Autonomic Computing Systems on Distributed Multimedia Objects

Kenichi Watanabe, Tomoya Enokido, and Makoto Takizawa
Tokyo Denki University, Japan
E-mail {nabe, eno, taki}@takilab.k.dendai.ac.jp

Abstract

We discuss an autonomic computing system to manipulate multimedia objects distributed in multiple peer computers interconnected with high-speed networks. Multimedia objects are distributed, replicated, and versioned in nature since the objects are cached and downloaded to local computers in peer-to-peer (P2P) systems. Thus, various types of replicas of objects are distributed in various types and large number of peer computers. Even if servers which have objects are faulty or do not support enough quality of service (QoS) due to overloads and congestions, applications can obtain service of object by accessing to types of their replicas with different parts, QoS, and versions. In this paper, we discuss how to autonomically support applications with enough service on objects in presence of faults and QoS changes.

自律分散マルチメディアオブジェクトシステム

渡辺 健一　榎戸 智也　滝沢 誠
東京電機大学大学院理工学研究科情報システム工学専攻
E-mail {nabe, eno, taki}@takilab.k.dendai.ac.jp

本論文では、高速ネットワークで相互接続された複数のピアコンピュータ上に分散されたマルチメディアオブジェクトを操作するためのオートノミックコンピューティングシステムについて議論する。オブジェクトが P2P システム上のコンピュータにキャッシュ、またはダウンロードされることによって、レプリカが分散される。また、オブジェクトの更新により、バージョンが生成される。このようにして、さまざまな種類のレプリカが、ピアコンピュータ上に分散される。過負荷、輻輳、障害によるオブジェクトの QoS の変化に対して、アプリケーションはレプリカをアクセスすることによって、必要な QoS を取得することができる。本論文では、障害や QoS が変化するなかで、オブジェクトの提供するサービスをもとにアプリケーションの要求するサービスを自律的に提供する方法について議論する。

1 Introduction

Traditional information systems are realized in the client-server model. Application programs are performed on clients and application servers while issuing requests to servers like database servers [9] in 2-tier and 3-tier client server models, respectively. On receipt of requests from application programs on clients/application servers, requests like SQL [2] are performed on servers and then the responses are sent back to the application programs. Each computer plays one role of client, application server, and database server in the client-server model. According to the development of internetworking and computing technologies, various types and huge number of computers, possibly millions of personal computers (PCs) are now interconnected in networks. Here, each computer is *peer* named *servant*, i.e. each computer can play both roles of client and database server. This is a *peer-to-peer* (P2P) framework [11] which is now taking a central position in information systems.

Autonomic computing systems are developed by IBM [1] and Sun Microsystems [14] on the basis of new concepts, self-configuration, self-optimization, self-healing, and self-protecting to support fully available and reliable computation service in presence of component faults. Grid computing [4] is another example of P2P model, where computers, mainly personal computers in the Internet cooperate to perform a program to obtain as large computation power as a supercomputer.

A peer-to-peer (P2P) system is composed of large number and various types of peer computers interconnected in high-speed networks. Multimedia objects are distributed in multiple computers. Service supported by computers and networks is characterized by quality of service (QoS). Response time, throughput, reliability, and availability are also QoS parameters of computers. Delay time, bandwidth, and packet loss ratio are QoS parameters of networks. Change of a system is modeled to be change of QoS supported by the computers and networks. It is critical to support applications with enough quality of service (QoS) in change of computer and network services. In this paper, we discuss how to support applications with QoS required even if computers and networks do not support QoS required due to congestions, overloads, and faults.

In multimedia applications, a multimedia object is often replicated in nature on multiple computers since the object is downloaded and cached to local computers. In addition, only a part of a multimedia object may be stored in a computer. Furthermore, objects downloaded in computers may have QoS different from the original object. For example, a full-coloured video object stored in a video server is downloaded to personal computers. A monochromatic version of the video object is stored in a computer and only some scene of the video is stored in another computer since users are interested only in them and in order to reduce the storage size. Thus, an object is partially distributed to multiple computers, i.e. only part of the object, a version of the object, and object with different QoS. Here, even if a server is faulty, objects on the faulty server have been distributed in other computers as stated here. An application can access to a computer which has a part of the object which the application would like to manipulate. The application may obtain a part of the object from the partially distributed parts, which satisfies the requirement. We discuss how to obtain and manipulate multimedia objects which are distributed and replicated in networks.

