Architecture Based Approach to Adaptable Fault Tolerance in Distributed Object-Oriented Computing

Rodrigo Lanka, Kentaro Oda, and Takaichi Yoshida

Program of Creation Informatics, Kyushu Institute of Technology
lanka@mickey.ai.kyutech.ac.jp, oda@ci.kyutech.ac.jp,
takaichi@ai.kyutech.ac.jp

Abstract. To gain high level of performance in distributed object oriented computing, a required level of reliability in objects has to be maintained. This brings in a set of complex requirements into consideration. Furthermore depending on the unpredictability of the underlying environment, the replication should have architecture for the adaptable fault tolerance so that it can handle different situations of the underlying system before the system fails. We propose a mechanism for analyzing the complexity of this underlying environments and designing a dynamically reconfigurable architecture. The architecture provides the user required reliability by analyzing the performance and the reliability of the underlying environment and then either adjusting the replication degree or adaptively shifting to a suitable replication protocol. This architecture is a part of the Juice system which supports adaptation properties for a distributed environment.

1 Introduction

The system reliability is an important aspect for achieving high performance in distributed computing. Its importance is even further highlighted by the ever increasing usage of distributed systems in many aspects of today's life. However, *maintaining the required reliability in distributed systems* brings in the need for meeting a complex set of requirements. This complexity *depends on the underlying system's reliability* that includes reliabilities of operating system, hardware and network. The underlying system's reliability also fluctuates with *the unpredictable environmental changes*. Some examples of such changes are partial failures of operating systems, hardware failures and network breakdowns.

On the other hand, the required levels of system's reliability may also vary as they are decided by the users where *the cost of the system reliability mainly depends on the environment*, as depicted in Figure 1.

Our main objective is therefore to find both an approach for analyzing the complexities of the underlying environments and a dynamically reconfigurable architecture that supports the adaptive fault tolerance *to provide the required reliability according to changes in the underlying environment*. This would indeed be a significant step forward.

T. Enokido et al. (Eds.): EUC Workshops 2005, LNCS 3823, pp. 413–422, 2005.

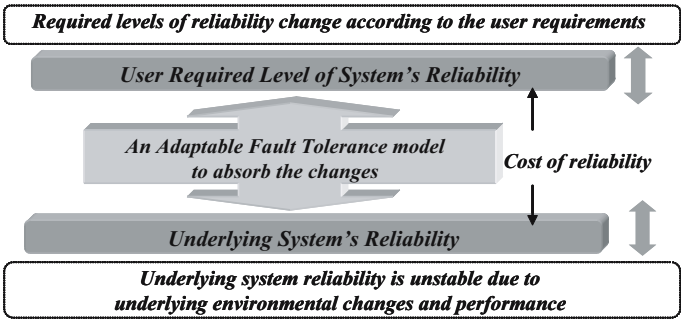

Fig. 1. The adaptable fault tolerance model which absorb the reliability changes

The mechanism that we focus on is *replication* as it is seen as a cost effective way to increase reliability of a system in the presence of potential failures. Moreover, in the distributed computing, replication is used for both performance and fault-tolerant purposes thereby introducing a constant trade-off between efficiency and consistency. However, our argument is that it is still difficult to presume a single replication protocol to get the required reliability throughout the lifetime of an object. Furthermore, maintaining the degree of replication at constant level does not ensure a constant level of reliability as the underlying environment continues to change. Therefore, to absorb the unpredictable changes in the environment, one must consider a suitable architecture for adaptable fault tolerance that is capable of handling different situations of the underlying system. This adaptability would enable the objects to change its respective behaviors that fulfills the current reliability requirements.

The Adaptable Fault Tolerance (AFT) model is based on two main strategies. On the one hand, the system can provide the required reliability by appropriately adjusting the replication degree. Secondly, the reliability of certain systems could be improved by adaptively shifting into a suitable replication protocol. The key factor in strategy selection by AFT model is the trade-off between the cost and the reliability. The AFT model is designed on the Juice object model that comprises adaptability as one of its main features. It allows objects to change its behavior on the fly by replicating some in a modular way [1][2]. The AFT model encapsulates the replication and communication which comprises a group membership service and reliable message delivery service to comply with the consistency of the replication protocol. The AFT model should therefore allow the system to reconfigure itself and maintain the required degree of system reliability. This would enable the system to provide the optimal reliable service even when the underlying environment continues to change.

