2009 IEEE International Conference on Networking, Architecture, and Storage

Fault-tolerant Online Backup Service: Formal Modeling and Reasoning

Hua Wang
birch_wh@163.com

Ke Zhou
k.zhou@hust.edu.cn

Ling Yuan
cherryyuanling@gmail.com

School of Computer Science and Technology
Huazhong University of Science & Technology
Wuhan, China

***Abstract*—Online backup service software provides automated, offsite, secure online data backup and recovery for remote computers. How to satisfy functional requirements and guarantee the fault tolerance of online backup service software is a difficult but crucial problem faced by software designers. In this paper, we investigate to incorporate the fault tolerant techniques in the system design, and propose a fault-tolerant online backup service model (FOBSM) to guide the development of online backup service system. The *FOBSM* comprises four components: *backup client (BC)*, *backup server (BS)*, *storage server (SS)*, and *online backup exception handler (OBEH)*. The first three components constitute three-party functional units, whereas *OBEH* serves as the centralized exception handling mechanism, which is devised to receive the external exceptions raised by the other entities, transform them into a global exception, and propagate it to the related entities to handle, so as to improve the fault tolerance of the software greatly. In order to provide precise and explicit idioms to system designers, we use Object-Z language to specify the *FOBSM*. Following the Object-Z reasoning rules, we reason about the fault tolerant properties of *FOBSM* and demonstrate that it can improve fault tolerance of the online backup service software effectively.**

***Keywords*-online backup service; fault tolerance; formal modeling; reasoning;**

I. Introduction

Data is the most valuable asset of an enterprise. Unfortunately, data is vulnerable in the face of natural disaster and malicious destruction. Backups are widely used to improve data reliability and provide a quick recovery from data losses. Currently, some large organizations and enterprises have established their own local backup systems. However, most of the backup systems cannot handle remote disaster recovery, and the maintainance cost is very high. Meanwhile, small and medium enterprises have to leave their data in an unprotected state for economic reasons.

With the popularity of Software-as-a-Service (SaaS), online backup service, a hot topic in the storage service area, provides an ideal solution for the above mentioned problems. The online backup service places the client data in the online storage space provided by the storage service provider via the WAN. It has many advantages, such as disaster recovery, economical deployment, easy-to-maintain, and etc, over traditional backup technologies.

As the soul of online backup service, the online backup service software determines the quality of backup service. Since the software is deployed within the WAN, the complexity of network application environment may affect the software negatively, which means that the software is prone to all kinds of exceptions. Besides, service-oriented feature demands the software to be provided with high fault tolerance to satisfy users' requirements. How to design effective fault tolerant mechanism for the online backup service software and prove the correctness of the fault tolerant properties formally is a challenging and valuable topic.

Fault tolerance means to avoid service failures in the presence of faults [1]. In recent years, many researches on fault tolerance have focused on exception handling, which is an effective way to realize fault tolerance of software systems [2]. Discussing exception handling in the software architectural level to guarantee fault tolerance has gained certain attention. R. de Lemos presents an approach for partitioning architectural elements such as components and connectors into normal and exceptional parts, which are responsible for delivering normal behaviors and handling failure behaviors respectively [2], [3]. A. F. Garcia proposes a generic software architecture for integrating exception handling with software, in order to support concurrent and sequential exception handling[4]. L. Yuan, etc., proposes a novel heterogeneous fault tolerant software architecture (GFTSA), which can guide the development of safety critical distributed systems [5], [6].

On the basis of research on fault tolerance and exception handling, We propose a practical fault-tolerant online backup service model, called FOBSM, which is incorporated with exception handling mechanisms to help guarantee the fault tolerance of online backup service software.

It is necessary to specify the model formally and reason about its fault tolerant properties [7], so as to demonstrate the accuracy of the proposed *FOBSM*. Z Notation is a formal specification technique based on predicate logic, which can describe the interface and operation of software system precisely [8]. Object-Z is an extension of Z Notation to facilitate object-oriented style specification [9]. Object-Z is

978-0-7695-3741-2/09 $25.00 © 2009 IEEE
DOI 10.1109/NAS.2009.76

a popular formal specification language, whose application has been discussed in many literatures [10], [11], it is an appropriate formal tool for our software, so we use Object-Z notation to formally model the proposed *FOBSM* and reason about its fault tolerant properties by following the related reasoning rules.

Because of the existing similarity on software structure and application environment among different storage service software, our accomplished research not only guides the development of online backup service software, but also has reference value for the design of similar softwares.

The remainder of this paper is organized as follows. Section 2 describes the online backup service software, including its structure and functions. Section 3 proposes the fault-tolerant online backup service model (FOBSM), and the formal model of the *FOBSM*. Section 4 reasons about the formal model of *FOBSM* to prove its fault tolerant properties. Section 5 gives the conclusions.

II. Online Backup Service Software

Our developed online backup service software is designed to be three-party structure: *backup client*, *backup server*, and *storage server*. The communication message flows among the three modules are shown as Figure 1.

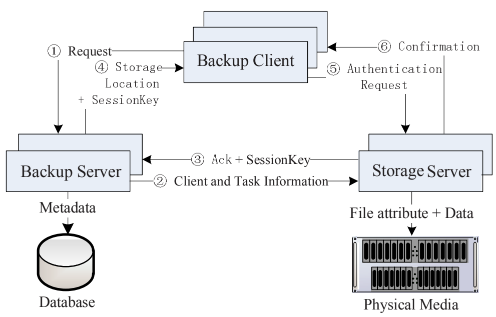

Figure 1. The structure of online backup service software.

