required that the failure of the primary copy be detectable. Otherwise, it is possible to have two primary copies if a network partitions leaving the current primary copy in one side, and the newly elected one at the other side. This approach considers a degree of replication to implement a service. However, it does not support for a dynamic distribution of replicas.

Finally, dynamic replication deals with the movement of replicas among processors according to some parameters. Replicas number and placement changes during the whole execution of the system. Generally, the correctness criterion for replication is based on the access pattern ratio.

3. Dynamic Replication Control Strategy

The dynamic replication control strategy proposed in [5] attempts to minimize communication costs. This approach considers the remote access cost and the consistency maintenance cost as the principal factors that affects directly communication costs insofar as they need to be measured during the whole life-time of the system. Thus, the replication degree can be determined to adapt the system to a much better configuration, that is, a system that optimizes the number of replicas so as to obtain minimum communication costs and consequently to obtain high performance. In dynamic environments, dynamic replication can guarantee the expected benefits since replicas move from processor to processor depending on the access ratio. For instance, a replica that was highly accessed within an overwhelming fraction of time, could drop accesses drastically or after some time a processor with no replica increases the rate of read accesses. It is necessary to keep track of the access pattern and then select the most appropriate configuration of replicas. This dynamic replication control strategy aims at finding this configuration. It is realized in centralized manner similar to the primary-backup replication technique.

The types of processors are classified as : the master copy, the assistants (processors with a replica) and the clients (processors with no replica) as is illustrated in Figure 2.

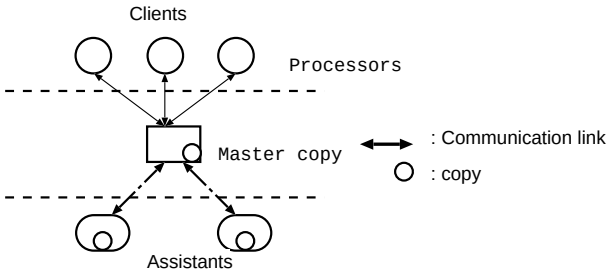


Figure 2. A centralized approach

The clients depend on the master copy in order to get an access to the data object–an access can be either a query or an update access to the data object. The assistants can query the data object locally. There is no need to contact the master copy before performing a query access. However, update access cannot be performed locally unless there is an authorization from the master copy. The last entity is the master copy. It has a special role that consists of receiving invocations from processors and send the response back. Furthermore, it plays a similar role as that of the primary copy that is to be responsible of keeping the correctness of replicas. In other words, it ensures mutual consistency by ordering update invocations. Each update invocation must be visible to all replicas (strict consistency). To be prevent inconsistencies, update accesses are controlled by means of a locking mechanism. Before performing an update access, an exclusive lock is required to proceed with the execution.

On this framework, the dynamic allocation scheme is performed by the master copy and communication costs are

0-7695-0571-6/00 $10.00 © 2000 IEEE
7th International Conference on Parallel and Distributed Systems: Workshops (ICPADS'00 Workshops)

related to the number of messages exchanged between the master copy and the rest of processors. Speaking roughly, Avoiding contacting the master copy as much as possible may help to decrease the communication overhead. All update requests are done via the master copy so a great deal messages might be required to perform the update access request to completion. On the contrary, since clients do not take part in the update propagation, they do not affect communication costs, but any query access on the data object implicates communication. The master copy must store read access information of all clients and the assistants stores its own read access information locally. At the time the replication algorithm is put into action the master copy first collects the monitored information from the assistants and then executes it.

The algorithm attempts to find the minimum number of messages to be exchanged between the master and other processors as shown in Figure 3.

