FT-SOAP: A Fault-tolerant web service

Deron Liang[1,2], **Chen-Liang Fang**[3,4], **Chyouhwa Chen**[4]

[1]Institute of Information Science, Academia Sinica, Taipei, Taiwan, 11529, R.O.C.

[2]Department of Computer and Information Science, National Taiwan Ocean University, Keelung, Taiwan, R.O.C.

[3]Department of Information Management, Jin-Wen Institute of Technology, Taipei, Taiwan, R.O.C.

[4]Department of Electronic Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, R.O.C.

Keywords: fault-tolerance, web service, SOAP, distributed computing environment

Abstract

Security is the major concern that refrains people from conducting commerce electronically. The security concerns related to electronic commerce (EC) includes transaction security *and* system security. *We can partially address the transaction security issues by message distribution middleware such as* simple object access protocol *(SOAP), which is one of the information technology (IT) infrastructures that facilitate EC. It requires a comprehensive set of IT in order to address the system security issues where fault-tolerance technology is one of the core technologies to enhance the system survivability after attack. Based on our preliminary investigation, we conclude that the current SOAP architecture is lack of mechanism to build a highly reliable EC system. We propose a comprehensive fault-tolerance framework based on the current SOAP architecture in order to address the system security issues for EC. We consider this research is the continuing effort of our previous work on fault-tolerant CORBA, thought there are similarities and differences between these two technologies. We propose a standard recommendation that outlines a set of interfaces named fault-tolerant SOAP* (FT-SOAP)*. The FT-SOAP includes four functionalities and three basic components. The SOAP architecture needs modifications/extensions to meet the requirements of FT-SOAP. Our design takes the advantages of SOAP features. The new proposed components in SOAP and the extensions of SOAP engine is backward compatible to non-fault-tolerant SOAP system. Our approach can be used to develop other supports on SOAP.*

1 Introduction

Recently, SOAP (Simple Object Access Protocol) has become the most popular technology to develop web service application. The application of electronic commerce is such an example. According to Forrester Research, annual B2B commerce is expected to grow from about $43 Billion in 1998 to about 1.3 Trillion in 2003. During the same period, business-to-consumer (B2C) commerce is expected to grow from $7.8 Billion to $108 Billion! [Free98][RC98].

The electronic commerce activities rely on five infrastructure technologies [Zwas96]. These infrastructure technologies are: 1) Common Business Service Infrastructure, 2) Messaging and Information Distribution Infrastructure, 3) Multimedia Content and Network Publishing Infrastructure, 4) Network Infrastructure, and 5) Interface Infrastructure. The Messaging and Information Distribution Infrastructure is called distributed middleware in information industry, such as OMG CORBA [OMG98], Microsoft's DCOM [MSFT98] and SOAP [W3C00].

Recently, the SOAP is becoming popular and important in distributed applications. According to our survey, the SOAP standard has the following advantages:

1. The SOAP message transfer is based on HTTP for SMTP protocol. These protocols can transfer through firewall and be well managed.
2. Since SOAP relies on HTTP/SMTP, the advantages of HTTP/SMTP can be applied to SOAP, such as proxy and SSL.
3. The SOAP message transfer model is based on XML standard. IThe integration with other standard is easy to achieve.
4. The SOAP is highly extensible.

There exist many risks when we use SOAP in EC. These risks, including system failures and intrusions, could cause customers serious losses. Recent researches have reported that security relative intrusion cases are increasing every year. Some polls also reported security issue is the most serious barrier for shopping on Internet. In general, a good EC security policy includes 5 phases: 1) frequently update vulnerability database and patch the system, 2) install intrusion detection system, 3) detect intrusion, 4) start the standard procedure for emergence response when the system is attacked, and 5) computer forensics. In general, system manager uses software fault-tolerance to increase the EC system survivability [CHI01][NMN00].