Most research on reliability estimation assumes that individual component reliabilities are available, which means that they ignore the method of estimating those reliabilities. The reliability estimation models can however be used to project each component's failure rate. But this is not always possible due to the scarcity of failure data [3]. Therefore, our concern is to estimate the underly-

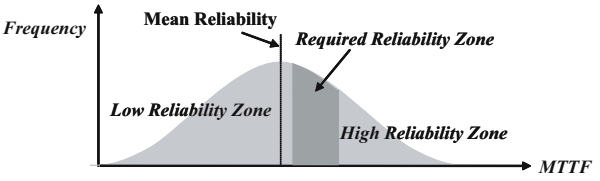

Fig. 2. Underlying System Reliability Zones

ing reliability by taking the entire underlying system as one unit because it has proven to be easy. However, we do not pursue this any further in this paper. For more information refer [3].

Assuming that underlying system reliability can be estimated, we can select strategies by locating it within one of the three reliability zones that are defined as high, required and low (Figure 2).

2 The Adaptation Strategies of the Algorithm

It is expected that distributed systems maintain their performance throughout its lifetime, even when it is frequently disturbed by the underlying environmental changes. However, it is impossible to provide a general purpose replication protocol that can be parameterized to accommodate all underlying environmental fluctuations. Therefore, it is important to design an adaptable algorithm to maintain the required reliability for the runtime environment.

The adaptation algorithm can be divided into two main strategies: (a) adjusting the replication degree and (b) changing the replication protocol. When the underlying system falls into the low reliability zone, both strategies can be applied to increase the reliability. But, when the underlying system is in high reliability zone, we need to reduce excess reliability that incurs unnecessary costs. Therefore we reduce the replication degree to remove unwanted reliability.

The reliability of a replicated object is significantly affected by both the number of replicas it has and their placement. Therefore, it appears from the outset that migrating the entire object onto another member who is more reliable than the current member could also be a feasible approach. (The objects are movable as they encapsulate their internal state). However, the object migration itself could increase the complexity of the system therefore we exclude this approach from the AFT model.

2.1 Adjust the Replication Degree

Depending upon the AFT policy, there are two aspects to why an object may change the replication degree.

On the one hand, an object will increase the replication degree when a member is admitted to the group to increase the reliability (i.e. when the reliability of the underlying system goes down; the object creates a new replica member and admits to the group to increase the reliability) or when one recovers from failure.

On the other hand, it will decrease the replication degree when an existing member leaves the group to reduce the reliability, concerned with the additional cost incurred for unwanted reliability or when an existing member fails.

2.2 Change the Replication Protocol

Replication Protocol Classification. Existing taxonomies of replication techniques take into account a broad spectrum of protocols, including those with both different levels of consistency and reliability. Therefore, the replication protocols can be classified into eight classes based on three parameters (levels) that determine their performance and properties [4]. They are *the method of synchronization, the type of protocol* and *the type of communication* (see Figure 3).

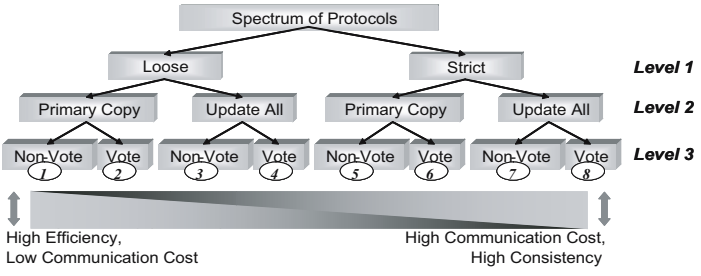

Fig. 3. Spectrum of replication protocols

Level 1 - The Synchronization Method (Loose or Strict). Synchronization depends on both the network traffic generated by the replication algorithm and the overall communication overheads. Therefore, we consider two types of synchronization according to the degree of communication. The *loose synchronization*, synchronizes the replicas by sending a single message containing a set of updates within a given period of time. This is therefore more attractive when dealing with efficiency. In contrast, the *strict synchronization* synchronizes each update individually by sending messages on a per update basis. It is therefore more attractive when dealing with consistency.

Level 2 - The Protocol Type (Primary Copy or Update-all). The *Primary copy* replication requires any update to the objects sent to the primary copy first where it is processed. The updates are then propagated to all other back-up sites. Therefore, this is more graceful when dealing with efficiency. In contrast, the *Update-all* replication allows updates to be performed anywhere in the system. That is, updates can concurrently arrive at two different copies of the same data item and it is more graceful when dealing with consistency.