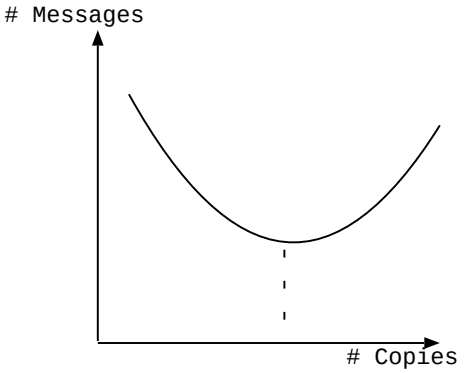


Figure 3. Minimizing communication costs

Before explaining the replication policy, the following notation is necessary:

n: Number of processors

$A : (a_1, a_2, ..., a_n)$ Number of accesses by each p_i

$S : (s_1, s_2, ..., s_n)$ The processor status vector (0: client, 1: assistant)

The replication policy is defined as follows:

$$N_c = 2(1 - S)A^t + (3SS^t + 1)$$

The first term computes the total number of messages between the master and the client. The second term computes the number of messages being used by an update operation. To minimize M, PS has to be selected.

$$N_c^* = \min\{2(1 - S)A^t + (3SS^t + 1)\}$$

A is ordered in descending order and presets PS to 0. Then the system sets each element of PS to 1 one after another until $M(k) > M(k-1)$ [5]

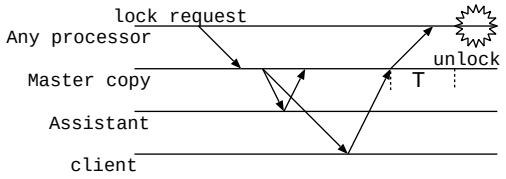
4. Dynamic Replication and Fault tolerance

The previous algorithm focuses on finding the number of replicas that make communication costs minimum. In our approach, we attempt to furnish fault tolerance, availability and at the same time to provide a dynamic replication control strategy. We propose an scheme in which the replication protocol considers only the correct processors to carry out the selection of the new set of replicas. This approach is similar to that of the primary-backup replication technique.

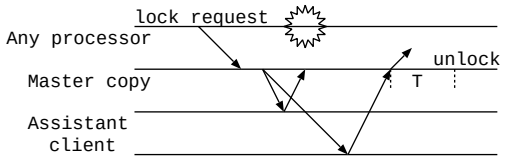
In our system, down processors are excluded from the set of processors to avoid the delay incurred by protocols when some processors fail. For example, the update protocol attempting to upgrade a failed copy. First, we consider the failure of clients and assistants and then later in this section the failure of the master copy which requires careful design. Recovering processors are added to the set of processors by contacting the master copy and are treated as clients. They are included in the set of processors so as to participate for the next configuration. The failure detection protocol, that is, a set rules that determined when a replica has failed, carries some extra communication overhead, however in return the system becomes more functional. The failure detection policy is based on a time-out constraint in which processors are declared as failed after a determined time-out period expires. In case that the system do not suffer failures of any kind, then communication costs would not be affected.

Processor failures can affect functionality due to the shared-object state is common to all processors. At the time of a failure, functionality degrades insofar as the system is not able to perform any computation. To provide the system with better functionality, it is necessary to tackle such failures such that the system can recover enough state so as to continue a computation. The query, update and lock invocations for accessing a shared-object need to be re-implemented so that they can execute to completion despite failures. The lock invocation used to serialize update invocations can cause serious degradations provided that a granted lock is at down processor. Since all processors remain locked until the down processor recovers, by detecting the failure of such processor, the master copy can annul the lock and transmit to the available copies an acknowledgement regarding the cancellation of the lock (See Figure 4). Furthermore, a processor with an execution time longer than the time-out period being set has to send a dummy acknowledgement to the master copy within the time-out period in order to keep the lock. After the time-out expires, the processor lost the lock and has to request it again for further processing.