The security issues include transaction security and system security etc. We apply SOAP technology to satisfy the partial requirements of transaction security. For system security, most researches and commercial products focus on intrusion detection including IDS, anti-virus, and firewall. The response and/or recovery mechanism after Internet attacks is lack

in most products and research works. According to our recent survey, the security related issue, especially intrusion response, is lack in SOAP specification. That becomes an important deficiency for developing SOAP-based applications. Therefore, we need a total solution by using software fault-tolerance technology to enhance the system survival capability after Internet attacks.

Based on our previous work FT CORBA [LFY99], we propose a fault-tolerant SOAP based middleware platform. We have two major targets in this work: 1) to define a fault-tolerant SOAP service standard recommendation, and 2) to implement an FT-SOAP service prototype. The standard recommendation is presented in section 3 in detail. The prototype implementation is discussed in section 4. Our performance experiments on the prototype have shown the efficiency of the service. We shall discuss this in the end of this paper.

The

SOAP-based web service lacks FT support when it is used in critical tasks. We identify the needed FT functionalities for SOAP. Based on our analysis, three basic components RM, fault manager, and logging/recovery mechanism are needed to support FT in SOAP. The replication manager manages and constitutes FT groups. The fault managers, including fault notifier and fault detector etc., is used to monitor protected web service group. The logging/recovery mechanism is used to log the service activities and perform recovery process for the web service group. The SOAP architecture needs modifications/extensions to meet the requirements of FT-SOAP. Such as, a new WSG tag in WSDL is needed to describe group web service; an interceptor on request path to perform FT functions; extend the capabilities of SOAP engine to handle SOAP group object information system. Our design takes the advantages of SOAP features. For example, the new WSG apply the XML extensibility. The interceptor extend axis SOAP, which is compliant to SOAP 1.1, in future, the interceptor can be achieved by using intermiadary of new message processing model of the SOAP 1.2. The design has to be compliant to the current SOAP standard. The new proposed components in SOAP and the extensions of SOAP engine is backward compatible to non-fault-tolerant SOAP system. Our approach can be used to develop other supports on SOAP. Such as security for web service, routing in web service, and interoperating with other middleware, etc.

2 The fault-tolerance for web service

The FT-SOAP architecture is shown in Figure 1. We design four functionalities in the FT-SOAP system: replication management, fault management, logging/recovery mechanism and client FT transparency. As shown in Figure 1, The FT-SOAP consists three components,

replication manager (RM), fault managers, and logging/recovery mechanism, in the system. The RM performs the replication management including group constitution and membership management. The fault managers, fault notifier and fault detector as shown in the figure, performs the fault detection and fault management functions. For SOAP 1.1, the logging/recovery mechanism is implemented as an interceptor in SOAP core engine. The interceptor captures and logs the invocation activities for recovery process. We shall use a complete example to explain the usage of our FT-SOAP system. The complete example includes initialization phase, run-time phase, and fault recovery phase. The fault tolerant web service group is constituted in the initialization phase. During the run-time phase, the logging mechanism logs the activities of all arrival requests. When fault occurs, the recovery mechanism in the new primary server will perform the recovery process and then take over the primary server. If client is built on a FT-SOAP, the client will be unaware the fault happening in the server.