Level 3 - The Communication Type (Non-voting or Voting). The method of acknowledging also can be divided into two types: non-voting and voting. Under *non-voting* communication, the object can decide whether to commit or abort a

message by itself. And, this is more graceful, when considering the efficiency. In contrast, *voting* communication requires an extra round of messages to coordinate the different replicas and it helps to provide high consistency.

Selection of Replication Protocol. The selection of replication protocols into the runtime environment should be done adaptively from a pool of available protocols depending on the condition of the underlying environment. For selecting the optimal protocol for a particular condition, it is required to estimate the costs of other available protocols and make comparisons. A principle aspect of such comparisons is the trade-off between consistency and cost.

When the AFT model needs shifting the current protocol into a higher efficient one, it shifts to the left from the current position of the spectrum (illustrated by table 1) and vise versa. Thus, one extreme of the above spectrum of replication protocols is represented by those that are highly reliable and efficient but not awfully consistent. And the other extreme is represented by those that guarantee the consistency but prohibitively expensive.

Table 1. The shifting methods of the spectrum (*to increase the efficiency*)

Current Protocol	How to Select the Next Protocol		
	Level 1	*Level 2*	*Level 3*
8, 6, 4, 2	-	-	Voting ⇒ Non-voting
7, 3	-	Update-all ⇒ Primary	Non-voting ⇒ Voting
5	Strict ⇒ Loose	Primary ⇒ Update-all	Non-voting ⇒ Voting

We assume that all members are connected to become a specified group of replicas and each member carrying a single vote. When the algorithm adopts the majority decision criterion to select the most optimal replication protocol, the members of the replica group take a vote, and if a majority agrees for switching into the new protocol, then they all switch together. The criterion of majority voting requires the agreement from more than one half of the available members. Thus, one member of the replica group, selected at random, gathers the votes and decides the next protocol on behalf of the others.

3 The Adaptation Algorithm

The AFT model proposes an adaptation algorithm for executing the switching between strategies, according to the dynamic environmental changes. As an example, suppose when systems reliability goes down, then the algorithm selects one strategy which suits system's reliability requirements. On the other hand, the selection of strategies could be done even according to user requirements (i.e. the user defines the cost and reliability level and the algorithm evaluates and selects the most optimal strategy). As an example, suppose when a user concerns higher level of reliability (or consistency) rather than the cost, or vise versa, then the algorithm can select one strategy which suits user's requirements. The proposed algorithm consists of following steps (Figure 4):

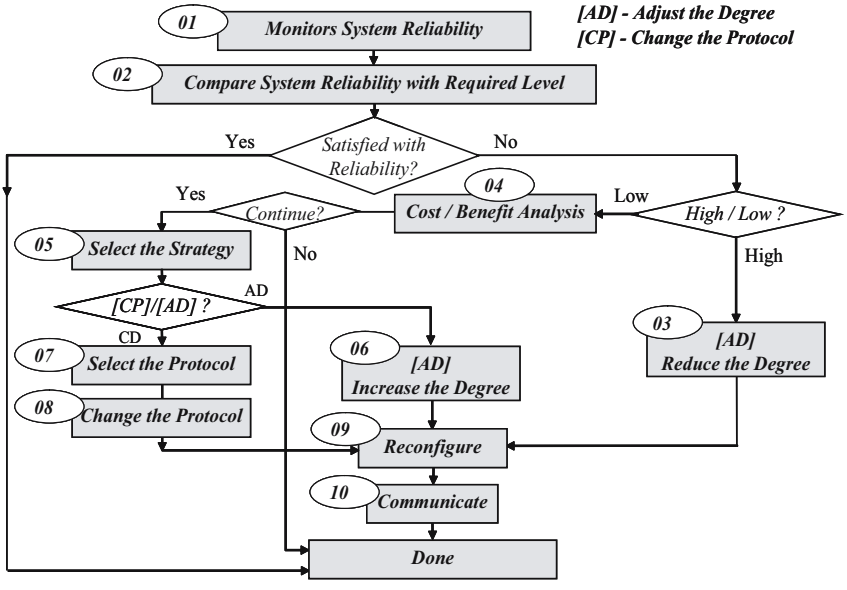

Fig. 4. The adaptable fault tolerance model which absorb the reliability changes

Step 1. Each monitor objects (CMI and USI) of the replica members evaluate and monitor the system reliability by using underlying system failure rates, network latency, local CPU load average, system resources and etc.

Step 2. Each member compares whether the current system reliability satisfies with either system required or user required reliability level.

Step 3. Reduce the current replication protocol degree in order to reduce the unwanted (excess) reliability.