A *backup client (BC)* is installed on a local computer requiring backup service. It can receive customized backup policies from the user to accomplish automated data backup and complete real-time backup and recovery according to users' manual operation. The server side includes two modules: *storage server (SS)* and *backup server (BS)*, both of which are installed in the computers of service center. The *SS* provides data storage services, communicating with thousands of *BCs* to process data backup and recovery, besides, *SS* carries out data management service, which include storage media's enrollment, space reclaimation and patch cleaning up. The *BS*, served as the supervisor of the whole online backup service software system, is responsible for user and storage management, task scheduling, state monitoring, and database maintainance. The online backup service software may involve multiple *BSs* or *SSs* to satisfy load balance and meet plentiful and complicated backup requirements from clients.

We can follow the number of each message flow to illustrate the communication protocol. The first arrow represents a *BC* submits a service request of backup or recovery, which named as a job, to the BS. The second arrow indicates that *BS* sends job and *BC* information to *SS*. The third arrow represents that *SS* is ready to execute the task and return a session key that will be used by *SS* and *BC* to *BS*. The fourth arrow indicates *BS* returns session key and *SS* information to *BC*. The fifth and sixth arrows represent the authentication request and confirmation process between *BC* and *SS* before performing a concrete job.

Online backup service software provides three granularity backup, which are full backup, incremental backup, and differential backup. Besides, two implementation modes are available: automated backup and manual backup. Since different backup clients may be installed with different operating systems, such as Windows and Linux, online backup service software must support file backup and recovery under different file systems. In terms of multi-user access, online backup service software not only supports concurrent access from huge amount of users, but also provides corresponding service quality according to the rights endowed to users.

The complex software requirement and application environment bring great challenges to attain high fault tolerance for the service software. Software design is the key stage during software development, it determines function and performance of the future software. How to design high fault tolerant software is a significant research subject.

III. Fault-tolerant Online Backup Service Model

A. Overall Description of Fault-tolerant Online Backup Service Model

In order to meet the system requirements and provide the mechanism to realize fault tolerance for the online backup service system, we propose a fault-tolerant online backup service model called *FOBSM*, presented in Figure 2, which incorporates the fault tolerant mechanism in the architecture level.

The *FOBSM* consists of three kinds of components: *NCOBE*, *COBE*, and *OBEH*. The *NCOBE* indicates *non-critical online backup entity*, which involves multiple distributed *BCs*. The *COBE* indicates *critical online backup entity*, which involves *BS* and *SS*. The reason why we categorize *BC* into non-critical entity, *BS* and *SS* into critical entities is that in the Internet environment, the exceptions raised from client-side components, which is *BC*, is not as critical to the system as the exceptions raised from the server-side components, which are *BS* and *SS*. The *OBEH* represents *online backup exception handler*.

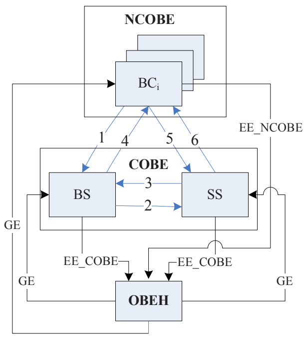

Figure 2. Fault-tolerant Online Backup Service Model.

The *OBEH* is responsible for dealing with the exceptions raised from other entities. According to the different properties of exceptions, we classify the exceptions into internal exceptions, external exceptions and global exceptions [3]. During the course of *NCOBE* originates a service request, if an exception occurs, the *NCOBE* will raise a corresponding internal exception, and use local exception handler to deal with the exception, such a process is transparent to the other components of the system [2]. If the *NCOBE* cannot deal with the internal exception successfully, it will propagate it as an external exception, called *EE_NCOBE*, to *OBEH*. When a *COBE* receives request from a *NCOBE*, it will process the request and return a response. During the process, once an exception is raised, the *COBE* will raise an external exception, called *EE_COBE*, and propagate it to *OBEH*. The external exception may due to an invalid request, which is called an interface exception, or due to an error in processing the request, which is called a failure exception [3]. During the communication between two *COBEs*, if exception occurs, the corresponding entity will raise an external exception, also called *EE_COBE*, and propagate it to the *OBEH*. Internal exceptions are only raised by *NCOBE* is because that the exceptions raised by *NCOBEs* may not affect the control flow of *COBE*.

Our proposed *FOBSM* incorporates two complementary exception handling strategies, which are internal exception handling and global exception handling. The internal exceptions raised by *NCOBEs* can be handled through local exception handling. Global exception handling mainly aims at two cases, one is that a *COBE* raises an external exception, the other is that a *NCOBE* cannot treat an internal exception successfully and propagate it as an external exception. It is the responsibility of *OBEH* to cope with external exceptions and transform them into a global exception.

B. Formal Model of FOBSM

In this section, we will give the formal specification of the proposed *FOBSM*. As mentioned in section 1, we adopt Object-Z as the formal language. The formal model involves four class schemas, which are *NCOBE*, *COBE*, *OBEH*, and *FOBSS*.

1) Global Type: Firstly, a set of definitions for global types are presented in the following, which will be used in all of the four class schemas.

