Automatic Connection of Components for Dynamic Distributed Applications

Guojin Zhu
Dept. of Computer Science, Donghua University, Shanghai 201620, China
Email: gjzhu.dhu@163.com

Yongjiang Zhou
Dept. of Computer Science, Donghua University, Shanghai 201620, China
Email: zyj_163com@126.com

***Abstract*—Self-management, self-connection and self-configuration capabilities are emerging as important requirements for a generation of large-scale, highly dynamic, distributed applications. We propose a new solution to manage the connection of the components in a distributed application automatically. Each component in the distributed application is equipped with an autonomous connection unit (ACU). The ACU will allocate to its corresponding component an address consisting of an IP address and a port number. This address of the component will be delivered by the ACU to remote components, so that the remote components could communicate with the local component. In addition, the ACU could start up the component software module autonomously according to user settings. This solution is a generic model which could be adapted to many distributed applications. It has been successfully applied to a distributed online judge system for a campus contest in Donghua University.**

***Index Terms*— Distributed application, Automatic connection, Automatic configuration, Online judge, Network, Automatic communication**

I. INTRODUCTION

Today, many organizations are facing problems of managing the large collections of distributed resources. These might include the distributed file systems, distributed workstations and other distributed applications. The computers that hold the distributed application may be located in a room, spread across a building or campus, or even be scattered around the world. The configurations of these applications could change from time to time. The computers that a distributed application works on also could be changed rapidly. To a growing degree, the distributed application is expected to be self-configuring and self-managing, because the management of the configuration will be becoming an enormously complex operation as the number of components grows.

Now, the most widely-used, scalable distributed application is the Domain Name System (DNS), which is a directory service that associates IP address. The DNS was designed primarily to map domain names to IP addresses and mail servers. Besides, the DNS has been extended to make it more dynamic and support wider variety of applications. However, the DNS system has some limitation, for it was just designed to resolve the problem that the computer connects to the Internet. The Hadoop[1] is another distributed application architecture. To set up a real Hadoop cluster, the system operator needs to write the host name (or IP addresses) and the port number of each Hadoop node into the configuration file *hadoop-site.xml*. This distributed application architecture is mainly used in distributed file systems and large data processing.

A distributed application is such an application that executes a collection of commands to coordinate the actions of multiple processes on a network, so that all components cooperate together to finish a single or small set of related tasks [2]. The ability to connect remote users with remote resources in an open and scalable way is a good property of a distributed system. When we say open, we mean each component is continually open to interaction with other components. The scalable property means that the system can easily be altered to accommodate changes in the number of users, resources and computing entities. A distributed application can be very large and very powerful by giving the combined capabilities of all components in the distributed application [3][4][5]. But it is not easy - for a distributed application to be useful, it needs to have better connectivity. This is a difficult goal to achieve because of the complexity of the interactions between simultaneously running components.

The connection issue here for a distributed application refers to that every component in a distributed application is able to know the IP addresses and the port numbers of other components that it will communicate with. For a Hadoop node, it could be done by its reading the configuration file *hadoop-site.xml*. With the same problem, each component in the Donghua University On-line Judge [6] (DHUOJ) also needs to fetch these addresses from its configuration file. The DHUOJ is a distributed application whose software modules are installed on different computers. In order to communicate with each other, each of these modules has a

Corresponding author: Guojin Zhu; Email gjzhu.dhu@163.com.

doi:10.4304/jsw.8.6.1419-1427

configuration file that contains the IP addresses and port numbers of others. Unfortunately, the components of the DHUOJ do not always stay in the same computers. The DHUOJ may be installed on different computers for different contests at different sites. Furthermore, the topological structure of the DHUOJ may be changed to fit the scale of the contest. Each of computers could get its IP address by Dynamic Host Configuration Protocol (DHCP) [7]. However, it is not so easy to configure the DHUOJ. The system operator needs to know the IP addresses and available ports of all the computers that the DHUOJ modules will be installed on. After writing the IP addresses and available port numbers to the configuration file of each component, the system operator needs to test the installed DHUOJ system to make sure that the configuration could work correctly. It is obvious that automatic assignment of these IP addresses and port numbers can avoid the mistake that the system operator may make during the configuration operation.

In this paper, we propose a new distributed application management method called autonomous connection unit (ACU). Unlike the DNS, the ACU is not bound to specific servers or applications. The target systems maintained by the ACU could be different from each other. The ACU could make the target distributed application more flexible and scalable. We take the DHUOJ as an example to illustrate how the ACU works.

II. The Problems

Fig.1 illustrates the architecture of the DHUOJ that has been used in Donghua University. The below side is the hardware platform which is composed of networked normal computers. The hardware platform could be networked computers in a college computer room. The upper part of Fig.1 is the software modules of the DHUOJ. Each module is installed on the corresponding machine in the below side, see Table I.