Step 4. Carry out a cost-benefit analysis by considering the both user requirements and current system reliability with underlying system properties and its configurations to check whether it is worth to continuing. The impact of the cost of changing strategies and trade-offs need to be carefully studied with different reliability levels.

Step 5. Select the optimal strategy between (a) adjusting the degree (AD) and (b) changing the replication protocol (CP) based on their costs. Each member compares the reliability levels and the cost of targeted strategies.

Step 6. Increase the current replication protocol degree in order to meet the required level of reliability.

Step 7. According to environmental changes and user requirements, select the optimal protocol from the available pool following an agreement amongst the replica group members. For the protocol selection criteria, latency, bandwidth, CPU power, and etc. have to be taken into the consideration.

Step 8. Change the protocol when the optimal is selected. One of the members switches the current replication protocol into the optimal protocol.

Step 9. Reconfigure the old replication protocol and other configurations (if any) according to the current one.

Step 10. The members resume, pending and delivering messages according to the new communication protocol.

However, it is difficult to estimate the total cost of the system, because each member of the replica group is affected by its local environmental changes. Therefore, while each member estimates its local cost by itself, the decision on total cost is made by all. Furthermore, it is difficult to optimize each of the parameters: reliability, adaptability, scalability and consistency individually, which may result in conflicting and inconsistent solutions. For example, the latency might cause messages to arrive in different orders at different machines. Also, the different users getting different views of the underlying environment might result in consistency problems.

4 Design on the Juice System

4.1 The Adaptable Juice Object Model

The Juice system [1][2] is based on the adaptable object model. Adaptable Juice objects can reconfigure its internal objects with according to the new configurations to adapt with the changing execution environment at runtime. Therefore, this model can support adaptation properties for an open distributed environment. The adaptable object consists of five internal objects (Figure 5).

The Context Object: It is defined by the application programmer, and deals with the application domain implementation. It holds both the state and behavior of the adaptable object.

The Executor: It provides execution and concurrent control. It executes methods from the context object corresponding to the messages received from the communicator.

The Communicator: This object interprets the communication protocol. The communicator can be installed modularly to provide a new communication protocol and its semantics.

The Adaptation Strategy: It provides strategies for adaptation as environmental changes occur. These changes are informed in the form of events.

The Encapsulation Object: It hides the internal structure of the adaptable object. The encapsulation object type is the same as the user-defined type for the problem domain (also same as the type of the context object).

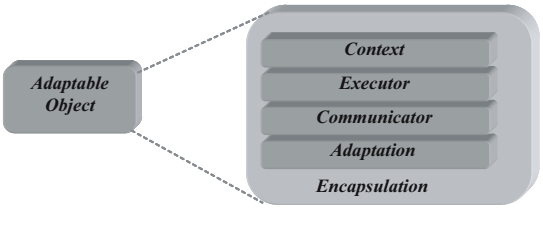

Fig. 5. The Adaptable Juice Object Model

4.2 The Adaptable Fault Tolerance (AFT) Model

The AFT model needs to provide adaptation for open distributed environments. As this model is designed on the Juice object, it can acquire adaptation as it inherits the properties of the Juice object. Therefore, the components of the AFT model have to be encapsulated within the Juice object to fulfill the requirements of the model. The internal structure of the model is illustrated in Figure 6.

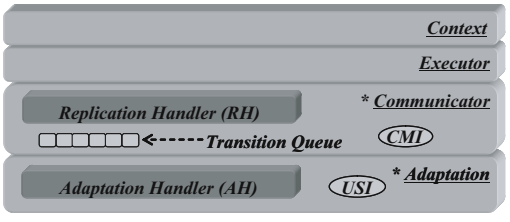

Fig. 6. The Internal Structure of the AFT Model

The model is based on component composition and addresses two levels of configuration: i.e. upper and lower levels. The upper level handles the reconfigurations of the Juice object model components (context, communicator, executor and adaptation) while the lower level reconfigures the objects that lie inside each Juice object model components (replication and adaptation handler).

One of the main advantages of this approach is that, because there are two separate configuration levels, it can help reconfiguring one level of components independently without any involvement of the other level. In other words, it is possible to reconfigure one level without losing the other configurations. As a result of these two reconfigurable structures, this approach helps to perform its switching strategies without leading to any inconsistencies.