$[NCOBE\_State]$ [set of states that *NCOBE* can be in]
$[COBE\_State]$ [set of states that *COBE* can be in]
$[NCOBE\_EH]$ [set of exception handlers used by *NCOBE*]
$[COBE\_EH]$ [set of exception handlers used by *COBE*]
$[GE]$ [set of global exceptions generated by *OBEH*]
$[EX]$ [set of specific exceptions the entities can be in]
$[MSG]$ [set of messages transmitted among entities]
$[B\_Job]$ [set of backup jobs]
$[R\_Job]$ [set of recovery jobs]

$RESULT ::= tolerate \mid stop$ [the result]
$SIG == \{0, 1\}$ [the signal]

$$
\begin{array}{|l}
NCOBE\_Normal : \mathbb{P}\, NCOBE\_State \\
NCOBE\_IE : \mathbb{P}\, NCOBE\_State \\
NCOBE\_EE : \mathbb{P}\, NCOBE\_State \\
NCOBE\_Fail : NCOBE\_State \\
COBE\_Normal : \mathbb{P}\, COBE\_State \\
COBE\_EE : \mathbb{P}\, COBE\_State \\
COBE\_Fail : COBE\_State \\
OBEH\_GE : \mathbb{P}\, GE \\
EXCEPTION : \mathbb{P}\, EX \\
\hline
NCOBE\_Normal \cap NCOBE\_IE = \varnothing\ \wedge \\
NCOBE\_Normal \cap NCOBE\_EE = \varnothing\ \wedge \\
NCOBE\_Normal \cap OBEH\_GE = \varnothing\ \wedge \\
NCOBE\_IE \cap NCOBE\_EE = \varnothing\ \wedge \\
NCOBE\_IE \cap OBEH\_GE = \varnothing\ \wedge \\
NCOBE\_EE \cap OBEH\_GE = \varnothing \\
NCOBE\_Fail \notin NCOBE\_Normal \cup NCOBE\_IE \cup \\
\quad NCOBE\_EE \cup OBEH\_GE \\
NCOBE\_State = NCOBE\_Normal \cup NCOBE\_IE \cup \\
\quad NCOBE\_EE \cup OBEH\_GE \cup \{NCOBE\_Fail\} \\
COBE\_Normal \cap COBE\_EE = \varnothing \wedge COBE\_Normal \cap \\
OBEH\_GE = \varnothing \wedge COBE\_EE \cap OBEH\_GE = \varnothing \\
COBE\_Fail \notin COBE\_Normal \cup COBE\_EE \cup OBEH\_GE \\
COBE\_State = COBE\_Normal \cup COBE\_EE \cup \\
\quad OBEH\_GE \cup \{COBE\_Fail\} \\
EXCEPTION = NCOBE\_EE \cup COBE\_EE \cup OBEH\_GE
\end{array}
$$

2) Non-Critical Online Backup Entity (NCOBE): The *NCOBE* class schema denotes how a *backup client (BC)* originates service request and acquires specific service, such as backup and recovery, by interacting with *backup server (BS)* and *storage server (SS)*.

The local variables: nor_state, $in\_exceptions$, $ex\_exceptions$, and $ge\_exceptions$ denotes that the *NCOBE* is in normal state, internal exception state, external exception state, and global exception state respectively. The variable $msgs$ specifies messages transferred among entities. The $except\_context$ function means that each exception corresponds to a exception handler.

NCOBE

$$
\begin{array}{l}
nor\_states : \mathbb{P}\, NCOBE\_Normal \\
in\_exceptions : \mathbb{P}\, NCOBE\_IE \quad ex\_exceptions : \mathbb{P}\, NCOBE\_EE \\
ge\_exceptions : \mathbb{P}\, OBEH\_GE \qquad msgs : \mathbb{P}\, MSG \\
except\_context : NCOBE\_IE \cup OBEH\_GE \nrightarrow NCOBE\_EH \\
except\_handle : NCOBE\_EH \rightarrow NCOBE\_State \\
nco\_backup : NCOBE\_Normal \rightarrow NCOBE\_State \times B\_Job \\
nco\_recover : NCOBE\_Normal \times R\_Job \rightarrow NCOBE\_State \\
\hline
nor\_states \cap in\_exceptions \cap \\
ex\_exceptions \cap ge\_exceptions = \varnothing \\
\operatorname{dom} except\_context \subseteq in\_exceptions \cup ge\_exceptions
\end{array}
$$

$$
\begin{array}{l}
cur\_state : NCOBE\_State \\
checkpoint : NCOBE\_Normal \\
ge\_rec\_sig : SIG \\
\hline
cur\_state \in nor\_states \cup in\_exceptions \cup ex\_exceptions \\
\cup ge\_exceptions \cup \{NCOBE\_Fail\} \\
checkpoint \in nor\_states
\end{array}
$$

INIT

$$
\begin{array}{l}
cur\_state \in nor\_states \\
checkpoint = cur\_state \\
ge\_rec\_sig = 0
\end{array}
$$

Re_Init

$$
\begin{array}{l}
\Delta(cur\_state, checkpoint) \\
\hline
cur\_state' \in nor\_states \\
checkpoint' = cur\_state'
\end{array}
$$

Send_request

$$
\begin{array}{l}
nco\_req! : \downarrow NCOBE \\
req\_co! : \downarrow COBE \\
co? : \downarrow COBE \\
\hline
cur\_state \in nor\_states \\
nco\_req! = self \wedge req\_co! = co?
\end{array}
$$