0-7695-0571-6/00 $10.00 © 2000 IEEE
7th International Conference on Parallel and Distributed Systems: Workshops (ICPADS'00 Workshops)



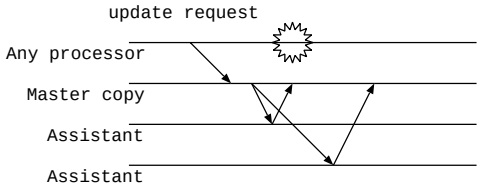
a) The processor fails after receiving reply



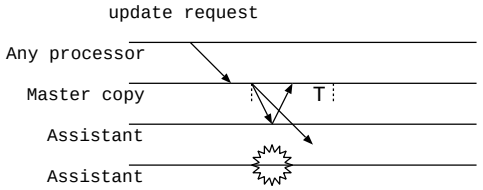
b) The processor fails after sending request

Figure 4. Lock request and fault tolerance

An update invocation handled by the master copy has to ensure that all correct replicas receive the shared-object current state. An extra-message is required as an acknowledgement stating that the replica has been updated. In the case that an assistant is down, the master copy waits for the reply within a time period and then declares the copy as failed (see Figure 5).



a) Requesting processor fails



b) A down assistant is excluded after time-out

Figure 5. Update request and fault tolerance

A read operation does not affect shared-state so in case that the reply is lost, the client is obliged to resend the previous read operation. The master copy is unaware of clients failures until an update on processor state itself is done by the replication protocol.

Recovering processors must talk to the master copy so as to be included into the set of processors. By default, a recovering process impersonate a client. After being accepted, they can resume the pending activities.

So far, failures of clients and assistants can be detected whenever an invocation that requires cooperation to perform a computation is executed. The failure of the master copy can be detected by time-out and retransmission. If a processor fails to contact the master copy after following the previous especificacions, it has to broadcast a master-copy-failure message so that a new processor can become a master so that the system can resume processing. The failure of the master copy generates more issues to cope with, for instance, replication consistency which is one of its goals that should be maintained through all failure ocurrences. Invocations that fail to execute simply because the master copy is down has to be redirected to a new master copy. In the previous work, after the delivery of an update invocation requested by any processor to the master copy, it assumes that the request will be perform to completion. Yet, the update operation execution is not guaranteed to be performed if the master copy fails right before the update request was sent. The calling processor is required to make sure the update request has been perform to completion. Otherwise, the system could end up having replicas with different states violating serializability.

Another case is when the master copy accepts an update request, but fails to send the update request to all replicas. If the new master copy is selected according to a predetermined order from the set of replicas and sees the last update on the replica, then inconsistencies arises in that the new master copy might assume that all replicas has also been updated properly, and immediately proceeds to offer services.

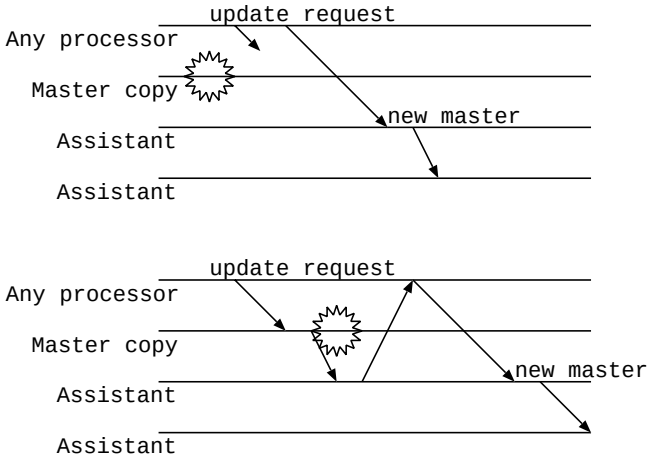


Figure 6. Update request and the primary copy failure

At the time the master copy is declared as failed, proces-

sors stop performing computation until the system recovers from that inconsistent state by selecting a new master. Cooperation might be necessary from all replicas to agree on the most recently state of the shared-object. After having agree on that, then the new master broadcast an acknowledgement to all processors so as to let them know the new system status.