The installation of the DHUOJ needs many operations. When setting up the module *Proxy*, for instance, the system operator needs to export its IP address and port number for the client to use the services that the component *Proxy* offers, such as paper downloading, program submitting and result delivering. Moreover, the IP address of the component *Primer* needs to be set into the configuration file of the component *Proxy*, otherwise the component *Proxy* will fail to get the web service of the component *Primer*. For the configuration of the component *Primer*, the system operator needs to make sure the port number that the component *Primer* will use to offer its web service is not occupied by other services on the same host machine. The configuration operations of the component *Schedule* are just as complicated as the component *Proxy*. The system operator needs to write the IP address of the component *Primer* into the configuration file of the component *Schedule*. Furthermore, the system operator needs to set the IP address and port number of the component *Judge* into the configuration file of the component *Schedule*, too. Sometimes, the component *Schedule* has to communicate with more than one *Judge* component. When a new *Judge* component is added to the DHUOJ, the system operator needs to restart the component *Schedule*.

TABLE I.
MODULES IN DHUOJ

No.	Module name	Corresponding machine
1	*Primer*	Primer Server
2	*Proxy*	Proxy Server
3	*Schedule*	Schedule Server
4	*Judge*	Judge Server
5	*Client*	Proxy Server (but run on PC)

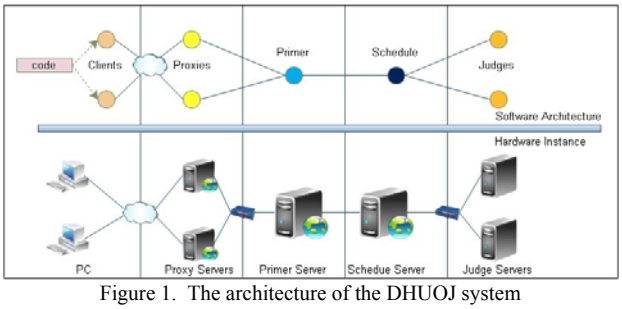

Figure 1. The architecture of the DHUOJ system

A. Assigning Addresses to Components

The address of a component consists of an IP address and a port number. The IP address assigned to a component should be the IP address of the host machine on which the component runs. The port number assigned to the component should not be occupied by other applications on the same host machine. Furthermore, the system operator needs to know the path and the file name of the configuration file in which the address of the component should be put.

Table II shows the information of the address which will be assigned to a component named *Proxy*. With this information, the system operator is able to assign the address to the component *Proxy* by writing the IP address 192.168.3.91 and the port number 80 into the configuration file "launch.jnlp" located in the directory "./apache-tomcat/webapps/dhuoj/". The information item *Location* in Table II indicates that the assigned address should be put in the configuration file as the new value of the attribute "codebase".

B. Writing Remote Addresses into Configuration Files

For each local component in DHUOJ, the IP address and port number of its remote components need to be set into its configuration file. For the component *Proxy*, it needs to know the address assigned to the component *Primer*. For the component *Schedule*, it needs to know not only the address assigned to the component *Primer* but also the addresses assigned to the components named *Judge*. Table III shows the remote address information

TABLE II.
THE LOCAL INFORMATION FOR THE COMPONENT *PROXY*

No.	Item name	Example
1	Module name	*Proxy*
2	Module IP	192.168.3.91
3	Module port	80
4	Configuration file	launch.jnlp
5	Relative path	./apache-tomcat/webapps/dhuoj/
6	Location	Attribute: codebase

TABLE IV.
THE REMOTE ADDRESS INFORMATION FOR THE COMPONENT *PROXY*

No	Item name	Example
1	Remote module	*Primer*
2	Remote role	*Primer*
3	Remote IP	222.204.211.72
4	Remote port	80
5	Configuration file	web.xml
6	Relative path	./apache-tomcat/webapps/dhuoj/WEB
7	Location	tag name: param-value

for the component *Proxy*. The address of the remote module *Primer* consists of the remote IP address 222.204.211.72 and the remote port number 80, which should be written into the configuration file *web.xml* located in the directory*./apache-tomcat/webapps/dhuoj/ WEB*. The information item *Location* in Table III indicates that this remote address should be put between the tags named *param-value* in the configuration file *web.xml*. The remote role is needed for the system operator to know which module should be used. Usually, the remote role shows the main function of this remote component.

C. Exchanging Local Address Information

The ACU could get the local address listed in Table II from the local computer. However, how does the ACU get the remote address listed in Table III? It is necessary for ACUs to exchange their local address information. Each ACU should deliver its local message information to remote ACUs in the network. Table IV shows an example of the exchanged information delivered by the ACU that the component *Primer* is equipped with. It tells the address of the component *Primer* as well as its name and role.

TABLE V.
THE EXCHANGED INFORMATION

No.	Item name	Example
1	Remote name	*Primer*
2	Remote Role	*Primer*
3	Remote IP	222.204.211.72
4	Remote Port	80

D. Applicable to Each Component

There are many modules in a distributed application [9]. It is not wise to develop many specific connection programs such that each of them is dedicated to a specific module. Such dedicated connection programs could not be used for the modules that might be developed to add to the distributed application in the future. To adapt to all possible modules including those that might be developed in the future, the ACU needs to handle all possible situations that might be encountered in the configuration operations for a distributed application. For example, the ACU needs to adapt to not only the component *Primer* but also the component *Proxy*. This requires that the ACU should set the address information to each component of a distributed application in a general way. Furthermore, the ACU needs to adapt to not only the DHUOJ but also other distributed applications.