Rec_response

$$
\begin{array}{l}
co\_res? : \downarrow COBE \\
\hline
cur\_state \in nor\_states \\
co\_res? = self
\end{array}
$$

Trans_data

$$
\begin{array}{l}
\Delta(cur\_state) \\
backup\_job! : B\_Job \\
\hline
cur\_state \in nor\_states \\
(cur\_state', backup\_job!) = nco\_backup(cur\_state)
\end{array}
$$

Recover_request

$$
\begin{array}{l}
rreq\_job! : R\_Job \\
job? : R\_Job \\
\hline
cur\_state \in nor\_states \\
rreq\_job! = job?
\end{array}
$$

Recover_data

$$
\begin{array}{l}
\Delta(cur\_state) \\
reco\_job? : R\_Job \\
\hline
cur\_state \in nor\_states \\
cur\_state' = nco\_recover \\
(cur\_state, reco\_job?)
\end{array}
$$

IE_handle

$$
\begin{array}{l}
\Delta(cur\_state) \\
\hline
cur\_state \in in\_exceptions \\
except\_handle(except\_context(cur\_state)) \in nor\_states \Rightarrow \\
cur\_state' = except\_handle(except\_context(cur\_state)) \\
except\_handle(except\_context(cur\_state)) = \\
NCOBE\_Fail \Rightarrow cur\_state' \in ex\_exceptions
\end{array}
$$

EE_send

$$
\begin{array}{l}
ex\_exception! : \\
NCOBE\_EE \\
\hline
cur\_state \in \\
ex\_exceptions \\
ex\_exception! = \\
cur\_state
\end{array}
$$

GE_rec

$$
\begin{array}{l}
\Delta(cur\_state, ge\_rec\_sig) \\
ge\_exception? : OBEH\_GE \\
\hline
cur\_state' = ge\_exception? \\
ge\_rec\_sig' = 1
\end{array}
$$

GE_handle

$$
\begin{array}{l}
\Delta(cur\_state, ge\_rec\_sig) \\
\hline
cur\_state \in ge\_exceptions \wedge ge\_rec\_sig = 1 \\
except\_handle(except\_context(cur\_state)) \in nor\_states \Rightarrow \\
cur\_state' = except\_handle(except\_context(cur\_state)) \\
except\_handle(except\_context(cur\_state)) = \\
NCOBE\_Fail \Rightarrow cur\_state' = NCOBE\_Fail \wedge \\
ge\_rec\_sig' = 0
\end{array}
$$

The $except\_handle$ function is used to handle each exception and output the state of *NCOBE*. The nco_backup and $nco\_recover$ functions represent the state transition of *NCOBE* after the execution of a backup job and a recovery job respectively.

The state schema declares three variables: cur_state represents current state of the *NCOBE*, $checkpoint$ denotes the normal state of the entity before transition, ge_rec_sig is a boolean value, representing whether the entity receives the global exception transmitted from *OBEH*.

The *INIT* operation is used to initiate local variables: cur_state, $checkpoint$, and ge_rec_sig. When the *NCOBE* requires service from the *COBE*, it will startup *Send_request* operation. Once the request has been permitted, the *Rec_response* operation will be implemented to receive response from *COBE*. During the service process, if current service type is data backup, then the *NCOBE* should prepare the data to be backed up and perform the

Trans_data operation to complete the data transmission to *COBE*, otherwise, *COBE* will fetch the data from storage area and transmit it to client, the *NCOBE* should perform *Recover_request* and *Recover_data* operation to send request to the *COBE*, then receive data from *COBE*, and process the recovery. When local exception occurs in the *NCOBE*, *IE_handle* operation will be originated to process the internal exception, which is transparent to other components in the system. If the process fails, the internal exception will be transformed into an external exception, and sent to the *OBEH* through the *EE_send* operation. The *OBEH* receives all the external exceptions originated from both *NCOBE* and *COBE*, disposes them and obtains the global exception [12]. Subsequently all the related entities will be informed of the global exception and handle it. The *GE_rec* operation denotes that the *NCOBE* receives the global exception from the *OBEH*. The *GE_handle* operation describes how the entity handle the global exception, this process may cause the *cur_state* and *ge_rec_sig* variables to change.

3) Critical Online Backup Entity (COBE): The *COBE* class schema specifies how server receives service request, returns response, and provides data backup or recovery services.

COBE

$nor\_states : \mathbb{P}\, COBE\_Normal$
$ex\_exceptions : \mathbb{P}\, COBE\_EE$
$ge\_exceptions : \mathbb{P}\, OBEH\_GE \quad msgs : \mathbb{P}\, MSG$
$except\_context : OBEH\_GE \rightarrowtail\!\!\!\!\!\!+ \; COBE\_EH$
$except\_handle : COBE\_EH \rightarrow COBE\_State$
$co\_backup : COBE\_Normal \times B\_Job \rightarrow COBE\_State$
$co\_recover : COBE\_Normal \times R\_Job \rightarrow COBE\_State \times R\_Job$

$nor\_states \cap ex\_exceptions \cap ge\_exceptions = \varnothing$
$\mathrm{dom}\, except\_context \subseteq ge\_exceptions$

$cur\_state : COBE\_State \quad checkpoint : COBE\_Normal$
$ge\_rec\_sig, vflag : SIG$

$cur\_state \in nor\_states \cup ex\_exceptions \cup ge\_exceptions \cup \{COBE\_Fail\}$
$checkpoint \in nor\_states$