In section 2, we discuss a system model. In section 3, we present replications of objects. In section 4, we discuss how to manipulate replicas distributed in peer computers.

2 System Model

An *object* is an instantiation of a *class*. A class is composed of attributes and methods for describing the class and manipulating its object. A new class can be derived from existing classes whose attributes and methods are inherited to the derived class. The derived class is referred to as *subclass* of the classes. A subclass is in an is_a relation with the classes. In addition, a class is composed of component classes, i.e. domains of attributes are classes. A component class is in a $part\_of$ relation with a class.

A system is composed of multiple peer computers $p_1, \ldots, p_n$, which are interconnected in communication networks. Each computer p_i supports an object base (OB_i) which is a collection of persistent objects. An object is an encapsulation of data, i.e. values of attributes and methods for manipulating the data. Objects are mainly multimedia objects. Each object is characterized by types of service, i.e. a collection of methods and quality of service (QoS) like frame rate and number of colours. QoS of an object obtained by an application depends on what QoS is supported not only by the object itself but also networks among the application and the object. For example, even if an object supports high bandwidth in a computer, applications in another computer cannot take enough bandwidth if the network is slower than the object.

QoS supported by an object is changed in a computer. For example, QoS is degraded due to some fault like performance fault of the computer. If a computer p_i is faulty, the object base OB_i in the computer p_i is also faulty. Availability and reliability are kinds of QoS. If an object is in a mobile computer, QoS of the object is changed depending on QoS of the communication connection, e.g. wireless channel according to the movement. Thus, the change of an object including movement of the computer can be modeled to be change of QoS supported by the object. Response time, throughput, bandwidth, reliability, and availability are QoS parameters of an object.

QoS of an object is supported to applications by performing methods on a state of the object. Let s be a state of an object o. Let *op*(s) and [*op*(s)] denote a state of the object o and output obtained by performing a method *op* on a state s, respectively. Here, $Q(s)$ shows QoS of a state s itself. On the other hand, $Q([op(s)])$ indicates QoS of the output of the method *op*. Even if a state s of an object supports enough QoS, an application cannot obtain enough QoS from the state s if a method *op* is not facilitated to manipulate the object with its QoS. For example, a video object v is composed of fully coloured video data. However, the object v supports only a *display* method which can display monochromatic version of video object. An application can only watch monochromatic video from the coloured video object. Thus, QoS of an object depends on both state QoS and method QoS.

A computer can communicate with the other computers by exchanging messages in networks. A message is a unit of data transmission among computers. A message is decomposed into a sequence of packets and then the packets are transmitted in a network. The service supported to a computer by the underlying network is characterized by quality of service (QoS), delay time, bandwidth, and packet loss ratio. QoS of the network is changed due to faults and congestions.

An application program is initiated and then performed on a peer computer. The application program issues a method request to an object, which exists locally on the computer or remotely on another computer. On receipt of the method request, the method is performed on the object and the response is sent back to the application program. The method being performed may issue methods to other objects. The remote procedure call (RPC) is an example of this computation on which applications in the client-server model are based. A *transaction* is an atomic unit of work and is a program which is being performed by manipulating objects. A transaction is defined to be an atomic sequence of methods issued to objects which satisfies ACID (atomicity, consistency, isolation, durability) [3, 5]. Multiple transactions are required to be *serializable* in order to keep objects mutually consistent.

In another way, an application program only locally manipulates objects by moving to remote computers where the objects are stored. Then, the program moves to another computer. The program which is moving around computers and locally manipulates objects is referred to as *mobile agent* [13]. A *transactional* agent [12] is a mobile agent which manipulates objects in computers so as to satisfy commitment constraints like atomicity and majority ones.