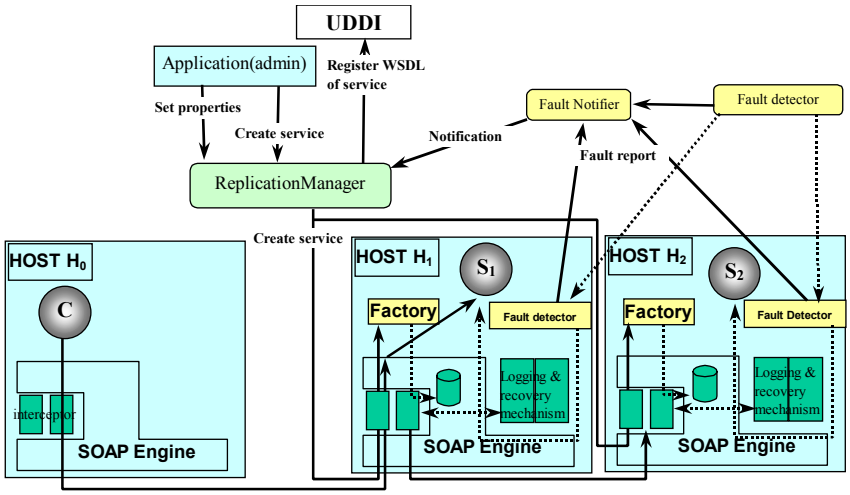

Figure 1 The FT-SOAP System Architecture

Group constitution

As shown in Figure 2, the application (AP) administration program uses the built-in group constitution function of FT-SOAP service to constitute a fault tolerant web service replication group. The AP may register their fault-tolerance policy to the FT-SOAP service. Such that, the FT-SOAP service may perform the fault detection and recovery process based on the given policy. The 6 steps procedure is shown in the following:

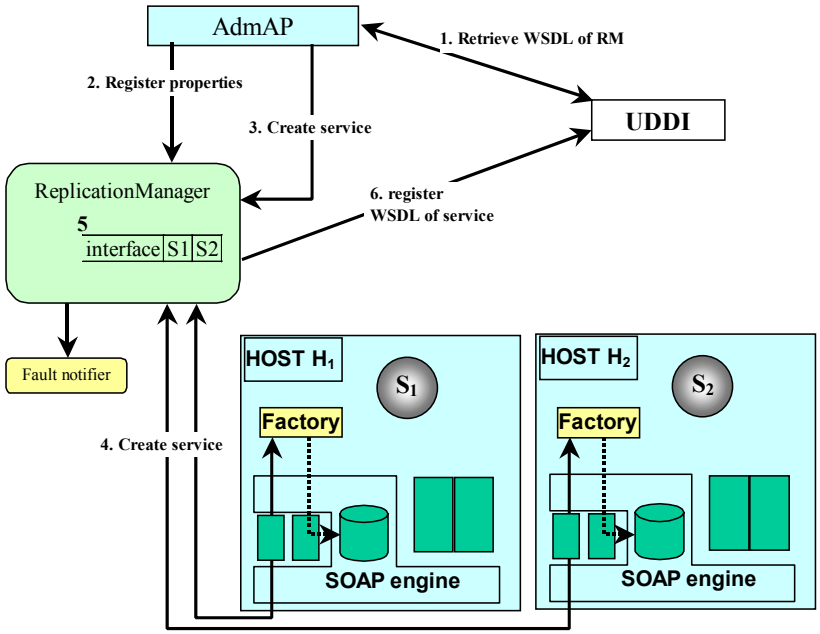

Figure 2 Constitute a web service group

Step 1: The user administration AP (AdmAP) retrieves WSDL of replication manager (RM) from UDDI.

Step 2: The AdmAP registers necessary replication properties to RM, such as replication style, and fault monitoring style.

Step 3: The AdmAP requests RM to create service.

Step 4: The RM requests the factory of each group member to deploy required web service based on the registered properties. That is, the factory registers all relative information of the web service to SOAP engine. The factory has to return the WSDL of the group member to RM.

Step 5: For each group member:

a. RM adds member to the web service group.

b. RM activates member by referring to the replication style.

c. RM decides whether to activate fault detection or not based on the replication style, and fault monitoring style.

d. RM subscribes fault notification from fault notifier for this member.

e. For passive replication style, RM chooses a member as primary.

f. RM composites Group Service WSDL, and activates the primary member.

Step 6: RM registers the WSDL of the service to UDDI.