INIT

$cur\_state \in nor\_states$
$checkpoint = cur\_state$
$ge\_rec\_sig = 0$
$vflag = 0$

Re_Init

$\Delta(cur\_state, checkpoint)$

$cur\_state' \in nor\_states$
$checkpoint' = cur\_state'$

Rec_request

$req\_co? : \downarrow COBE$

$req\_co? = self$

Send_response

$\Delta(vflag) \quad co\_res! : \downarrow COBE$

$vflag = 0 \quad co\_res! = self$
$vflag' = 1$

Backup_data

$\Delta(cur\_state)$
$backup\_job? : B\_Job$

$cur\_state \in nor\_states$
$cur\_state' = co\_backup(cur\_state, backup\_job?)$

Restore_data

$\Delta(cur\_state)$
$rreq\_job?, reco\_job! : R\_Job$

$cur\_state \in nor\_states$
$(cur\_state', reco\_job!) = co\_recover(cur\_state, rreq\_job?)$

EE_send

$ex\_exception! : COBE\_EE$

$cur\_state \in ex\_exceptions$
$ex\_exception! = cur\_state$

GE_rec

$\Delta(cur\_state, ge\_rec\_sig)$
$ge\_exception? : OBEH\_GE$

$cur\_state' = ge\_exception?$
$ge\_rec\_sig' = 1$

GE_handle

$\Delta(cur\_state, ge\_rec\_sig)$

$cur\_state \in ge\_exceptions \land ge\_rec\_sig = 1$
$except\_handle(except\_context(cur\_state)) \in nor\_states \Rightarrow cur\_state' = except\_handle(except\_context(cur\_state))$
$except\_handle(except\_context(cur\_state)) = COBE\_Fail \Rightarrow cur\_state' = COBE\_Fail \land ge\_rec\_sig' = 0$

The local variables: *nor_state*, *ex_exceptions*, and *ge_exceptions* specify that the *COBE* be in normal state, external exception state and global exception state respectively. The variable *msgs* specifies message flows among entities. The meanings of *except_context* function and *except_handle* function are similar to the corresponding functions in the *NCOBE* class schema. The *co_backup* and *co_recover* functions indicate that the state transition of a *COBE* after executing a backup job and a recovery job respectively.

The state schema declares variables: *cur_state*, *checkpoint*, and *ge_rec_sig*, whose meanings are similar with those in the *NCOBE* class schema. The variable *vflag* is a boolean value representing whether the *COBE* is visited by any *NCOBE*. The INIT operation is used to initiate variables *cur_state*, *checkpoint*, *ge_rec_sig*, and *vflag*. After the *NCOBE* sends service request, the *COBE* will receive the request by the *Rec_request* operation. If the *COBE* permits the request, it will process the request and return an answer to the *NCOBE* via *Send_response* operation. For data backup service, the *COBE* will receive the data from client and store it in suitable storage area, which is specified in the *Backup_data* operation schema. For data recovery, the data transmission process is opposite to the data backup, which is specified in *Restore_data* operation. The operation schemas: *EE_send*, *GE_rec*, and *GE_handle* specify similar functions with the corresponding operation schemas in the *NCOBE* class schema.

4) Online Backup Exception Handler (OBEH): The *OBEH* class schema specifies how the *OBEH* in the *FOBSM* deals with the external exceptions raised by *NCOBE* or *COBE*. The *Exception_rec_nco* operation denotes that the *OBEH* receives an *ex_exception?* from a *NCOBE*. The *Exception_rec_co* operation specifies that the *OBEH* receives an *ex_exception?* from a *COBE*. The *Exception_handle* indicates that the *OBEH* resolves multiple exceptions raised by *NCOBE* or *COBE* by using *exception_generator* function, and send out *ge_exception!* to the related entities.

OBEH

$exception\_generator : \mathrm{seq}_1\, EXCEPTION \to OBEH\_GE$

$exceptions :$
$\quad \mathrm{seq}\, EXCEPTION$

INIT
$exceptions = \langle\ \rangle$

Exception_rec_nco
$\Delta(exceptions)$
$ex\_exception? :$
$NCOBE\_EE$

$exceptions' = exceptions$
$\frown\langle ex\_exception?\rangle$

Exception_rec_co
$\Delta(exceptions)$
$ex\_exception? :$
$COBE\_EE$

$exceptions' = exceptions$
$\frown\langle ex\_exception?\rangle$

Exception_handle
$\Delta(exceptions) \quad ge\_exception! : OBEH\_GE$

$exceptions \neq \langle\ \rangle$
$ge\_exception! = exception\_generator(exceptions)$
$exceptions' = \langle\ \rangle$

5) Fault-tolerant Online Backup Service System (FOBSS): The *FOBSS* class schema specifies how the components communicate and cooperate with each other to accomplish the functions and reach reasonable fault tolerant performance.

