I. Introduction

The problem of implementing load-balancing distributed systems has been studied extensively. And based on different kinds of architectures in distributed systems, there are many different methods to solve load-balancing problem.

However, no matter what kinds of environments, load distribution and balancing are primary functions for improving system performance and users' comfort. Incoming tasks' allocation and remote process execution are main consideration of a well designed distributed systems in order to achieve performance improvements.

For the purpose of automatically selecting remote execution hosts in a distributed environment, this paper proposed a user-supervised processor allocation scheduler. It shows how load-balancing is provided in this model and how to collect information and disseminate it to support the user decision to select remote execution hosts.

1.1 Motivation and background

With the progress of computer technology and communication service, the traditional central processing work is gradually replaced by distributed systems' architecture. This phenomenon is arisen by dropping price of high-performance PC and dramatically improvement of manufacturing process in microprocessors. According to the past scholars said, the updating speed of microprocessors is 50% to 100% per year, and the memory or storage space is increased four times each three years. And this trend speeds up these few years.

In the past, high-speeding computers are used for computing complex problem or processing huge data. So the scientists are aimed to develop non time-consuming computers. And then the massively parallel processors computers were produced. The concepts include many single-packed chip CPUs, high-speed bus, complex cached, and cheap RAMs. And this 'big' machine serves all requests and computing works. As time goes by, because

- The quality and bandwidth of network are improved.
- The computing power of Workstation is strengthened.
- The bottleneck of I/O is gradually improved.

Network of Workstations（NOWs）,which integrates huge memory, storage space, and processors, arise in these few years. Its concepts include a group of connected computers and grouping their computing ability to solve single complex problem. Even more, some NOWs will dynamically distribute works according to their loading status to save the computing time and system resources, and some also dynamically re-distribute works when some machines fail. It indicates how to implement the distributed systems.

Besides price and stability, the advantages of distributed computing also includes economics, reliability and scalability. Dandamudi[1995] pointed out the profits of distributed systems ： They are effectively using resource, better in price/performance ratio, speed and scalability. Many interests of distributed systems are its load-distribution characteristics. Using different kinds of load distribution strategies will affect the performance of distributed systems. Livny[1982] pointed out that in a distributed system, users can share each others' resources and they are different in their equipments. Therefore some machines may always be idle and some may have many works staying in the queues. So how to solve the loading distribution problem is what they concern a lot.

Dandamudi pointed out that there are three methods used for load-balancing of distributed systems. They are stable, dynamic, and adaptive. Stable method is to decide the works allocation before works come in. Dynamic method is to use the real-time machines' information to decide the work allocation. Adaptive method is the hybrid of the above two, and it can change the allocation strategy based on parameters.

Dynamic loading-balance strategy is what we concern and try to give the optimal solution to balance the overhead and the real works. And how to gather the real-time information which supports load distribution is quite important.

Traditional loading-balance algorithms are specified to homogeneous environments, that is, the same OS and same languages used in each distributed host. It seems not flexible and extensible for scalable systems. And since the concept of "Object" grows, different hosts can call same "object" to be implemented via different languages, as long as this object is implemented in at least one host. This is the basic

idea of “CORBA”.

Within the above ideas, we tried to combine CORBA’s characteristics of cross-platform and a real-time information reporting subsystem to see if the overall performance is better than other work scheduling algorithms.

1.2 **Purpose and Contribution**

Enhancing CORBA’s load-balancing while maintaining CORBA’s transparency and simplicity of application programming is a challenge. Some previous study or software provided a third party directory service or naming service to handle this problem, such as smart agent published by Visibroker. Each registered object has its own name in smart agent’s repository. And once a client issues an object, the smart agent automatically passed the object’s reference to one of the implemented objects. This object can be implemented in any host and will return the result of issues. Here the smart agent works as a directory service to provide objects’ reference to client and what more, it provides some simple load-balancing algorithm. In Visibroker and other ORB publishers, this directory service used “Round-Robin” algorithm for load-balancing.
But it will cause the following problems：

