5ZC-05

Fault-tolerant Distributed Systems for Multimedia Objects *

Motokazu Yokoyama, Katsuya Tanaka, and Makoto Takizawa †
Tokyo Denki University ‡
Email : {moto, katsu, taki}@takilab.k.dendai.ac.jp

1 Introduction

Distributed applications like teleconferences using high-speed networks are composed of multiple multimedia objects. The multimedia objects are required to be fault-tolerant in mission-critical applications. In addition, each multimedia object is required to support applications with some quality of service (QoS) even if the objects suffer from some fault or the system environment. In this paper, we discuss how each multimedia object supports QoS required by the applications in change of system environment and objects of the system.

In the traditional systems, checkpoints and replications are used to make the systems fault-tolerant. Larger and more complex multimedia objects are manipulated and transmitted in the multimedia applications than the traditional systems. Hence, it takes a longer time to save a state in the log and a large volume of log storage is required to take a checkpoint. It is also not easy to manipulate multiple replicas of multimedia objects since large volume of storage and computation overhead are required to store the replicas and to perform the requests on the replicas.

It is significant for the multimedia objects to support the applications with QoS required by the applications. The reliability and availability are considered to be kinds of QoS. The object fault is also considered to be change of some QoS parameters. At the checkpoint, a state obtained by reducing QoS of the multimedia object can be taken if the state satisfies QoS required by the application. By this method, we can reduce the volume of the log and the time to take the checkpoints. Suppose an object is faulty. The object can be rolled back to a state which is not the same as the previous one but supports QoS required by the application. The state of the multimedia object is large. In stead of storing the state of the object, methods performed on the object are logged. The object is rolled back by performing *compensating* method of the methods in the log. By this method, we can reduce the time for objects to be rolled back.

In distributed systems, multiple objects are required to take *consistent* checkpoints. In the multimedia applications, not only the state of the object but also the size of message is so large that the events for taking a checkpoint and sending and receiving the messages are not atomic. We discuss what is a consistent checkpoint to be taken in the multimedia objects.

*マルチメディアオブジェクト環境におけるフォールトトレラントな分散システム
†横山 基一, 田中 勝也, 滝沢 誠
‡東京電機大学

2 System Model

2.1 Object-oriented model

A system is composed of objects which are distributed in computers interconnected by high-speed networks. An object is an encapsulation of data and methods for manipulating the data. Applications can obtain service only through the methods supported by the objects. There are two types of objects, *classes* and *instances*. A class c is composed of *attributes* $A_1, \ldots, A_m$ ($m \geq 0$) and *methods* $op_1, \ldots, op_l$ ($l \geq 1$). The values of the attributes of the instance o are changed only through the methods. A collection $\langle v_1, \ldots, v_m \rangle$ of values of the attributes is a *states* of the instance o where each v_i is a value taken by the attribute A_i ($i = 1, \ldots, m$). An object has exactly one state at a time.

A new class c_2 can be derived from an existing class c_1. Here, c_2 *inherits* the attributes and the methods from c_1. c_2 can have additional attributes and methods. The attributes and methods inherited can be overridden in c_1. c_2 is a *subclass* of c_1, i.e. c_2 *is-a* c_1.

A class c can be composed of other classes $c_1, \ldots, c_n$, i.e. c_i is referred to as a *component* class of the class c. c_i is a *part-of* c. Let $c_i(s)$ denote a projection of a state s of the class c to a subclass c_i of c. Here, the *class* c includes an class c_i as an attribute.

On receipt of a request of a method *op*, *op* is performed on an object o. Let $op(s)$ and $[op(s)]$ denote a state and response obtained by performing a method *op* on a state s of an object o, respectively.

Let op_1 and op_2 be methods supported by an object o. $op_1 \circ op_2$ shows that a method op_2 is performed after op_1 competes. $op_1 \parallel op_2$ shows that op_1 and op_2 are concurrently performed on o.

2.2 QoS model

Applications obtain service supported by an object o through the methods of the object o. Each service is characterized by parameters like level of resolution and number of colors. These parameters are referred to as *quality of service* (*QoS*) supported by the object o.