Logging mechanism

Figure 3 shows a centralized logging mechanism. The logging mechanism logs the invocation activities to a centralized reliable logging file system for the future recovery process. When the recovery mechanism on new primary member is activated, the recovery mechanism retrieves the invocation logs and replays the invocations if necessary. The

scenario is show in the following:

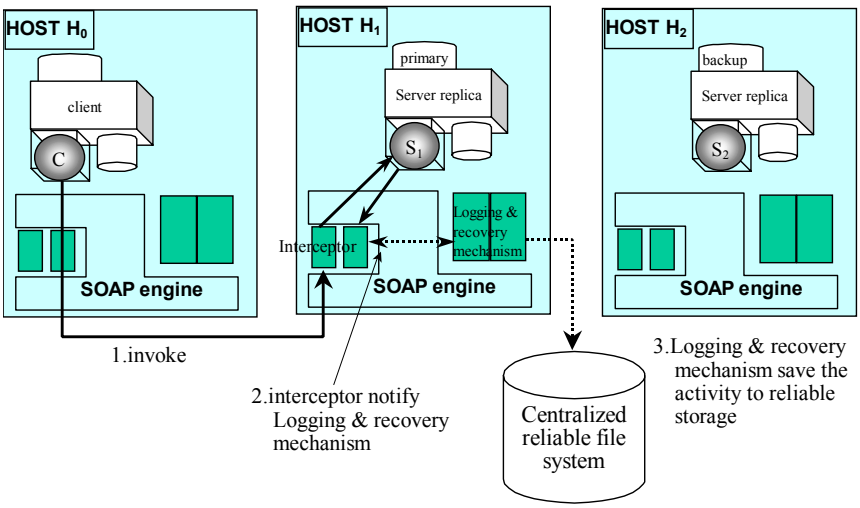

Figure 3 Logging management

Step 1: Client application make a request to the primary server replica of the web service.

Step 2: The interceptor of the SOAP engine in primary server replica has been called for notifying the completion of the request. The interceptor informs logging mechanism to log all necessary information of this request.

Step 3: The logging mechanism receives the logging notification and writes the information to a centralized reliable file system.

Fault detection and recovery

The Figure 4 depicts the scenario of fault management. The service coordinates fault detection for AP. When a fault is detected, the system notifies the corresponding devices to response the fault. Then the relative fault-tolerance information is updated for current state. The RM registers new service information to UDDI based on the current group state. The usage scenario is shown in the following steps:

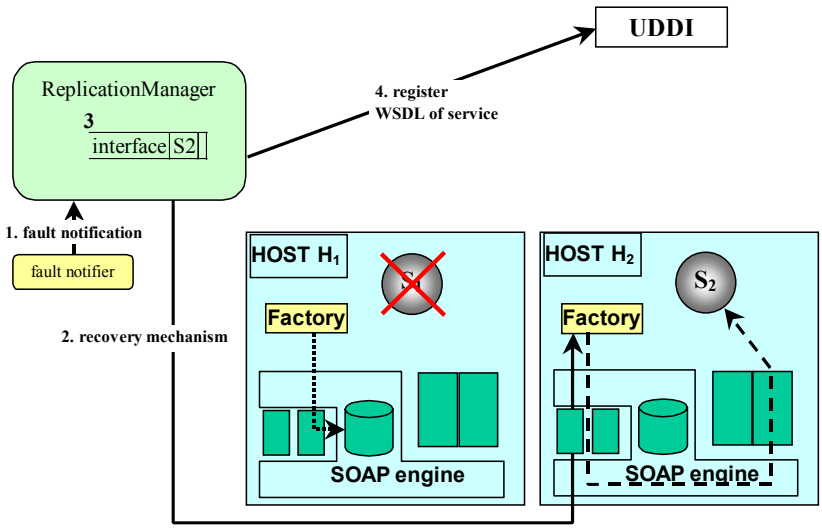

Figure 4 Fault management

Step 1: The fault notifier receives fault report from fault detector when the primary member S_1 is failed. The fault notifier sends fault notification to corresponding RM for the group.

Step 2: RM starts the recovery process on host H_2.

Step 3: RM modifies the new group service WSDL.

Step 4: RM registers the modified group service WSDL to UDDI.

Client side fault transparency

The client side FT-SOAP engine tries to re-transfer the invocation to other replica when the primary server is failed to response. Such that, the client AP unaware happening of the failure.