- Since “Round-Robin” algorithm only provides works to be distributed to different hosts based on their arriving time, some hosts may be high-loading at issued time. So the coming work will be put in its queue or just make the loading status heavier. And some hosts may be still idle. This problem is further discussed in Andrea C. Duseau et al. , who proposed a “local scheduler” for interactive work scheduling in distributed system. They said some interactive work will be experienced excessive delay by users when using round-robin scheduling dominantly.
- We said that “distributed systems” should have no central processing work. But if we assign a machine to provide directory service, all invocations will first serve in this machine. It may cause some bottleneck problem. Directory naming lookup operation is a significant factor in system performance.（Leffler et al.,1984）And there are also some papers proposed to solve this problem, such

as David R. Cheriton et al.. They proposed “three-level” directory and client-side cached technology to alleviate central directory service.

Our design purpose is aimed to solve the problem. By using real-time reporting information and user-supervised processor allocation scheduler, no central processing work will be involved and can exactly achieve “loading-balance”.

The Common Object Request Broker Architecture（CORBA）supports applications that consist of objects distributed across a system, with client objects invoking server objects that return responses to the client objects after performing the requested operations. CORBA’s Obejct Request Broker（ORB）acts as an intermediary in the communication between a client object and a server objects, transcending differences in their programming language （ language transparency ） and their physical locations （ location transparency）.CORBA’s TCP/IP-based Internet Inter-ORB Protocol（IIOP） allows a client object and a server object to communicate, regardless of differences in their respective operating systems, byte orders and hardware architectures.

1.3 Research Method

1.3.1 Define research direction

We define the whole work is based on “heterogeneous” and connected distributed environment.

1.3.2 Related works and survey

Distributed environment models and information collection and publishing methods. And the previous study on “loading –balance” of distributed system.

1.3.3 Problem Definition

What we want to solve is loading-balance in each host of distributed system to improve the efficiency. And the policy of collecting so-called “real-time” information but not increase the overhead of executing works.

1.3.4 Design the blueprint

Design the algorithm to solve the above problem. Analyze the algorithm step by

step to split every function should be performed.

Point out which subsystems should perform which functions and assemble them together to form the model. Do some scenario testing works for this model.

1.3.5 Assumption and restriction

Before implementation stage, we should make some assumption and restriction to make the implementation more specified and clearly.

1.3.6 Implementation

Make the design to be implemented by codes.

1.3.7 Analysis and conclusion

Doing some simulation work based on assumption and analyze the results.