FOBSS

$system\_state : \mathbb{P} \downarrow NCOBE \to RESULT$

$ncos : \mathbb{P} \downarrow NCOBE \quad cos : \mathbb{P} \downarrow COBE \quad obeh : \downarrow OBEH$
Δ
$nco\_fail : \mathbb{P} \downarrow NCOBE$

$\forall\, nco : ncos \mid$
$nco.cur\_state = NCOBE\_Fail \bullet nco \in nco\_fail$

INIT
$\forall\, nco : ncos \bullet nco.\mathrm{INIT}$
$\forall\, co : cos \bullet co.\mathrm{INIT}$
$obeh.\mathrm{INIT}$

System_recover
$system\_state(nco\_fail) = tolerate$
$\forall\, nco : ncos \bullet nco.Re\_Init$
$\forall\, co : cos \bullet co.Re\_Init$

$Serv\_req \mathrel{\widehat=} \bigwedge nco : ncos \bullet nco.Send\_request \parallel$
$\quad (\bigwedge co : cos \bullet co.Rec\_request)$
$Serv\_resp \mathrel{\widehat=} \bigwedge co : cos \bullet co.Send\_response \parallel$
$\quad (\bigwedge nco : ncos \bullet nco.Rec\_response)$
$Provide\_serv \mathrel{\widehat=} (\bigwedge nco : ncos \bullet nco.Trans\_data \parallel$
$\quad (\bigwedge co : cos \bullet co.Backup\_data)) \lor$
$\qquad (\bigwedge nco : ncos \bullet nco.Recover\_request \parallel$
$\qquad (\bigwedge co : cos \bullet co.Restore\_data) \parallel$
$\qquad (\bigwedge nco : ncos \bullet nco.Recover\_data))$
$EE\_rec \mathrel{\widehat=} (\bigwedge nco : ncos \bullet nco.EE\_send \parallel$
$\quad obeh.Exception\_rec\_nco) \lor$
$\qquad (\bigwedge co : cos \bullet co.EE\_send \parallel obeh.Exception\_rec\_co)$
$GE\_send \mathrel{\widehat=} (obeh.Exception\_handle \parallel$
$\quad (\bigwedge nco : ncos \bullet nco.GE\_rec)) \lor$
$\quad (obeh.Exception\_handle \parallel (\bigwedge co : cos \bullet co.GE\_rec))$

Since the different exceptions have different impacts upon the system, when a *NCOBE* fails, the *FOBSS* can tolerate the failure, whereas a *COBE* fails, the *FOBSS* cannot tolerate. The variable *nco_fail* represent the failed *NCOBE*. When the *NCOBE* fails, the *System_recover* is used to initiate the related entities.

The remaining operations bind corresponding operations defined in other class schemas to achieve the interaction and cooperation by using the operator $\parallel$ to compose two operations. The *Serv_req* operation denotes how to send and receive service request among *NCOBE* and *COBE*. The *Serv_resp* operation specifies the transmission and reception of response to the request. The *Povide_serv* represents how to provide service by COBE to NCOBE. The *EE_rec* operation denotes that the *OBEH* receives the external exceptions from different entities. The *GE_send* specifies that *OBEH* resolves all the received external exceptions and sends the global exception to all the related entities.

IV. Reasoning about the FOBSM

Reasoning mechanism is used to show that the system possesses relative fault tolerant properties. Based on the Object-Z model of *FOBSM*, we can reason about fault tolerant properties of *FOBSM* by following the reasoning rules of Object-Z. We summarize four generic fault tolerant properties of *FOBSM* in the following sections.

A. NCOBE raises an internal exception

When a *NCOBE* raises an internal exception and deals with it by means of invoking an internal exception handler, all the *COBEs* in the system will not be influenced. This property can be expressed formally as follows:

Theorem

$$\begin{array}{l} FOBSS :: \exists\, nco : ncos \mid nco.cur\_state \in nco.in\_exceptions \\ \quad \vdash \forall\, co : cos \bullet co.cur\_state \in co.nor\_state \end{array}$$

First we will analyze the reasoning process. Once a *NCOBE* raises an internal exception, it will invoke a corresponding internal exception handler to deal with the exception. If this exception cannot be handled successfully, the *NCOBE* should propagate it as an external exception to *OBEH*. Since the *NCOBE* only needs to use local resource to handle the exception, the whole process is transparent to the other entities of the system, all the *COBE* will not be influenced.

Proof

$$\begin{array}{l} FOBSS :: \exists\, nco : ncos \mid nco.cur\_state \in nco.in\_exceptions \vdash \\ \quad nco.IE\_handle \\ FOBSS :: \exists\, nco : ncos \mid ncos \in \mathbb{P} \downarrow NCOBE \vdash nco \in\downarrow NCOBE \\ NCOBE :: IE\_handle \vdash \\ \quad cur\_state' \in nor\_states \vee cur\_state' \in ex\_exceptions \\ \hline FOBSS \vdash nco.cur\_state' \in nco.nor\_states \vee \\ \quad nco.cur\_state' \in nco.ex\_exceptions \\ FOBSS :: co : cos \vdash co.cur\_state \neq nco.cur\_state' \\ \hline FOBSS :: \forall\, co : cos \vdash co.cur\_state \in co.nor\_state \end{array}$$

B. NCOBE raises an external exception

When a *NCOBE* raises one external exception, which will be propagated to *OBEH* and transformed into a global exception by *OBEH*, all the related entities in the *FOBSS* should be informed about the global exception. This property can be expressed as follows:

Theorem

$$\begin{array}{l} FOBSS :: \exists\, nco : ncos \mid nco.cur\_state \in nco.ex\_exceptions \vdash \\ \quad \forall\, nco : ncos;\ co : cos \bullet nco.ge\_rec\_sig' = 1 \wedge \\ \quad co.ge\_rec\_sig' = 1 \end{array}$$

First we will analyze the reasoning process. When a *NCOBE* cannot handle the internal exception successfully, it will propagate it as an external exception to the *OBEH*. The *OBEH* would use the *exception_generator* to resolve the received exception to a global exception, called *ge_exception*. The related entiteis in the system should be informed about the exception.