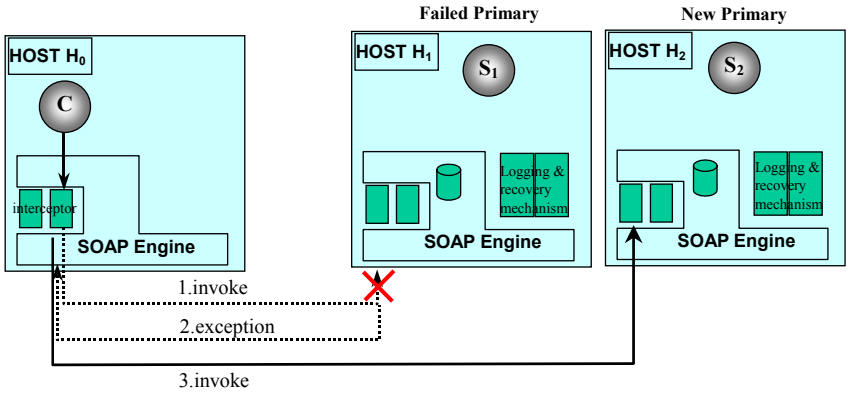

Figure 5 Client side fault transparency

Client:

Step 1: send request to primary service.

Step 2: primary is faulty and returns exception to client SOAP engine.

Step 3: The client SOAP engine retrieves the replicas information from the given WSG. The SOAP engine resends the request to all other replicas in sequence. If all replicas return failure exception, then client SOAP engine returns exception to client AP. The client AP needs to retrieve new WSG and try to invoke again starting from the step 1.

Server:

The interceptor in the server side SOAP engine decide whether to serve this arriving request or not based on the fault tolerant information in the SOAP engine. That is, the SOAP engine will return an exception to client if this replica is not primary.

3 The fault-tolerant SOAP service

In this section, we propose three basic components to support the fault tolerance on SOAP. Furthermore, an extended SOAP architecture is presented to meet the fault tolerant requirements.

Basically, we use service approach to support the fault-tolerance in SOAP. We propose three basic components, replication manager (RM), fault managers, and logging/recovery mechanism. The fault managers include fault notifier and fault detector. The RM and fault managers are defined as interfaces in web service description language (WSDL). We shall given detail discussion in this section later. Beside these basic components design, we propose extended SOAP architecture to achieve full fault tolerance supports in SOAP. The extended functionalities are extended WSDL schema for group service description: a new added web service group (WSG) tag; interceptor concept in SOAP; and extended object information in SOAP engine.

A new tag, web service group or WSG, is added in WSDL for a fault-tolerant WS group. The schema is shown Figure 6. The new WSG tag is designed for client side fault transparency. The client side SOAP engine will have a chance to inspect the WSG information if the invocation is failed. The client SOAP engine may try to send invocation to rest replicas in order till the invocation is successfully executed. If all replicas listed in the WSG could not response the request, then the SOAP engine returns a failure exception to client application. If client application got a failure exception, then it should consider retrieving new WSDL again. A sample WSG is shown in Figure 7

```
<WSG>
      <PRIMARY version=”xx.xx” location=”host :port” />
     <REPLICA version=”xx.xx” location=”host : port” />
     <REPLICA version=”xx.xx” location=”host : port” />
</WSG>
```

Figure 6 The extension to the WSDL: the service group tag <WSG/>

```
<WSG>
      <PRIMARY version=”1.0” location=”H1.our.domain: :8080” />
     <REPLICA version=”1.0” location=”H2.our.domain::8080t” />
</WSG>
```

Figure 7 The sample service group tag <WSG/>

As discussed in previous section, the replication management provides the following functionalities: create passive group, set property, membership management, query group member, and register fault notifier. These functionalities are divided into four interfaces: *PropertyManager*, *GenericFactory*, *ServiceGroupManager*, and *ReplicationManager*. We have to remind readers, these interfaces are defined in Java IDL for the readability only. The definitions of these interfaces are given in Figure 8, Figure 9, Figure 10, and Figure 11 respectively.

```
interface PropertyManager {
        void set_default_properties(in Properties props);
        Properties get_default_properties();
        void remove_default_properties(in Properties props);
        void set_type_properties(in typeID type_id, in Properties props);
        Properties get_type_properties(in typeID type_id);
        void remove_type_properties(in typeID type_id, in Properties props);
};
```