III. AUTONOMOUS CONNECTION UNIT

In this solution, every component in a distributed application is equipped with an autonomous connection unit (ACU). The ACU will allocate to its corresponding component an address consisting of an IP address and a port number. This address of the component will be delivered by the ACU to remote components, so that the remote components could communicate with the local component. We regard these two actions as address information synchronization [8]. All the ACUs have the same structure, which consists of a local address information file, an address lookup table file, a task list file and an executive cell, see Fig. 2.

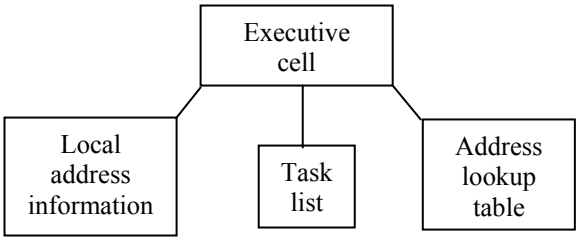

Figure 2. The structure of the ACU

A. The Executive cell

The executive cell will get the IP address and an available port number of the local machine on which the component has been deployed. This IP address and the port number are then written as the local address of the component into the local address information file. At the same time, the executive cell broadcasts this local address to the network. Every ACU in the network will receive the local address and keep it in its own address lookup table file. With the address lookup table file, the executive cell updates the task list file, so that it can then execute the tasks in the task list file to update the configuration file of the component. Finally, the executive cell starts up the component to run it.

B. The Local Address Information

The local address information file keeps the details of the local address, including the IP address of the host machine on which the component runs and the port number assigned to the component. The executive cell could modify the local address information file automatically. The information items that the executive cell needs to write into the local address information file is listed in Table V.

Component name: The name of the component that the

TABLE III.
THE LOCAL ADDRESS INFORMATION FOR THE COMPONENT *PROXY*

No.	Item name	Value
1	Component name	*Proxy*
2	Component role	*Proxy*
3	Component IP address	222.204.211.35
4	Component port number	80
5	Out IP address	192.168.3.91
6	ACU IP address	222.204.211.35
7	ACU TCP port	8001
8	ACU UDP port	8101

ACU works for. The system uses this name to distinguish one component from others.

Component role: The role of the component that the ACU works for. The role indicates what function that the component has. Components with the same role have the same function.

Component IP address: An IP address that is assigned to the component. This IP address is identical to the IP address of the host machine on which the component runs.

Component port number: A port number that is assigned to the component. This port number is an available port number on the host machine.

ACU IP address: The IP address of the ACU. The ACU uses this IP address to communicate with other ACUs.

ACU UDP port: An UDP port number that is used by the ACU. It is this port number that the ACU uses to search the whole local area network to get addresses of remote components when the ACU enters into the network at its first time.

ACU TCP port: A TCP port number that is used by the ACU. This TCP port number is very important for the system. Almost all the messages between ACUs are exchanged by the TCP protocol.

Out IP address: An IP address that is used to resolve a situation that there are two network interface cards in one computer. When an ACU comes across this situation, its executive cell will select one of these two IP addresses as the out IP address. For DHUOJ, the component *Proxy* needs one IP address as the "out IP address" for the client to use its service while the other one as the "component IP address" to communicate with the component *Primer*.

All the information items listed in Table V are stored in the local address information file, which is an xml file named *NativeMessage.xml*. Table VI lists the tags of items in the xml file *NativeMessage.xml*.

TABLE VII.
TAGS IN THE LOCAL ADDRESS INFORMATION FILE

No.	Item name	Tag name
1	Component name	OJ_MODEL_NAME
2	Component role	OJ_MODEL_ROLE
3	Component IP address	OJ_MODEL_IP
4	Component port number	OJ_MODEL_PORT
5	Out IP address	OUT_IP
6	ACU IP address	IP
7	ACU TCP port	tcp_port
8	ACU UDP port	udp_port

C. The Address Lookup Table

The address lookup table keeps addresses of all the components in a distributed application. This table is stored in an xml file, called address lookup table file. It will be updated by the ACU in time. Each entry in the address lookup table includes the name, the role, the IP address and the port number of the component, see Table VII.

Each component has a unique name but may have the same role as the others have. In Table VII, for example, the component whose name is *JackForB3* has the same role as the component whose name is *JackForB5*. Some component may have two addresses. For instance, the component named *JackForB3* has two addresses 192.168.3.91:80 and 222.204.211.35:80. The address indicated by "out" is the "out IP address" in the local address information file, whereas the address indicated by "in" is the "component IP address" in Table V.