Proof

$$\begin{array}{l} FOBSS :: \exists\, nco : ncos \mid nco.cur\_state \\ \quad \in nco.ex\_exceptions \vdash nco.EE\_send \\ FOBSS :: \exists\, nco : ncos \mid ncos \in \mathbb{P} \downarrow NCOBE \\ \qquad \vdash nco \in\downarrow NCOBE \\ NCOBE :: EE\_send \vdash ex\_exception! = cur\_state \\ \hline FOBSS \vdash obeh.Exception\_rec\_nco \\ FOBSS \vdash obeh \in\downarrow OBEH \\ \hline FOBSS \vdash exceptions' = exceptions \frown \langle ex\_exception? \rangle \\ OBEH :: except\_handle \mid exceptions \neq \langle\ \rangle \vdash \\ \qquad ge\_exception! = exception\_generator(exceptions) \\ FOBSS \vdash GE\_send \\ \hline FOBSS :: nco : ncos \vdash nco.GE\_rec \\ NCOBE :: GE\_rec \vdash ge\_rec\_sig' = 1 \\ FOBSS :: co : cos \vdash co.GE\_rec \\ FOBSS :: \exists\, co : cos \mid cos \in \mathbb{P} \downarrow COBE \vdash co \in\downarrow COBE \\ COBE :: GE\_rec \vdash ge\_rec\_sig' = 1 \\ \hline FOBSS :: \forall\, nco : ncos;\ co : cos \bullet \\ \quad nco.ge\_rec\_sig' = 1 \wedge co.ge\_rec\_sig' = 1 \end{array}$$

C. COBE raises an external exception

When an external exception is raised by a *COBE*, propagated to *OBEH* and transformed into a global exception, the *FOBSS* will inform all the related entities to deal with the global exception. This property can be expressed formally as follows:

Theorem

$$\begin{array}{l} FOBSS :: \exists\, co : cos \mid co.cur\_state \in co.ex\_exceptions \vdash \\ \quad \forall\, nco : ncos;\ co : cos \bullet nco.ge\_rec\_sig' = 1 \wedge \\ \quad co.ge\_rec\_sig' = 1 \end{array}$$

First we will analyze the reasoning process. There are two cases that external exceptions may be raised by the *COBE*. One is during the process of *COBE* dealing with service request, the other is during the communication between two *COBEs*. When external exception occurring, the *COBE* that originates the exception will submit it to the *OBEH*, which should use the function, called *exception_generator* to cope with it. The result of the function is a global exception, called *ge_exception*. Then the system will inform all the related entities to deal with the exception.

Proof

$$\begin{array}{l} FOBSS :: \exists\, co : cos \mid co.cur\_state \in co.ex\_exceptions \vdash \\ \qquad co.EE\_send \\ FOBSS :: \exists\, co : cos \mid cos \in \mathbb{P} \downarrow COBE \vdash co \in\downarrow COBE \\ COBE :: EE\_send \vdash ex\_exception! = cur\_state \\ \hline FOBSS \vdash obeh.Exception\_rec\_co \end{array}$$

$$FOBSS \vdash obeh \in\downarrow OBEH$$

$$OBEH \vdash exceptions' = exceptions \frown \langle ex\_exception? \rangle$$
$$OBEH :: Exception\_handle \mid exceptions \neq \langle\ \rangle \vdash ge\_exception! = exception\_generator(exceptions)$$
$$FOBSS \vdash GE\_send$$

$$FOBSS :: co : cos \vdash co.GE\_rec$$
$$COBE :: GE\_rec \vdash ge\_rec\_sig' = 1$$
$$FOBSS :: nco : ncos \vdash nco.GE\_rec$$
$$FOBSS :: \exists nco : ncos \mid ncos \in \mathbb{P} \downarrow NCOBE \vdash nco \in\downarrow NCOBE$$
$$NCOBE :: GE\_rec \vdash ge\_rec\_sig' = 1$$

$$FTSystem :: \forall nco : ncos;\ co : cos \bullet nco.ge\_rec\_sig' = 1 \wedge co.ge\_rec\_sig' = 1$$

D. NCOBE fails

When a *NCOBE* fails, the *FOBSS* can tolerate the failure, which means that all the entities in the system can recover to normal state. This property can be expressed formally as follows:

Theorem

$$FOBSS :: \exists co : cos \mid cos \in \mathbb{P} \downarrow COBE \vdash co \in\downarrow COBE$$
$$:: \exists nco : ncos \mid nco.cur\_state = NCOBE\_Fail \vdash$$
$$\forall nco : ncos;\ co : cos \bullet nco.cur\_state' \in nco.nor\_states \wedge co.cur\_state' \in co.nor\_states$$

We will describe the reasoning process first. According to the definition of *FOBSA* class schema, when a *NCOBE* fails, the system will implement the function of *system_state*, whose result is *tolerate*. In this case, system will further startup *INIT*, which operates towards both *COBE* and *NCOBE*. By this *INIT* operation, all the entities will recover to normal states.