The *scheme* of QoS is a tuple of QoS parameters named attributes $\langle a_1, \ldots, a_m \rangle$ ($m \geq 1$). Let dom(a_i) be a *domain* of an attribute a_i, i.e. a set of possible values to be taken by a_i ($i = 1, \ldots, m$). For example, dom(*resolution*) is a set of numbers each of which shows the number of pixels for each frame. Each state s of an object o supports a QoS *instance* denoted by $Q(s)$. $Q(s)$ of the scheme $\langle a_1, \ldots a_m \rangle$ is given in a tuple of values $\langle v_1, \ldots, v_m \rangle \in \mathrm{dom}(a_1) \times \ldots \times \mathrm{dom}(a_m)$. Let $a_i(Q(s))$ show a value v_i of an attribute a_i in $Q(s)$. Let S be a set of possible QoS instances. A QoS value v_1 *precedes* another v_2 ($v_1 \succeq v_2$) in dom(a_i) if v_1 shows better QoS than v_2. For example, $120 \times 100 \preceq 160 \times 120$ [pixels] for the attribute *resolution*. Let A be a subset $\langle b_1, \ldots, b_k \rangle$ of the QoS scheme $\langle a_1, \ldots, a_m \rangle$ where $b_j \in \{a_1, \ldots, a_m\}$

(j = 1, ..., k) and $k \leq m$. A QoS *instance* q_1 of a scheme A_1 *partially dominates* q_2 of A_2 iff $a(q_1) \succeq a(q_2)$ for every attribute a in $A_1 \cap A_2$. q_1 *dominates* q_2 ($q_1 \succeq q_2$) iff q_1 partially dominates q_2 and $A_1 \supseteq A_2$. A QoS instance q_1 is *minimal* in S iff there is no QoS instance q_2 in S such that $q_2 \preceq q_1$. q_1 is *minimum* iff $q_1 \preceq q_2$ for every q_2 in S. q_1 is *maximal* iff there is no q_2 in S such that $q_1 \preceq q_2$. q_1 is *maximum* iff $q_2 \preceq q_1$ for every q_2 in S.

An application requires an object o to support some QoS which is referred to as *requirement* QoS (*RoS*). Let r be an RoS instance. Here, suppose an object o supports a QoS instance q. If $q \succeq r$, the applications can obtain enough service from the object o. Here, q is referred to as *satisfy* r. Otherwise, q is *less qualified* than r.

3 QoS Based Relations among Methods

3.1 QoS equivalency

Suppose that a class c is composed of component classes c_1, ..., c_m ($m \geq 0$). An application specifies whether each subclass c_i is *mandatory* or *optional* for the class c. If c_i is mandatory, every object o of the class c is required to include an object o_i of c_i. If c_i is optional, o may not include any object of the class c_i.

[Definition] A state s_1 of a class c is referred to as *semantically equivalent* with a state s_2 of c ($s_1 \equiv s_2$) iff s_1 is the same as s_2 or $c_i(s_1) \equiv c_i(s_2)$ for every mandatory component class c_i of the class c. □

Let s_1 and s_2 be states of an object o. Here, suppose that the state s_2 is obtained by reducing the QoS of s_1, i.e. s_2 supports monaural type of sound while s_1 supports stereo type of sound. $Q(s_1) \succeq Q(s_2)$. Here, s_1 is referred to as *state-equivalent* with s_2 ($s_1 \approx s_2$) although $s_1 \not\equiv s_2$.

[Definition] A state s_1 of a class c is *state-equivalent* with another state s_2 of c ($s_1 \approx s_1$) iff $c_i(s_1)$ and $c_i(s_2)$ are obtained by less qualifying s, i.e. $Q(c_i(s_1)) \cap Q(c_i(s_2)) \preceq Q(s)$, for every mandatory class c_i of c and some state s of c. □

If $s_1 \approx s_2$, s_1 and s_2 show the same state but their QoSs are different, $Q(s_1) \neq Q(s_2)$.

Next, suppose each application requires to get service with requirement QoS (RoS) R from an object o. Suppose that there are two states s_1 and s_2 of an object o, which are state-equivalent ($s_1 \approx s_2$) and $Q(s_1) \preceq Q(s_2)$. If s_1 satisfies R, the application can use s_1. We define the equivalent relation on RoS as follows.