TABLE VI.
THE ADDRESS LOOKUP TABLE

Name	Role	IP	In or out	Port
JackForB6	*Judge*	222.204.211.231	in	58002
JackForB3	*Proxy*	192.168.3.91	out	80
JackForB3	*Proxy*	222.204.211.35	in	80
JackForB5	*Proxy*	192.168.3.92	out	80
JackForB5	*Proxy*	222.204.211.36	in	80
JackForB4	*primer*	222.204.211.72	in	80
JackForB2	*Schedule*	222.204.211.83	in	80
JackForB	*Judge*	222.204.211.232	in	58002

D. The Task List

It is the task list that makes the ACU applicable to many distributed applications. The ACU is designed to be applicable to every component in a distributed application although the component that the ACU works for could be different from each other. For this reason, the task list is employed to record the operations that the ACU will execute to update the configuration file of its target component. Changing the operations described in the task list makes the same ACU work for a different component. In other words, the task list describes the functions that the ACU will perform. The system operator could add more operations to the task list if the component needs more configuration operations. All the configuration operations that the ACU needs to do come from the task list. Thus, an ACU could be adapted to a different component in a different distributed application by modifying its task list.

The task list is mainly composed of two parts: the commands used to start up the component and the operations used to update the configuration file of the component. The structure of the task list is shown in Table VIII.

There are several operations in a task list. Each operation updates one address in a configuration file, such as assigning a local address to the component or writing a remote address into the configuration file. Table IX shows an operation that instructs the ACU to write the address 222.204.211.72:3306 of the remote component *Primer* into the configuration file *hibernate.cfg.xml* of the component *Schedule* by putting the string "jdbc:mysql: //222.204.211.72:3306/dhuoj?useUnicode=true" between

TABLE VIII.
THE STRUCTURE OF TASK LIST FILE

Item name	Comment
Command 1	Add a path to a batch file
Command 2	Write a command to a batch file
Operation 1	Modify a xml configuration file
Operation 2	Add an item to a ini file
Operation 3	Add another item to ini file

TABLE IX.
AN OPERATION IN THE TASK LIST

No.	Field name	Example
1	Type	xml
2	File name	./persistence/hibernate.cfg.xml
3	Tag name	property
4	Property	Hibernate.connection.url
5	Attribute	
6	Need role	primer
7	Header	jdbc:mysql://
8	IP	222.204.211.72
9	Port	3306
10	Tail	/dhuoj?useUnicode=true

the tags <property name=Hibernate.connection.url>. The values of the fields *tag name* and *property* indicate the place where the address should be put, whereas the values of the fields *header* and *tail* are the head and tail of the address, respectively. The field *type* indicates whether the configuration file is an xml file or an ini file.

The commands in the task list are used to start up a component of a distributed application. Table X shows the structure of the commands in the task list file. The two configuration commands in this table will generate two corresponding commands as follows:

cd C:\workspace\ judge_server\judge_server
java -jar judge_server.jar

which are used to start up the component *Schedule*.

TABLE X.
TWO COMMANDS IN THE TASK LIST

Commands	Tag name	Example
Command 1	type	path
	prefix	cd
	value	.\judge_server\judge_server
Command 2	type	string
	prefix	
	value	java –jar judge_server.jar

IV. AUTOMATIC CONNECTION

A. Getting the Local Address

To generate the local address information file, the ACU needs to get the IP address of the local computer. The ACU could obtain the IP address from the information of the network interface card by some method, such as in Fig. 3.

However, the ACU needs to verify whether the obtained IP address is legal before putting it into use. Table XI lists the illegal IP addresses that the ACU uses to verify an obtained IP address. If its format matches one item in the illegal IP address table, the obtained IP address will be discarded.

For a distributed application, every component uses the port number to comminute with others. There are 65535 port numbers in each computer. The ACU needs to get some available port number from the local computer to assign to its target component. Fig. 4 shows an algorithm to get available port number in the local computer. It chooses a port number and tests it. If the chosen number is occupied by other applications, it will choose another port number until the chosen number is not occupied by any application. The ACU uses ServerSocket [11] class to test a TCP port and use DatagramSocket [12] class to test an UDP port.

```
netInterfaces = getNetworkInterfaces();
ni = netInterfaces.nextElement();
ips = ni.getInetAddresses();
ip = ips. getHostIPAddress();
```

Figure 3. The algorithm to find the local IP address

TABLE XI.
ILLEGAL IP ADDRESSES

IP address	Reason
127.0.0.1	This is the loopback IP address and can not be used as the external IP address.
0.0.0.0	This IP address is illegal.
..***	There is no enough segments number.
...***	The value of one IP address segment is null.
..***.256	The value of one IP address segment is not a legal value. For example, the value is less than 0 or greater than 255.

In addition to the IP address and available port number, the ACU will obtain the name and the role of the local component from its task list to write them into the local address information file.

```
Algorithm to get an available port in a computer
    port = ChoiceAPort(8000);
    while(PortAreAvailable(port))
        port = port + 1;
    StorePort(port);
End
```

Figure 4. The algorithm to find an available port number

B. Generating the Address Lookup Table

The address lookup table keeps the addresses of all components in the distributed application. Each ACU needs to update its address lookup table file in time.