Figure 8 Interface PropertyManager

```
interface GenericFactory {
        Service create_service(in typeID id);
        void destory_service(in Service service);
};
```

Figure 9 Interface GenericFactory

```
interface ServiceGroupManager {
        ServiceGroup create_member(in TypeID type_id, in Location the_location);
        ServiceGroup add_member(in ServiceGroup srvGrp, in Service serviceMember);
        ServiceGroup remove_member(in ServiceGroup srvGrp, in Location the_location);
        ServiceGroup set_primary_member(in ServiceGroup srvGrp, in Location the_location);
        Locations locations_of_member(in ServiceGroup srvGrp);
}
```

Figure 10 Interface ServiceGroupManager

```
interface FaultNotifier: NS_consumer;
interface ReplicationManager:PropertyManager,GenericFactory,ServiceGroupManager, NS_consumer{
        void register_fault_notifier(in FaultNotifier fault_notifier);
```

```
        FaultNotifier get_fault_notifier();
}
```

Figure 11 ReplicationManager

In PropertyManager interface, the application may set up their desired fault tolerant attributes, or properties. We propose eight types of properties: ReplicationStyle, MembershipStyle, ConsistencyStyle, FaultMonitoringStyle, InitialNumberReplicas, MinimumNumberReplicas, FaultMonitoringIntervalandTimeOut, and CheckpointInterval. The detail property definitions are given in Appendix.

In order to log the invocation activities for future recovery process, we use interceptor concept to achieve the logging function. Current available platform, such axis and MS SOAP, only support SOAP 1.1, we have to extend SOAP engine to intercept request and log the activities. According to new developing SOAP 1.2, a new message model is proposed. That is, the new SOAP 1.2[] defines message routing in intermiadaries. The intermiadary can be used as interceptor to perform logging function. Therefore, the logging/recovery mechanism can be easily implemented as a portable logging mechanism.

An example

In Figure 12, we reuse the usage example in the previous section to show the usage of the replication related interfaces and properties, the sample code of the user administration AP is shown inFigure 13:

Step 1:The user administration AP (AdmAP) retrieves the WSDL of replication manager from UDDI.

Step 2:Before constituting a replication group, the user application sets the desired fault tolerance properties by calling the setting property function of PropertyManager interface. The properties include ReplicationStyle, MembershipStyle, InitialNumberReplicas, and MinimumNumberReplicas etc. The PropertyManager::set_default_properties() is to set up system default properties. PropertyManager::get_default_properties() is used to query the system default properties.

Step 3:The GenericFactory interface provides infrastructure-controlled membership style management. It provides an easiest and automatic way to manage a service group. After setting the desired fault tolerant properties by using PropertyManager interface, the user application can easily constitute the service group by calling GenericFactory::create_service() and GenericFactory::delete_service().

Step 4:The replication manager makes factory::create_service() requests to several hosts, H1 and H2, to deploy service and returns the WSDL of the service to Replication Manager.

Step 5: RM decides whether to activate fault detection or not based on the replication

style, and fault monitoring style. Then RM register to fault notifer by calling FaultNotifier::connect_structured_comsumer() to receive the fault notification of this service group. If it is passive replication style, the RM chooses the one member as primary by calling GenericFactory::activate_service() to the primary .

Step 6: RM composites the WSDL of the web service group, and registers to UDDI.