3 Replication of Objects

3.1 Replications

An object o can be replicated to be a collection $\{o_1, \ldots, o_n\}$ of replicas. The replicas are distributed to object bases in multiple peer computers. For example, a user downloads an object o in a remote server computer into the local computer in object-oriented database systems [15]. Then, the user locally manipulates the objects downloaded in its local computer and eventually uploaded to the server computer. The optimistic concurrency control [7] is used to maintain the mutual consistency of objects and their replicas. An object downloaded is a replica of the object o. An object is also replicated in multiple computers in order to increase the reliability and availability of the object. There are many discussions of replicas is distributed relational database systems [10].

There are various discussions on how to decompose and replicate tables in distributed database systems [10]. A table is decomposed into segments by projection and restriction, *horizontal* and *vertical* decompositions of the table [10], respectively. On the other hand, an object is composed of not only data but also methods. Hence, we have to discuss how data and methods are replicated in multiple computers. In addition, it is critical to discuss how much QoS is supported by data and methods of the object.

In a way, one new version is created if an object is updated. Thus, an object is realized to be a sequence of versions.

3.2 State-based replication

There are ways to replicate an object o. A replica o_i of the object o is referred to as a *full replica* of the object o ($o_i \equiv o$) iff o_i is the same as the object o, i.e. o_i has same values of attributes and methods as the object o. If a replica o_i has only a part of the object o_i, the replica o_i is referred to as *partial* ($o_i \subseteq o$). For example, if only some attributes of an object is downloaded into a local computer, the object downloaded is a partial replica of the object. A replica with a subset of methods is also a partial replica. If a replica o_i has the same attributes as an object o, the replica o_i is referred to as *fully instantiated* ($o_i \equiv^I o$). Otherwise, the replica o_i is *partially instantiated* ($o_i \subseteq^I o$), i.e. o_i is composed of a subset of attribute values of the object o which is obtained by horizontal and vertical decompositions of the data of the object o. A replica o_i is *fully equipped* ($o_i \equiv^E o$) if the replica o_i has a same set of methods as the object o. Otherwise, the replica o_i is *partially equipped* ($o_i \subseteq^E o$).

In Figure 1, boxes indicate attributes and circles show methods. A replica o_1 is created from an object o. Here, o_1 is a full replica of o ($o_1 \equiv o$). Then, a replica o_2 is created. Here, since o_2 has a subset of attribute values of the object o while having the same methods as the object o, o_2 is partially instantiated while fully equipped, i.e. $o_2 \subseteq^I o$ and $o_2 \equiv^E o$. A replica o_3 is partially equipped while fully instantiated, i.e. $o_3 \subseteq^E o$ and $o_3 \equiv^I o$. A replica o_4 is partially instantiated and equipped, i.e. $o_4 \subseteq^I o$ and $o_4 \subseteq^E o$.

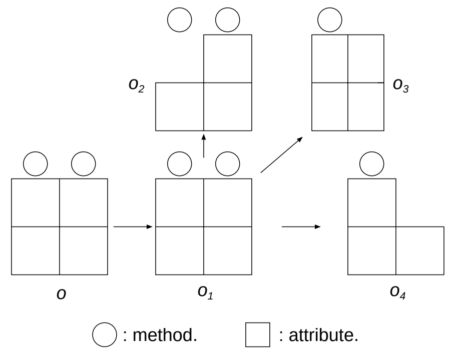

Figure 1. Instantiation and equipment of replicas.

3.3 QoS-based replication

Each object o is characterized by quality of service (QoS) like frame rate. If a replica o_i supports the same QoS as the object o, the replica o_i is referred to as *fully qualified* ($o_i \equiv^Q o$). If a replica o_i supports lower QoS than the original object o, the replica o_i is *less-qualified* ($o_i \subseteq^Q o$). If a replica o_i supports higher QoS than the object o, the replica o_i is *more-qualified* than the object o ($o_i \supseteq^Q o$). An application obtains QoS of an object by performing a method. For example, high-resolution image data can be displayed only through a high-resolution *display* method. Unless there is a high-resolution *display* method, an application can only view less-qualified image even if the object itself has high-resolution image data. A method t of a replica o_i is fully qualified if the method t of the replica o_i supports the same QoS as the object o. Otherwise, a method t of a replica o_i is less-qualified.