The data in the address lookup table come from local address information files of all ACUs in the network. Each ACU sends its local address information to the remote ACUs, so that each ACU could receives the remote address information to fill in its address lookup table. More details of how an ACU builds a connection with a remote ACU to get the address information of a remote component could be found in Section V in this paper.

C. Updating the Task List

Initially, the system operator writes into the task list file all operations that the ACU should do. The executive cell of the ACU will update the IP address and the port number in each operation in the task list file according to the address lookup table.

The executive cell of an ACU modifies the task list file as follows.

(1) Find an unmodified operation *OP* in the task list;
(2) Get the role value *R* in the operation *OP*;
(3) Search for an address entry *A* which contains the role value *R* in the address lookup table;
(4) Update the address in the operation *OP* by the address in the address entry *A*;
(5) Repeat from Step (2) to Step (4) until all operations in the task list are modified.

Fig. 5 shows the algorithm to update the task list file (taskListConfig.xml) by using the address lookup table file (AddressInformation.xml).

```
addressInformation = ReadAddressFile(AddressInformation.xml);
taskList = ReadTaskListFile(taskListConfig.xml);
While
  Task = ReadATask(taskList, TASK_OPERATE);
  Role = GetNeedRoleFromTask(Task, NEEDROLE);
  Require = GetRequire (Task);
  Message = GetNeedMessageFromAddressFile(Role, Require,
              AddressInformation);
  ModifyCorrespondFile(Task, Message);
End;
Exit;
```

Figure 5. The alogrithm to modify task list file

D. Updating the Configuration File

Usually, there is much work needed to do, when the system operator configures a distributed application by hands. To modify a configuration file of the component *Schedule* in the DHUOJ, for example, the system operator needs to find and open the configuration file named *hibernate.cfg.xml* under the directory *.\persistence*, and locate the position of the value that needs updating. Fig. 6 shows the details of the position in the configuration file. The boldface digits between tags <property name="hibernate.connection.url"> are the IP address and port number that the system operator should put.

This configuration operation can be described in the task list (see Table IX). The ACU could use the task list to modify the configuration file of a component in a distributed application. Fig. 7 show the algorithm to modify the configure file (an xml file) of a component in a distributed application. The steps to modify the configuration file are as follows.

(1) Find an unexecuted operation *OP* in the task list;
(2) Get the file type *T* and name *F* of the configuration file from the operation *OP*;
(3) Get the tag name *N*, the attribute *A* and the property *P* from the operation *OP*;
(4) Get the *header*, *IP*, *Port* and *tail* from the operation *OP* and generate a new value *V* by *V* = *header* + *IP* + *Port* + *tail* (The sign "+" means that the corresponding values will be combined.);
(5) Locate the position *Pos* in the configuration file *F* by *N*, *A* and *P*, and update the value at the position *Pos* with *V* by a method for the file type *T*;
(6) Repeat from Step (1) to Step (5) until all operations in the task list are executed.

```
…
<property name="hibernate.connection.url">
jdbc:mysql://222.204.211.72:3306/dhuoj?useUnicode=true&amp;characterEncoding=UTF-8
</property>
…
```

Figure 6. The updated value in the configuration file *hibernate.cfg.xml*

```
Operations = ReadOperationFromTaskListFile(taskListConfig.xml);
While
  Operation = ReadAOperation(Operations, TASK_OPERATE);
  fileType = GetFileType(Operation, FILETYPE);
  filename = GetFileName(Operation, FILE);
  tag = GetTAG(Operation, TAG_NAME);
  property = GetChangeProperty(Operation, CHANGE_PROPERTY);
  attribute = GetChangeAttribute(Operation, CHANGE_ATTRI);
  header = GetHeader(Operation, HEADER);
  ip = GetIP(Operation, IP);
  port = GetPort(Operation, PORT);
  tail = GetTail(Operation, TAIL);
  value = header + ip + port + tail;
  UpdateConfigureFile(tag, property, attribute, value);
End;
```

Figure 7. Algorithm to modify the configuration file

E. Starting up the Component

After finishing the configuration of a distributed application, the system operator needs to start up this application. He needs to initialize the environment and execute the executable program [13][14]. Sometimes, the system operator could use a batch file to make this process more convenient. To start up the component *Schedule* in the DHUOJ, for example, the system operator may create a batch file (such as startup.bat) and add the needed commands to it as in Fig. 8.

There are two commands in this batch file: one is a command to enter into a directory under which the component *Schedule* is located, the other is a command to start up the component *Schedule*. The executive cell of the ACU will also do this operation. It will create a batch file which contains the commands constructed on a basis of the information from the task list (see Table X). The ACU will start up the component *Schedule* with this created batch file.

The method to create a batch file and start up the component is as follows.