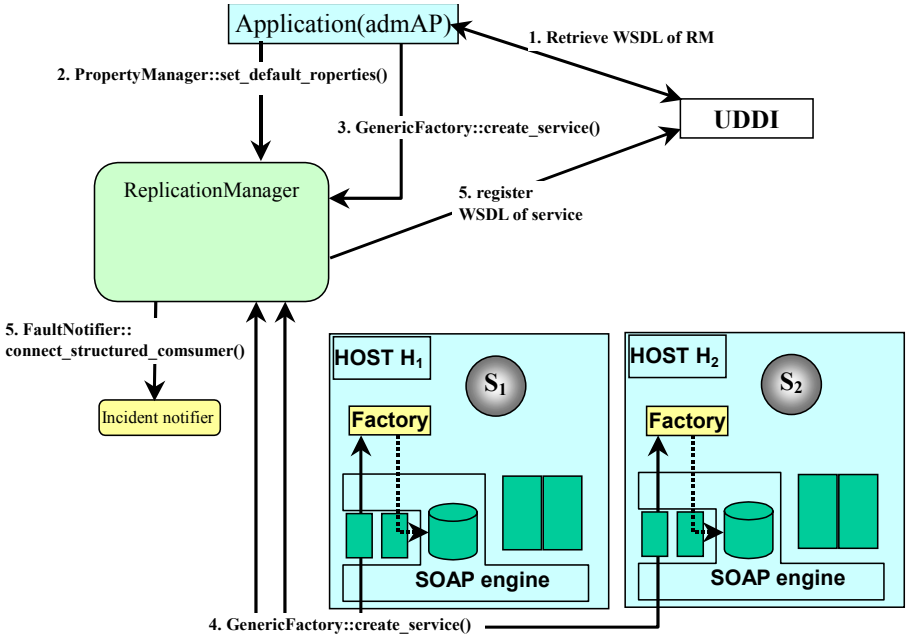

Figure 12 A replication management example

```
Import …;
public class AdmAP {
   …
   public static void main(java.lang.String[] args) {
   …
   // step 1 retrieve RM's WSDL from uddi
   …
   Properties props = new Properties();    //prepare propoerties
   ..
   Service service = new Service();
   Call call = (Call) service.createCall();
   …
   call.setOperationName(new QName("ReplicationManagerImpl","set_default_properties"));
   …
   call.invoke(new Object [] {typeId,props} );    //step 2 set properties
   Call call = (Call) service.createCall();
   …
   call.setOperationName(new QName("ReplicatonManagerImpl","create_service"));
   ..
   String WSDL = (String) call_createSrevice.invoke(new Object [] {typeId} ); //step 3 create_service
   …
   }
}
```

Figure 13 The samples java code of admAP

4 Performance evaluation

In this section, we shall discuss the implementation and evaluate our prototype system. Due to lack of the new SOAP 1.2 open source product, we use Apache Axis to implement our FT-SOAP. Furthermore, java SOAP product is suitable to develop platform-independent SOAP applications. We implement and evaluate our FT-SOAP on Linux and Microsoft 2000 platform. The both platforms are installed SunMicro java development tool kit J2SDK. The MS and Linux systems are installed on Pentium III PC with 512MB memory.

To deploy a fault tolerant web service by using our FT-SOAP, overhead will occur in our fault tolerant service components. According to our analysis, the system overhead comes from the logging mechanism, fault detection, and checkpointing. We shall discuss three experiments design and show the experiment result in this section.

The logging mechanism overhead happens when it intercepts the invocation and logs the information into a reliable file system. We measure two sets of response time on the same server with logging and without logging. The two hosts are installed on a local area network. Therefore, the experiment can avoid the interference of irrelevant network traffic. The logging overhead is obtained by comparing the two sets of response time. The logging execution time depends on the message size of the invocation. We vary the message size from 1KB to 512KB to measure the response time on the client. The result is shown in Figure 14. The result shows the system takes only 330 ms to log an invocation when the message size is up to 512KB. In general, the average message size of an invocation is less 10KB and the overhead is less than 1.5ms. The result shows that the logging overhead is very insignificant when compare to seconds response time for a web service invocation.