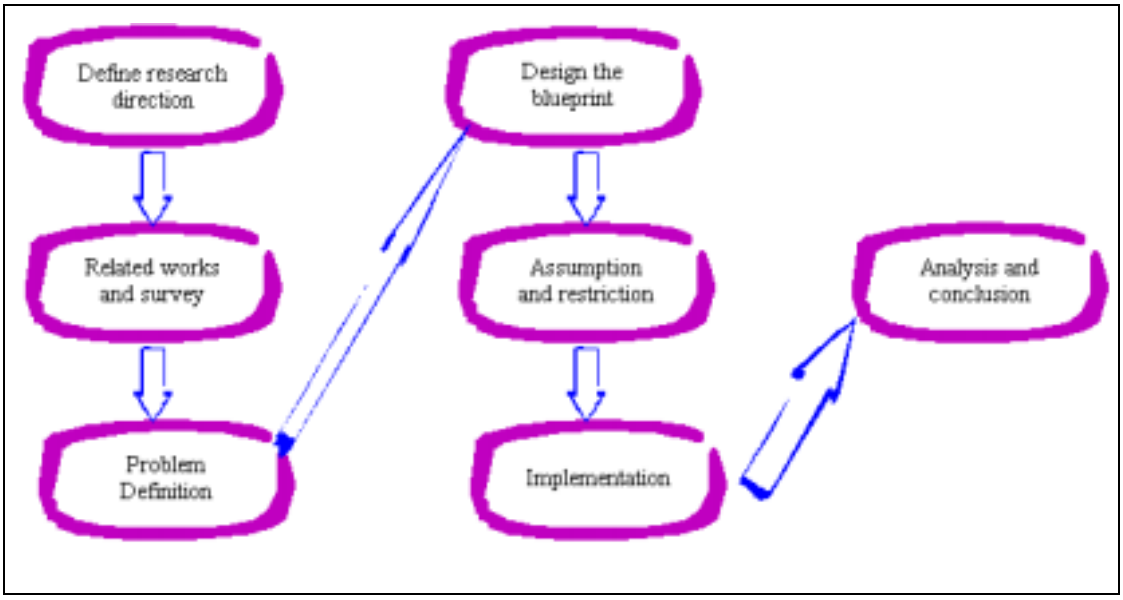

Figure. 1 my research method flow

II. Related Work

2.1 CORBA Model

The Common Object Request Broker Architecture (CORBA) allows distributed applications to interoperate (application to application communication), regardless of what language they are written in or where these applications reside.

The CORBA specification was adopted by the Object Management Group to address the complexity and high cost of developing distributed object applications. CORBA uses an object-oriented approach for creating software components that can be reused and shared between applications. Each object encapsulates the details of its inner workings and presents a well defined interface, which reduces application complexity. The cost of developing applications is reduced, because once an object is implemented and tested, it can be used over and over again.

The Object Request Broker (ORB) in figure 2. connects a client application with the objects it wants to use. The client program does not need to know whether the object implementation it is in communication with resides on the same computer or is located on a remote computer somewhere on the network. The client program only needs to know the object's name and understand how to use the object's interface. The ORB takes care of the details of locating the object, routing the request, and returning the result.

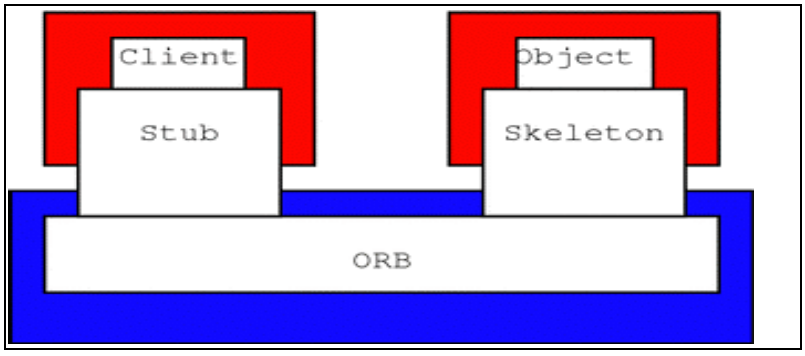

Figure 2. ORB Communication

ORB is the base of CORBA applications, and different venders developed different ORBs. The ORB itself is not a separate process. It is a collection of libraries and network resources that integrates within end-user. applications, and allows your client applications

to locate and use objects.

- There are a couple of things to note about CORBA architecture and its computing model: A CORBA-based system is a collection of objects that isolates the requesters of services (clients) from the providers of services (servers) by a well-defined encapsulating interface.
- All requests are managed by the ORB. In other words, every invocation (whether it is local or remote) of a CORBA object is passed to an ORB, as shown in Figure 2 above.

The interface between the client and server is simply a contract that specifies what kind of services are provided by the server and how they can be used by the client. CORBA interfaces are specified in a special definition language known as the Interface Definition Language (IDL). Through IDL, a particular object implementation tells its potential clients what operations are available and how they should be invoked.

From IDL definitions, CORBA objects are mapped into different programming languages like C, C++, Java, and Smalltalk. This means once you define an interface to objects in IDL, you are free to implement the object using any suitable programming language that has IDL mapping. Consequently, if you want to use that object, you can use any programming language to make remote requests to the object.