(1) Get the file name *startupFileName* that the ACU will use to start up the target component;
(2) Create a batch file named *startupFileName*;
(3) Get the next undealt command *CMD* in the task list in order;
(4) Get the type *T*, the prefix *P* and the value *V* from the command *CMD*;
(5) Generate a command *cmd* by *T*, *P* and *V*;
(6) Append the command *cmd* to the batch file *startupFileName;*
(7) Repeat from Step (3) to Step (6) until all commands in the task list are dealt with;
(8) Start up the target component by executing the batch file *startupFileName*;

Fig. 9 shows the algorithm to start up a component.

```
cd C:\workspace\judge_server\judge_server
java -jar dhuoj_server.jar
```

Figure 8. A file startup.bat

```
fileName = GetFileName(taskListConfig.xml, TASK_FILENAME);
RecreateFile(fileName);
While
  Command =
      GetACommand(taskListConfig.xml, TASK_COMMANDFILE)
  addType = GetAddType(Command, ADDTYPE);
  prefix = GetPrefix(Command, PREFIX);
  value = GetValue(Command, VALUE);
  AddToFile(fileName, addType, prefix, value);
End;
ExecuteFile(fileName);
End;
```

Figure 9. The algorithm to start up a component

V. COMMUNICATION BETWEEN ACUS

To fill the information in the address lookup table, the ACU needs to connect to other ACUs in the network. Basically, most of information is exchanged through the connections [14].

When a new ACU is started up and enters into the network, it will send a searching message with the UDP protocol to the network to find other ACUs that may exist in the network, see Fig. 10.

This message contains the IP address and port number of the sender (see Table XII). If there is an ACU running on the network, it will receive this message and send a responding message back with the TCP protocol, see Fig. 11.

This responding message contains the IP address and port number of the responder (see Table XIII). Thus, either of these two ACUs obtains the address of the other, so that they could comminute with each other, see Fig. 12.

After that, the ACU is in a listening state to wait for the searching messages that come from other ACUs. When receiving a message (searching or responding), the ACU will obtain the IP address, the port number and other information of the remote component and keep these information items in its own address lookup table file.

If an ACU wants to leave the system, it will send a leaving message to other ACUs. Then other ACUs will get this leaving message and delete the address of the leaving component from their address lookup table files.

There are two protocols needed [15][16] for ACUs to exchange their messages: the TCP protocol and the UDP protocol.

The format of the message transferred by UDP is shown in Table XII. The fields *ACU name*, *ACU IP address*, *TCP port number* and *UDP port number* are used to send the address information of the local ACU to remote ACUs, so that the remote ACU could communicate with the local ACU. The fields c*omponent IP*, *component port*, *component role* and *component name* are used to tell remote components the address information of the local component, so that the remote components can communicate with the local component. The field *header* is used to distinguish the message sent by the ACU from the one not sent by the ACU (e.g., other applications may send messages on their own purpose). The field *tail* is used to verify that an UDP message is legal.

TABLE XII.
THE FORMAT OF THE MESSAGE TRANSMITTED BY UDP

Field name	Comment
Header	The head of the UDP message
ACU IP address	The IP address of the ACU
TCP port number	The TCP port number of the ACU
UDP port number	The UDP port number of the ACU
ACU name	The name of the ACU (the sender)
Component IP	The IP address of the component
Component port	The port number of the component
Component role	The role of the component
Component name	The name of the component
Tail	The tail of the UDP message

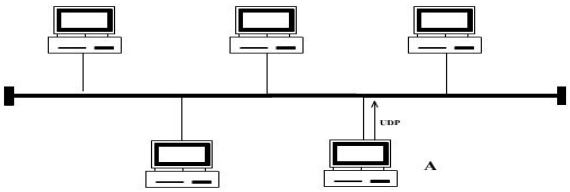

Figure 10. Sending a UDP message to the network

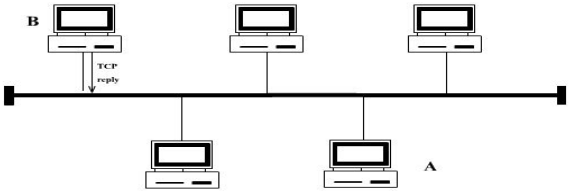

Figure 11. Replying the message

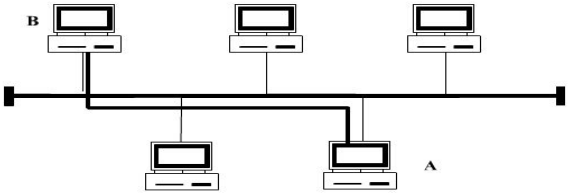

Figure 12. A connection is built.

The format of the message transmitted by TCP protocol is shown in Table XIII . There are three kinds of messages that are transferred by TCP. The field *message type* indicates the type of the message.