A replica o_i is a *full* replica of an object o ($o_i \equiv o$) if the replica o_i is fully instantiated, qualified, and equipped. Otherwise, the replica o_i is partial.

For a part of objects o_i and o, $o_i \equiv^{IEQ} o$ means that $o_i \equiv^I o$, $o_i \equiv^E o$, and $o_i \equiv^Q o$. $o_i \subseteq^{IEQ} o$ shows that $o_i \subseteq^I o$, $o_i \subseteq^E o$, and $o_i \subseteq^Q o$. $o_i \equiv o$ and $o_i \subseteq o$ indicate $o_i \equiv^{IEQ} o$ and $o_i \subseteq^{IEQ} o$, respectively. Notation like $o_i \equiv^{IE} o$ and $o_i \subseteq^{EQ} o$ are similarly used.

The relation $\equiv^\alpha$ is equivalent and $\subseteq^\alpha$ is transitive when $\alpha \in 2^{\{I,E,Q\}}$. There are following properties on the relations $\equiv^\alpha$ and $\subseteq^\alpha$ for every pair of objects o_1 and o_2:

1. $o_1 \equiv^\alpha o_2$ if $o_1 \equiv^\alpha o_3$ and $o_3 \equiv^\alpha o_2$ for some object o_3.
2. $o_1 \equiv^\alpha o_2$ if $o_2 \equiv^\alpha o_1$.
3. $o_1 \equiv^\alpha o_1$.
4. $o_1 \subseteq^\alpha o_2$ if $o_1 \subseteq^\alpha o_3$ and $o_3 \subseteq^\alpha o_2$ for some o_3.
5. $o_1 \subseteq^\alpha o_2$ if $o_1 \subseteq^\alpha o_3$ and $o_3 \equiv^\alpha o_2$ for some o_3.
6. $o_1 \subseteq^\alpha o_2$ if $o_1 \equiv^\alpha o_3$ and $o_3 \subseteq^\alpha o_2$ for some o_3.

3.4 Version-based replication

An object state is changed by update types of methods, i.e. attribute values are changed. A *version* is a snapshot, i.e. state of an object. Each time an object is updated, a new version of the object is created. Thus, an object is considered to be a sequence of versions which shows a history of the object. Replicas might be different versions of an object. Suppose a version o_i of an object o is obtained by updating a version o_j of the object o. Here, o_i is referred to as *directly follow* the version o_j ($o_j \vdash o_i$) (or o_j *directly succeeds* o_i) [Figure 2]. The version o_i *follows* o_j ($o_i \vdash^* o_j$) iff $o_i \vdash o_k$ and $o_k \vdash^* o_j$ for some object o_k.

Each version o_i of an object o has a starting time *st*(o_i) when o_i is created and an ending time *et*(o_i) when o_i is changed. The version o_i is valid from *st*(o_i) to *et*(o_i). Every version of an object o is identified by the identifier of the object o and version identifier. On the other hand, each replica is considered to be an object with the different identifier than the object o.

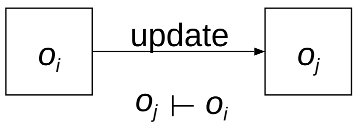

Figure 2. Version.

4 Acquaintances

We discuss how to manipulate multimedia objects whose replicas are distributed in peer-to-peer (P2P) networks. There are numerous and various peer computers in peer-to-peer (P2P) systems. Objects are replicated and distributed in peer computers with various ways as discussed in the preceding subsection. An application has to find computers which have objects, maybe replicas of objects which satisfy application's requirements. Since replicas of objects are dispersed to large number of peer computers in P2P systems, it is not easy to find the computers. Furthermore, it is not easy, maybe impossible to perceive what objects and replicas are stored on what computers in P2P systems.