[Definition] A state s_t of a class c is *RoS-equivalent* with a state s_u of c on RoS R ($s_t \equiv_R s_u$) iff $Q(op_t(s)) \cap Q(op_u(s)) \succeq R$ and $op_t(s) \approx op_u(s)$ for every state s of the class c. □

[Definition] A state s_t of a class c is *semantically RoS-equivalent* with a state s_u of c on RoS R ($s_t \cong_R s_u$) iff $op_t(s) \equiv op_u(s)$ and $Q(op_t(s)) \cap Q(op_u(s)) \succeq R$ for every state s of the class c. □

A method op_t is *semantically RoS-equivalent* with op_u of a class c on RoS R ($op_t \cong_R op_u$) iff $op_t(s) \cong_R op_u(s)$ for every state s of the class c.

4 Compensation

Let op_t and op_u be methods supported by a class c and o be an object of c. A method op_u is a *compensating* method of another method op_t if $op_t \circ op_u(s) = s$ for every state s of an object o Let s_1 be a state obtained by performing op_t on a state s of the object o, i.e. $s_1 = op_t(s)$. Here, o can be rolled back to the initial state s from the state s_1 if the compensating method of op is performed on s_1. For example, *append* is a compensating method of *delete*.

[Definition] A method op_u *semantically compensates* another method op_t in a class c iff $op_t \circ op_u(s) \equiv s$ for every state s of c . □

RoS-compensating methods are defined as follows based on the RoS-equivalent relations.

[Definition] A method op_u *RoS-compensates* another method op_t on RoS R in a class c iff $op_t \circ op_u(s) \equiv_R s$ for every state s of c and RoS R. □

[Definition] A method op_u *semantically RoS-compensates* another method op_t on RoS R in a class c iff $op_t \circ op_u(s) \equiv_R s$ for every state s of c. □

5 QoS Based Consistent State

Multimedia Objects o_1, ..., o_n are distributed in the network, where each object o_i is created from a class c_i. A global state s is a tuple of local states $\langle s_1, \ldots, s_n \rangle$ where each s_i is a local state of an object o_i (i = 1, ..., n). The classes c_1, ..., c_n are structured in the *part-of* relation. A global scheme c is a collection $\{c_1, \ldots, c_n\}$ of the classes which are related in the *part-of* relation. A global instance o of the global scheme c is a collection $\{o_1, \ldots, o_n\}$ of the objects where each o_i is an object of the class c_i. According to the *part-of* hierarchy of the classes, the objects o_1, ..., o_n are also related in the *part-of* relation. We define a semantically equivalent relation among the global states.

[Definition] A pair of global states $s_1 = \langle s_{11}, \ldots, s_{1n} \rangle$ and $s_2 = \langle s_{21}, \ldots, s_{2n} \rangle$ are *semantically equivalent* ($s_1 \equiv s_2$) iff $s_{1t} \equiv s_{2t}$ for every mandatory class c_t. □

Even if a pair of states s_{1u} and s_{2u} for an optional class c_u are not semantically equivalent ($s_{1u} \not\equiv s_{2u}$), $s_1 \equiv s_2$ if $s_{1t} \equiv s_{2t}$ for every mandatory class c_t.

Next, we consider the QoS based equivalency of the global states s_1 and s_2.

[Definition] A pair of global states $s_1 = \langle s_{11}, \ldots, s_{1n} \rangle$ and $s_2 = \langle s_{21}, \ldots, s_{2n} \rangle$ are RoS-equivalent on RoS R ($s_1 \equiv_R s_2$) iff $s_{1t} \equiv_R s_{2t}$ for every mandatory class c_t. □

If some object o_t is faulty, all the objects are rolled back to a state equivalent with the current states. The system can take a state which is semantically or RoS equivalent state in the multimedia application. It is not easy to take a checkpoint where a state of a multimedia object is stored in a log due the large volume and complex structure of the object. Instead of taking a checkpoint, each object is compensated by performing compensating methods of the methods which have so far been performed on the object.

6 Concluding Remarks

This paper has discussed how to treat the QoS change of the object. We have defined semantically, RoS and semantically RoS equivalent relations among states of multimedia objects. By using the relation, we have defined the new types of conflicting and compensating methods.

References

[1] Kanezuka, T., Higaki, H., Takizawa, M., and Katsumoto, M., "QoS Oriented Flexible Distributed Systems for Multimedia Applications," *Proc. of the 13th Int'l Conf on Information Networking (ICOIN-13)*, 1999, 7C-4.