The failure of the master copy risks availability since clients depend directly upon the master copy. In view that the system can suffer serious availability lost, a better design is required to enhance availability whenever the master copy is down. Obviously, the election of a new master copy may render the system with an enhanced availability, but it turns that the master copy apart from offering services to all processors, it runs the replication protocol. Additionally, the master copy keeps clients information required by the replication policy. It is required that each client keeps its own information as well so when the master copy fails, the new master copy can recover each client information.

5. Design Issues

The shared-object model comprises a communication handler, a replication handler and a user defined object (See Figure 7).

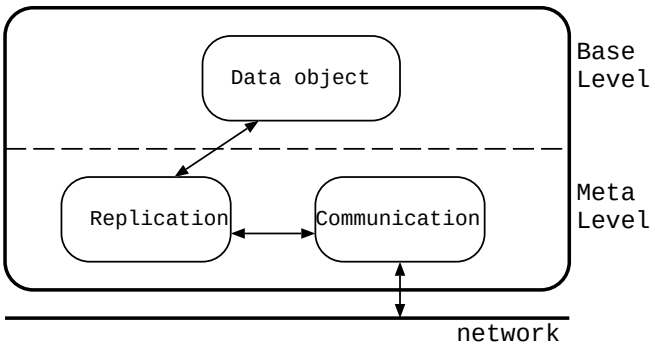


Figure 7. The shared-object model

The issues to be treated in this model are:

- Network transparency : enables programmers to access shared objects by using the same interface as any data object. Programmers do not need to have any knowledge regarding the location of shared-object as well.

- Replication transparency : users do not participate in the replication decisions. The system takes full responsibility for the replication protocol execution and the finding of the optimum solution so as to reduce communication costs.

- Modularity : maintenance and design of distributed applications is hard to realize in systems in which the algorithmic code and the allocation code is mixed together [6]. Modularity arises as a solution to make distributed applications easy to design and maintain. In our system, meta level architecture is harnessed to provide with the required modularity.

6. Conclusion

Dynamic replication plays an important role for providing a distributed system the tools to minimize communication costs. However, issues such as fault tolerance and availability need to be consider in its design to prevent the system from sacrificing functionality.

The shared-object management system is designed in such a way that down processors are excluded from the set of replicas by a failure detection protocol. A extra communication overhead rises as messages are required to detect down processor, however, the system obtains better functionality. A time-out constraint determines whether processors are in a failed state or not. The replication protocol uses only the set of correct processors to determine the new set of replicas and then proceeds to propagate the new states. Since a master copy handles serialization and executes the replication protocol by making use of access ratio information from all processors, the failure of the master copy causes availability loss. We attempt to enhance availability by masking the failure of the master copy through an election protocol.

References

[1] J. Bennett, J. Carter, and W. Zwaenepoel. Munin:distributed shared memory based on type-specific memory coherence. In *Proceedings of the Second ACM SIGPLAN Symp. on principles and practice of Parallel Programming(PPoPP)*, pages 168–176, Seattle, 1990.

[2] R. Guerraoui and A. Schiper. Fault tolerance by replication in distributed systems. Technical report, Ecole Polytechnique Federale de Lausanne, Switzerland, 1996.

[3] A. Helal, A. Heddaya, and B. Bhargava. *Replication Techniques in Distributed Systems*. Kluwer Academic Publishers, 1996.

[4] S. Mullender. *Distributed Systems*. Addison Wesley, second edition, 1993.

[5] S. Narazaki, N. Yoshida, H. Hamachi, T. Shimokawa, and K. Ushijima. Dynamic copy allocation scheme for distributed resource sharing based on meta-level computation. In *Proceedings of The 1998 International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 829–834, jul. 1998.

[6] F. Zambonelli. How to achieve modularity in distributed object allocation. In *ACM SIGPLAN Notices*, June 1997.

0-7695-0571-6/00 $10.00 © 2000 IEEE
7th International Conference on Parallel and Distributed Systems: Workshops (ICPADS'00 Workshops)