In this model, the Underlying System Information (USI) evaluator seeks information from both the underlying virtual machine and operating system (i.e. the information such as available memory, link bandwidth, workload etc.). And, the Client Member Information (CMI) evaluator handles both the most recently connected client's information and the current replica group members' information (message failure rate, response time, latency etc.). In other words, both USI and CMI are responsible for obtaining the relevant information and sending to the Adaptation Handler (AH) which resides in the adaptation object of the Juice object model. After receiving the above required information, AH analyses the underlying system reliability and selects the best fit fault tolerance strategy according to the adaptation algorithm, which suites current environmental conditions. This strategy is therefore suitable to provide the required level of reliability in the system. Furthermore, the AH notifies the Replication Handler (RH) to replace themselves with the new object according to the new strategy selected by itself (AH). The RH maintains the replicated Juice object, relevant information about clients/group members and the transition queue.

The RH is in charge of both enforcing the required replication degree and maintaining information about all the clients associated with the replica group. When a strategy is selected for execution, the AH communicates to the RH. The RH then considers the network resources and the number of clients currently in the system, and decides the replication degree needs to be enforced.

4.3 Model of Communication

This research further intends to suggest an adaptable client-server group communication model, which can communicate under different environmental conditions with different replication protocols. According to the past researches, there has been a significant progress in the development of group communication infrastructures capable of offering a very impressive range of configuration facilities. For example, (*a*) BAST [5] allows different protocols to be selected and implemented for the same services under different usage patterns, (*b*) Horus system [6] allows communication stacks to be changed in runtime and (*c*) Coyote [7] allows the same message to be processed by different protocols in parallel.

Furthermore, during the transition period no one is responsible for handling messages as they change their configurations. In other words, all the message transactions might be lost during the protocol transition time as communicator of the Juice object reconfigures itself according to the new protocol configurations. Therefore, we designed a mechanism to overcome the problems when replication protocol transition period exists. As a solution all the relevant message transactions will be stored in a message queue called *transition queue* which resides with the communicator object of the Juice model, during the transition period. At this time, the communicator of the Juice model reconfigures itself according to the new strategy selected by AH. And, the transition queue will remain unchanged as it has different levels of configuration. Therefore, after the transition period, all the stored messages can be sent to both the relevant clients and servers with appropriate actions and newly configured values. However, sharing the transition queue between new and old replication handlers (RH) may corrupt the contents of the queue. Therefore, the old RH is not permitted to use the transition queue after it has transferred to the new RH.

5 Concluding Remarks

The AFT model with its structure facilitates the maintenance of the required degree of reliability in the system. This enables it to provide the optimal reliable service even when the underlying system changes prevail. This model is designed based on two main strategies: (*a*) adjusting the replication degree and (*b*) changing the replication protocol at runtime.

Furthermore, it enables *Juice objects* to reconfigure the current configurations at runtime. The strategies for fault tolerance can be adopted according to the requirements of each adaptable object. The adaptable objects can enhance the reliability of the system whenever changes in the environment turn the working

place to become hostile. It allows the system to reconfigure and execute adaptable fault tolerance algorithm under the current situations of the underlying system. Therefore, unlike common replication protocols, our solution is tightly integrated with the underlying environment.

It also accommodates a replication handler and a transition queue inside the communication object of the Juice system which can handle the problems when the replication protocol transition period exists. The adaptable communication model is also a part of the Juice system.

References

1. Leonardo, J.C., Oda, K., and Yoshida, T.: An Adaptable Replication Scheme for Reliable Distributed Object-Oriented Computing, 17^{th} International Conference on Advanced Information Networking and Applications, (2003)
2. Oda, K., Tazuneki, S., and Yoshida, T.: The Flying Object for an Open Distributed Environment, 15^{th} International Conference on Information Networking, (2001)
3. Popstojanova, K.G., and Trivedi, K.S.: Architecture Based Approach to Reliability Assessment of Software Systems, Performance Evaluation, Vol. 45/2-3, (June 2001)
4. Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., and Alonso. G.: Database replication techniques: a three parameter classification, 19^{th} IEEE Symposium on Reliable Distributed Systems (SRDS2000), Germany, (October 2000)
5. Garbinato, B., and Guerraoui, R.: Flexible Protocol Composition in Bast, 18^{th} International Conference on Distributed Computing Systems (ICDCS-18), (1998)
6. van Renesse, R., Birman, K., Friedman, R., Hayden, M., and Karr, D.: A Framework for Protocol Composition in Horus, Symposium on Principles of Distributed Computing, (1995)
7. Bhatti, N., Hiltunen, M., Schlichting, R., and Chiu, W.: Coyote: A system for constructing fine-grain configurable communication services, ACM Trans. on Computer Systems, 16(4):321–366, (1998)