Proof

$$FOBSS :: \exists nco : ncos \mid nco.cur\_state = NCOBE\_Fail \vdash nco\_fail = \{nco\}$$
$$FOBSS :: nco\_fail = \{nco\} \vdash system\_state(nco\_fail) = tolerate$$

$$FOBSS \vdash system\_state(nco\_fail) = tolerate$$
$$FOBSS :: system\_state(nco\_fail) = tolerate \vdash System\_recover$$

$$FOBSS \vdash System\_recover$$
$$FOBSS :: System\_recover \vdash \forall nco : ncos;\ co : cos \bullet nco.\textsc{Init} \wedge co.\textsc{Init}$$

$$FOBSS :: nco : ncos \vdash nco.\textsc{Init}$$
$$FOBSS :: nco : ncos;\ ncos : \mathbb{P} \downarrow NCOBE \vdash nco \in\downarrow NCOBE$$
$$NCOBE :: \textsc{Init} \vdash cur\_state \in nor\_states$$
$$FOBSS :: co : cos \vdash co.\textsc{Init}$$
$$FOBSS :: co : cos;\ cos : \mathbb{P} \downarrow COBE \vdash co \in\downarrow COBE$$
$$COBE :: \textsc{Init} \vdash cur\_state \in nor\_states$$

$$FOBSS :: nco : ncos;\ co : cos \vdash nco.cur\_state' \in nco.nor\_states \wedge co.cur\_state' \in co.cur\_states$$

V. Conclusions

Only when the fault tolerance of online backup service software reaches a reasonable level, can clients use the software at ease. This paper proposes a fault-tolerant online backup service model called *FOBSM*, which comprises four components: backup client (*BC*), backup server (*BS*), storage server (*SS*), and online backup exception handler (*OBEH*). The first three components constitute the three-party functional units, whereas *OBEH* serves as the centralized exception handling mechanism, which is devised to receive the external exceptions raised by the other entities, transform them into a global exception, and propagate it to related entities to handle.

The proposed *FOBSM* possesses the following fault-tolerant properties: (1) When a *NCOBE* raises an internal exception and deals with it by means of an internal exception handler, all the *COBEs* in the system will not be influenced. (2) When a *NCOBE* raises one external exception, which will be propagated to *OBEH* and transformed into a global exception, all the related entities in the *FOBSS* should be informed about the global exception. (3) When an external exception is raised by a *COBE* and transformed into a global exception by *OBEH*, the *FOBSS* will inform all the related entities to deal with the global exception. (4) When a *NCOBE* fails, the *FOBSS* can tolerate the failure.

To provide precise idioms to the system designers, we use Object-Z formal language to specify *FOBSM* and reason about its fault tolerant properties by using the related reasoning rules.

In this paper, Object-Z is adopted into the formal modeling and reasoning of online backup service software structure, which make the research more systematic and scientific. Moreover, the functional units on the client side and server side are categorized into non-critical entity and critical entity respectively, and on that basis, fault tolerance is realized, it has substantial reference value to similar service software design on the aspect of fault tolerance.

Currently, the formal specification and reasoning process is accomplished manually, which is somewhat difficult, fussy and error prone. The further research direction may be checking the consistency and completeness of the specification, and proving the fault tolerant properties automatically by virtue of tools.

Acknowledgements

We thank the reviewers for their helpful comments and insights. This work was supported by National Basic Research 973 Program under Grant 2004CB318201, the Program for New Century Excellent Talents in University NCET-06-0650, the Program for Changjiang Scholars and Innovative Research Team in University(No.IRT-0725).

References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehrbm, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.

[2] R. de Lemos, P. A. de Castro Guerra, and C. M. F. Rubira, "A fault-tolerant architectural approach for dependable systems," *IEEE SOFTWARE*, vol. 43, no. 1, pp. 80–87, 2006.

[3] P. Guerra, C. Rubira, and R. de Lemos, "An idealized fault-tolerant architectural component," in *Proceeding of the 24th International Conference on Software Engineering-Workshop on Architecting Dependable Systems*, 2002.

[4] A. F. Garcia, D. M. Beder, and C. M. Rubira, "An exception handling software architecture for developing fault-tolerant software," *IEEE*, pp. 311–320, 2000.

[5] L. Yuan, J. Dong, and J. Sun, "Modeling and customization of fault tolerant architecture using Object-Z/XVCL," in *Proc. Asia Pacific Software Engineering Conference'06(APSEC'06)*. IEEE Computer Society Press.

[6] L. Yuan, J. Dong, J. Sun, and H. Basit, "Generic fault tolerant software architecture reasoning and customization," *IEEE Transactions on Reliability*, vol. vol 55, no. 3, pp. 421–435, 2006.

[7] R. J. Allen, *A Formal Approach to Software Architecture*. Technical Report CMU-CS-97-144. Carnegie Mellon University, 1997.

[8] J. M. Spivey, *The Z notation: A Reference Manual*. 2nd ed. International Series in Computer Science. Prentice-Hall, 2001.

[9] G. Smith, *The Object-Z Specification Language*, 1999.

[10] R. Duke, L. Wildman, and B. Long, "Modelling java concurrency with object-z," in *Proceedings of the First International Conference on Software Engineering and Formal Methods(SEFM'03)*, 2003.

[11] T. McComb and G. Smith, "Architectural design in object-z," in *Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04)*, 2004.

[12] R. Campbell and B. Randell, "Error recovery in asynchronous system," *IEEE Transactions on Software Engineering*, vol. 12, no. 8, pp. 811–826, 1986.