Each computer p_i has its own *view* which is a subset of computers to which the computer p_i can access in a P2P network. An *acquaintance* of a computer p_i is another computer p_j which the computer p_i perceives to have what objects and can directly communicate with p_i. Here, $p_i \Rightarrow p_j$ (p_j is an acquaintance of p_i). Let *view*(p_i) be a set of p_i's acquaintance computers, $\{ p_j \mid p_i \Rightarrow p_j \}$. The acquaintance relation $\Rightarrow$ is neither symmetric nor transitive while reflexive. Even if the computer p_i thinks a computer p_j to be its acquaintance, the computer p_j may not think p_i to be an acquaintance. That is, $p_i \notin$ *view*(p_j) even if $p_j \in$ *view*(p_i). A pair of computers p_i and p_j are referred to as *friends* iff $p_i \Rightarrow p_j$ and $p_j \Rightarrow p_i$.

The acquaintance relation "$p_i \Rightarrow p_j$" is weighted by the trustworthy factor f ($p_i \overset{f}{\Rightarrow} p_j$). Suppose that $p_i \overset{f_1}{\Rightarrow} p_j$ and $p_i \overset{f_2}{\Rightarrow} p_k$. If $f_1 > f_2$, the process p_i considers that p_j is more trustworthy than p_k. Suppose a pair of the processes p_j and p_k have different knowledge about a replica o_i. Here, p_i takes usage of the knowledge of the object o owned by p_j.

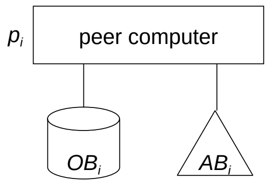

Figure 3. Computer.

Each computer p_i has an *object base* (OB_i) with *acquaintance base* (AB_i) [Figure 3]. The acquaintance base AB_i is composed of information on what objects are stored in what computers, which the computer p_i perceives. The acquaintance base AB_i is realized in a directed graph G_i named *object graph*. Each node o_i in the object graph G_i shows an object o_i. A directed edge from a node o_i to another node o_j shows the replication relations $\equiv$, $\equiv^I$, $\equiv^Q$, $\equiv^E$, $\subseteq^I$, $\subseteq^Q$, $\subseteq^E$, $\vdash$, and $\vdash^*$ among objects and replicas [Figure 4]. Here, $\alpha \in \{ I, Q, E \}$. For full replication relations $o_i \equiv o_j$, $o_i \equiv^I o_j$, $o_i \equiv^E o_j$, and $o_i \equiv^Q o_j$, there are directed straight edges $o_i \longrightarrow o_j$, $o_i \xrightarrow{I} o_j$, $o_i \xrightarrow{E} o_j$, and $o_i \xrightarrow{Q} o_j$ from o_i to o_j, respectively. For partial replication relations $o_j \subseteq o_i$, $o_j \subseteq^I o_i$, $o_j \subseteq^E o_i$, and $o_j \subseteq^Q o_i$, there are directed dotted edges

$o_i \dashrightarrow o_j$, $o_i \overset{I}{\dashrightarrow} o_j$, $o_i \overset{E}{\dashrightarrow} o_j$, and $o_i \overset{Q}{\dashrightarrow} o_j$ from o_i to o_j, respectively. For a version relation $o_i \vdash o_j$, there is a directed edge $o_i \longmapsto o_j$ from o_i to o_j.

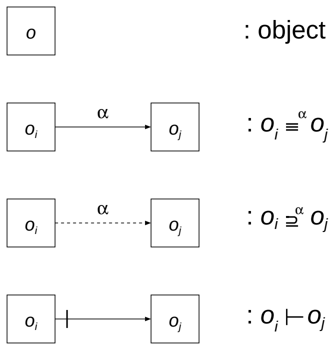

Figure 4. Object graph.

Figure 5 shows an object graph for Figure 1. For example, since o_1 is a full replica of an object o, $o \longrightarrow o_1$. $o_1 \xrightarrow{E} o_2$ and $o_1 \overset{I}{\dashrightarrow} o_2$ since $o_2 \equiv^E o_1$ and $o_2 \subseteq^I o_1$.

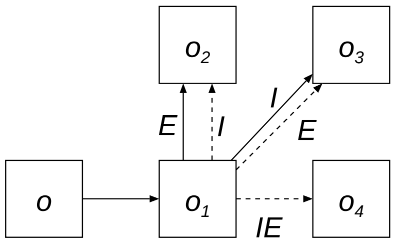

Figure 5. Object graph.