Table XIV lists four kinds of messages that the ACU needs to communicate with remote ACUs. The first kind of the message is used to request for connecting to a remote ACU by sending a local address to the remote ACU. The second kind of the message is used to return a local address in response to the remote ACU that sends a request message or a searching message to the responder. The third one is used to tell other ACUs that the sender will leave the system. The last one is used to search for

TABLE XIII.
THE FORMAT OF THE MESSAGE TRANSMITTED BY TCP

Field name	Comment
Message type	The type of this message
ACU IP address	The local IP address of the ACU
TCP port number	The TCP port number of the ACU
UDP port nmuber	The UDP port number of the ACU
ACU name	The name of the ACU (the sender)
Component name	The name of the component
Component role	The role of the component
Component IP	The IP address of the component
Component port	The port number of the component

TABLE XIV.
FOUR MESSAGES EXCHANGED BY ACUS

No	Function	Protocol	Comment
1	Request	TCP	For connecting to a remoter ACU.
2	Respond	TCP	Returning a local address in response.
3	Leave	TCP	A message used when the ACU leaves.
4	Search	UDP	Searching for remote ACUs.

remote ACUs that may exit in the network.

VI. APPLICATION

The ACU has been applied to the campus contest in Donghua University on December 26, 2012. This contest was held in three computer rooms, each of which is equipped with 80 computers for students and one computer for the teacher. The DHUOJ was installed in the forth computer room which is equipped with 40 computers

The method that the ACU builds connections and manages the DHUOJ was verified in the contest. When it is started up, the ACU uses the two algorithms in Fig. 3 and Fig. 4 to obtain the local IP address and an available port number, and write them into its local address information file. The ACU then gets its role from its task list and write it into its local address information file. After that, the ACU broadcasts its local address information to the network and waits for receiving the address information from remote ACUs. If it receives the address information of a remote ACU, the local ACU will keep the received information in its address lookup table as in Fig. 13. Fig. 14 shows that the ACU named *ACU4* has obtained the address information of remote ACUs. If a remote ACU withdraws from the system, the ACU will delete the information of the remote ACU from its address lookup table.

As the system operator click the button *Begin* in Fig. 14, the ACU will update the addresses in the operations in its task list file according to its address lookup table

```
…
<Address>
   <Name>ACU2</Name>
   <Role>Proxy</Role>
   <IP>222.204.211.19</IP>
   <OUT_IP>222.204.211.19</OUT_IP>
   <Port>80</Port>
</Address>
<Address>
   <Name>ACU1</Name>
   <Role>Schedule</Role>
   <IP>222.204.211.60</IP>
   <OUT_IP>222.204.211.19</OUT_IP>
   <Port>80</Port>
</Address>
<Address>
   <Name>ACU3</Name>
   <Role>Primer</Role>
   <IP>222.204.211.74</IP>
…
```

Figure 13. Remote information kept in the address lookup table file

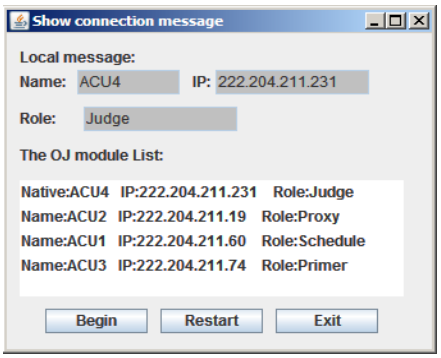

Figure 14. The ACU named *ACU4* has started up.

and will execute the updated operations to modify the configuration file of the component that the ACU works for. After that, the ACU starts up the component to run it. If the system operator wants to restart the component, he could click the button *Restart*.

The local ACU in Fig. 14 is named *ACU4*, which works for the component *Judge*. The three remote ACUs are named *ACU2*, *ACU3* and *ACU4*, which work for the components *Schedule*, *Proxy* and *Primer*, respectively. After the ACU *ACU2* starts up the component *Proxy* successfully, the client could get the service from the component *Proxy*, as shown in Fig. 15. Fig. 16 shows that the ACU *ACU1* has started up the component *Schedule*. The component *Primer* is started up by the ACU *ACU3*. These ACUs managed the DHUOJ successfully during the campus contest.

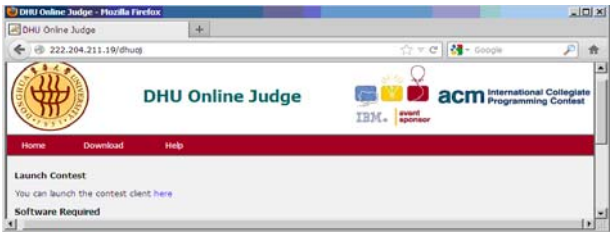

Figure 15. The service interface of the component *Proxy*

```
C:\workspace\judge_server\java –jar judge_server.jar
2012-12-26 12:27:47 org.hibernate.cfg.evironment
Message:Hibernate 3.2.5
2012-12-26 12:27:47 org.hibernate.cfg.evironment
Message:hibernate.properties
```

Figure 16. The information of starting up the component *Schedule*

CONCLUSION

There are many kinds of distributed applications in the world. Some of the distributed applications have many components and their configuration operations are complex. It is unavoidable for the system operator to make mistakes during the complex operations, leading to unpredictable results.