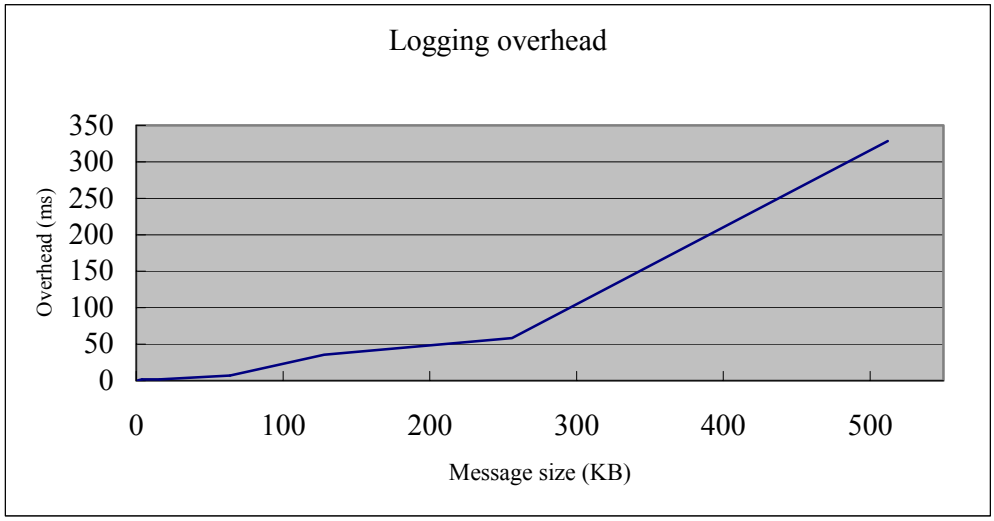

Figure 14 The logging overhead

With the same environment of logging overhead experiment, we use a distributed fault detector to poll a protected web service server and measure the CPU usage. We vary the polling time interval from 1 second to 30 second to measure the CPU utilization. As shown in Figure 15, the result shows the polling overhead is less than 2% of CPU utilization. In general, the polling interval is configured as 30 seconds and we found the detection overhead is less than 0.25% of CPU utilization.

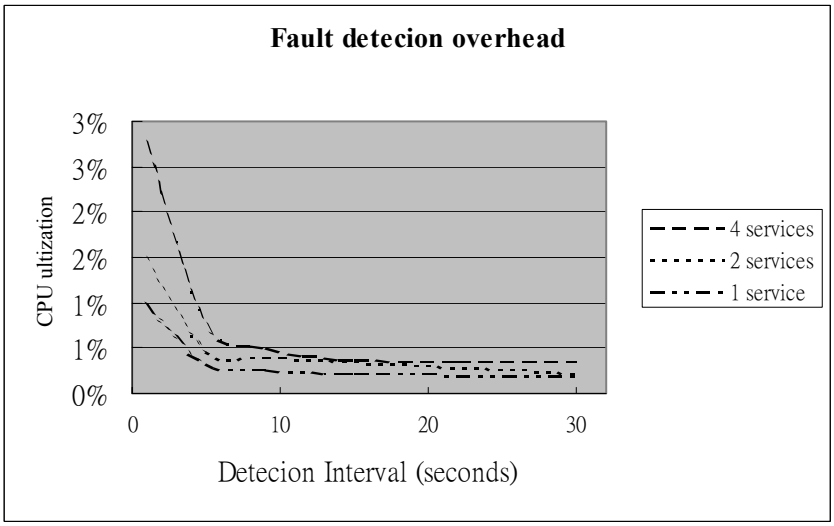

Figure 15 The fault detection overhead

References:

[CHI01] John Chirillo, Hack attacks denied - a complete guide to network lockdown, 2001.

[Free98] L. Freeman, “Net Drives B-to-B to New Highs World-Wide”, NetMarketing，January 1998 (www.netb2b.com).

[NMN00] Stephen Northcutt, Donald McLachlan, Judy Novak, Network Intrusion Detection: An Analyst's Handbook, September, 2000.

[MSFT98] Microsoft, DCOM Technical documents, http://msdn.microsoft.com/library/

[RC98] T. Retter and M. Calyniuk, Technology Forecast: 1998, (Price Waterhouse, March 1998).

[W3C00] Simple Object Access Protocol (SOAP) 1.1 , http://www.w3.org/TR/SOAP/

[Zwas96] V. Zwass, “Electric Commerce: Structures and Issues”, International Journal of Electric Commerce (Fall 1996).