Each node o is associated with following information:

1. computers where the object o exists.
2. class, i.e. attributes and methods.
3. types, i.e. replica, version.

It is critical to discuss who can manipulate an object in what way. Let p_i be a peer computer and o be an object in a computer p_j. If the object o is in a computer p_i, we assume the computer p_i is allowed to manipulate the object o. Suppose the object o is in a different computer p_j from p_i. There is information on the object o in the acquaintance base AB_i in the computer p_i, i.e. p_j is an acquaintance of p_i ($p_i \Rightarrow p_j$). Here, if the computer p_i is granted an access right $\langle o, t\rangle$ for some method t, the computer p_i is allowed to directly manipulate the object o. Here, p_i is referred to as *qualified acquaintance* of p_j ($p_i \overset{\alpha}{\Rightarrow} p_j$ where α is an access right on an object in p_j). If the computer p_i is not qualified on the object o, the computer p_i is required to ask the acquaintance computer p_j to access to the object o.

Next, we discuss how to find a computer which has an object which we would like to manipulate. First, an application specifies properties showing what object the application would like to manipulate. By using the ontology [6], identifies of objects which satisfy the properties are found. Then, it is found what computers have the object by taking usage of the acquaintance. We take a local-to-global strategy to find computers:

1. A computer p_i broadcasts a request to all the acquaintance computers in the view.
2. On receipt of the request, an acquaintance computer sends a reply to the computer p_i, i.e. information on the objects.
3. If the objects are not found in the view, the computer p_i asks some number of acquaintances in the view to find the object, which are more trustworthy.

In another way, mobile agents are moving around in the P2P network.

5 Concluding Remarks

We discussed how to manipulate multimedia objects distributed in peer-to-peer (P2P) systems. Multimedia objects are replicated in various ways, fully/partially instantiated/qualified/equipped, in the P2P systems. We discussed what types of replicas of multimedia objects are distributed. Then, we discussed how to manipulate objects through acquaintance.

References

[1] Autonomic computing architecture : A blueprint for managing complex computing environments. 2002. http://www-3.ibm.com/autonomic/pdfs/ACwhitepaper1022.pdf.

[2] American National Standards Institute. The database language SQL. *Document ANSI X3.135*, 1986.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in Database S ystems. *Addison-Wesley*, 1987.

[4] I. Foster and C. Kesselman. *The grid: Blueprint for a new computing infrastructure.* Morgan Kaufmann Publishers, 1999.

[5] J. Gray. Notes on Database Operating Systems. *Lecture Notes in Computer Science*, (60):393–481, 1978.

[6] N. Guarino. Formal ontology and information systems. *Proc. of FOIS 98*, pages 3–15, 1998.

[7] H. T. Kung and J. T. Robinson. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems*, 6(2):213–226, 1981.

[8] N. A. Lynch, M. Merritt, A. Fekete, and N. Lynch. *Atomic Transactions (Morgan Kaufmann Series in Data Management Systems).* Morgan Kaufmann Publishers, 1993.

[9] Oracle8i Concepts Vol. 1. 1999. Release 8.1.5.

[10] M. T. Ozsu and P. Valdurie. *Principles of Distributed Database Systems*. Prentice Hall, 2nd edition, 1998.

[11] N. S. Ross, L.Grahman, and G. Carroni. *Proc. of the Third Internationlal Conference on Peer-to-Peer Computing*. 2003.

[12] S. Shiraishi, T. Enokido, and M. Takizawa. Transactional agent model for distributed object systems. *Proc. of the 14th International Conference on Database and Expert Systems Applications (DEXA 2003)*, pages 340–349, 2003.

[13] A. D. Stefano, L. L. Bello, and C. Santoro. A distributed heterogeneous database system based on mobile agents. *Proc. of the 7th Workshop on Enabling Technologies WETICE98 IEEE Computer Society*, pages 223–229, 1998.

[14] Sun Microsystems, Inc. N1 - intruducing just in time computing. White paper, Sun Microsystems, Inc., 2002.

[15] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. *Proc. of IEEE ICDCS-2000*, pages 264–274, 2000.