The DHUOJ is such a distributed application that needs many configuration operations, which often result in many mistakes. The system operator needs to take several hours to install the DHUOJ for a contest held in college computer rooms. The computers on which the DHUOJ is installed change frequently and the system operator needs to reinstall the DHUOJ from time to time. Sometimes, the port numbers that the components of the DHUOJ used to communicate with each other were occupied by other programs. This made the configuration operations more complex.

The proposed ACU could do configuration operations for distributed applications automatically. The system operator just needs to set the task list file for one time, and then the ACU could serve for a given distributed application forever without any more configurations regardless where the distributed application would be installed. For example, if the system operator wants to make an ACU to maintain the component *Proxy* in the DHUOJ, he just needs to add corresponding commands

and operations to the task list file of the ACU. After that, the ACU could configure the component *Proxy* and start it up automatically.

The ACU is not just for the DHUOJ. It could be applied to any distributed application if needed. Because all the configuration operations come from its task list file, the ACU can adapt to any situation just by modifying its task list file.

ACKNOWLEDGMENT

This research is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 60973121.

REFERENCES

[1] K. Shvachko, "The Hadoop distributed file system," 2010 IEEE 26th Symposium on *Mass Storage Systems and Technologies (MSST),* pp 1-10, May 2010.

[2] G. Ferrari, R. Guanciale, D. Strollo, E. Tuosto, "Coordination via types in an event-based framework," *Formal Techniques for Networked and Distributed Systems – FORTE 2007*, 2007.

[3] José C. Cunha, Carmen P. Morgado, Jorge F. Custódio, "Group Abstractions for Organizing Dynamic Distributed systems," *Euro - Par 2008 Workshops - Parallel Processing*, 2009.

[4] B. Benatallah, M. Dumas, M.-C. Fauvet, F. Rabhi, "Patterns and skeletons for parallel and distributed computing," *Towards Patterns of Web Services Composition*, pp. 265–296. May 2003.

[5] Elena Verdúa, M. Luisa Reguerasa, J. María Verdúa, P. José Lealb, P. Juan de Castroa, "A distributed system for learning programming on-line," *Computers & Education*,Volume 58, Issue 1, January 2012.

[6] Guojin Zhu, Yang Guan, "Communities of Autonomous connection units for Distributed Online Judge Systems," pressed by the 2010 Second International Conference on Future Computer and Communication, IEEE Transel. Shanghai, China, 2010(9).

[7] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131, ftp://ftp.rfc-editor.org/in-notes/rfc2131.txt, Bucknell Univ., Lewisburg, Pa., Mar. 1997.

[8] Yoram Moses;Michel Raynal, "No Double Discount: Condition-Based Simultaneity Yields Limited Gain. Distributed Computing", 2008.

[9] B. Benatallah, M. Dumas, M.-C. Fauvet, F. Rabhi, "Patterns and skeletons for parallel and distributed computing," *Towards Patterns of Web Services Composition*, pp. 265–296. Springer, Heidelberg (2003).

[10] Damien Imbs, Michel Raynal, "A Lock-Based STM Protocol That Satisfies Opacity and Progressiveness," *Principles of Distributed systems*, 2008.

[11] M. Eid, A. Alamri, A.E. Saddik, "A reference model for dynamic Web service composition systems," Int.J.Web Grid Serv, 2008.

[12] D. Kandlur Dilip, Debanjan Saha, M. Willebeek-LeMair, "Protocol architecture for multimedia applications over ATM networks," *ACM SIGCOMM Computer Communication*, 1995, Volume 25 Issue 3, Pages 33 – 43.

[13] D. Skeen, M. Stonebraker, "A Formal Model of Crash Recovery in a Distributed System," *Software Engineering, IEEE Transactions on*, May 2009.

[14] M. Frank, D. Basin, J.M. Buhmann, "A Class of Probabilistic Models for Role Workering," Proc.15th *ACM Conf. Computer and Communications Security*, 2008.

[15] Andreas Grau, Klaus Herrmann, Kurt Rothermel, "Scalable Network Emulation - The NET Approach," *Journal of Networks*, 2012, Vol 7:3-16

[16] Hongzhen Xu, Guosun Zeng, Bo Chen, "Description and Verification of Dynamic Software Architectures for Distributed Systems," *Journal of Networks*, 2010, Vol 5: 721-728.

Guojin Zhu is an associate professor at the Department of Computer Science, Donghua University (DHU), Shanghai, China. He received his M.S. and Ph.D. degrees from DHU in 1991 and 2007, respectively. He was a visiting scholar at the Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, USA from November 2007 to November 2008. His current research interests include semantic web, knowledge discovery, and neural computing.

Yongjiang Zhou is a graduate student of Computer Application Technology at Donghua University. He was born in Shandong province, P. R. China in 1987, and received the bachelor degree of computer science and technology in Qufu normal University in 2010. His current main research interest is computer network and AI.