2.2 The steps of creating CORBA Applications

Since we address the universal usable characteristics of CORBA, we should create our application through CORBA procedures. Shown as Figure 3.

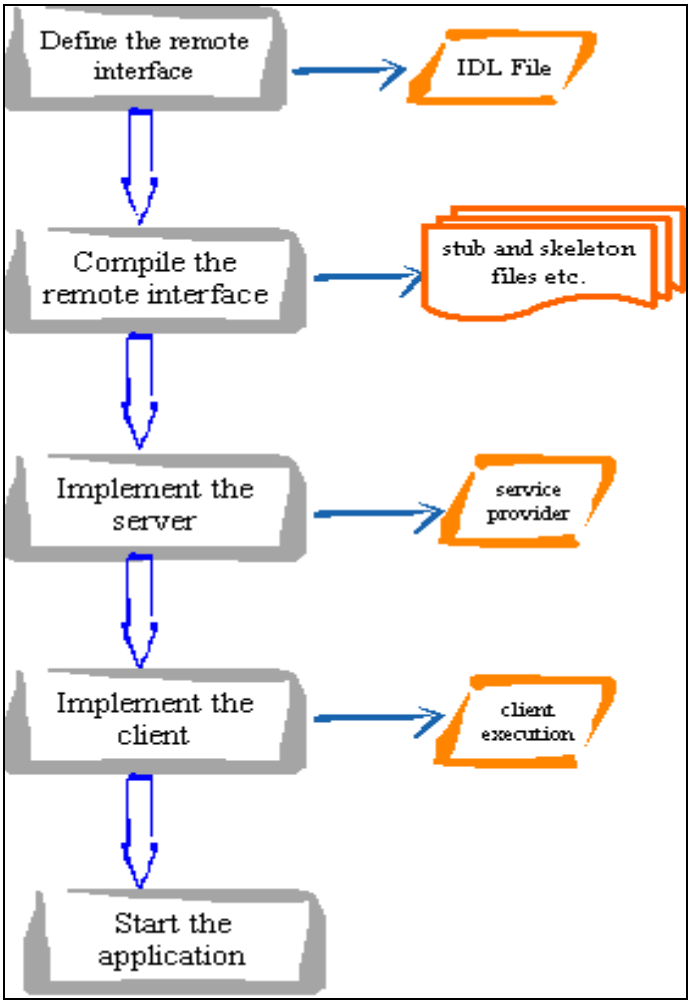

Figure 3. The development life cycle of CORBA-based systems

2.2.1 Define the remote interface

Define the interface for the remote object using the OMG's interface definition language. Using IDL instead of the specified language because the IDL compiler（based on different ORBs and different language mapping we will use）automatically maps from IDL, generating all specified language stub and skeleton source files, along with the infrastructure code for connecting to the ORB. Also, by using IDL, it’s possible for developers to implement clients and servers in any other CORBA-compliant language.

2.2.2 Compile the remote interface

When we run the ORB-specified and language-specified compiler over IDL files , it generates the language-specified version of the interface, as well as the class code files for the stubs and skeletons that enable your applications to hook into the ORB.

2.2.3 Implement the server

Once we run the IDL compiler, use the skeletons it generates to put together the server application. In addition to implementing the methods of the remote interface, the server code includes a mechanism to start the ORB and wait for invocation from a remote client.

2.2.4 Implement the client

Use the stubs generated by the IDL compiler as the basis of the client application. The client code builds on the stubs to start its ORB, look up the server using the URL Naming service provided by its importing library , obtain a reference for the remote object, and call its method.

2.2.5 Start the applications

Once implement a server and a client, start the server to hook the object IOR with run-time defined URL string, then run the client.

2.3 Loading balance in distributed system

Here we review some proposed methods to distributing work loading in previous work. In the distributed computing system, the user can use the other machine to execute works. This is the benefit of distributed systems, but may become a defect. That's because, if one working node's is full-loading, and the user still use this node to execute other works, it will affect the performance of former work and the latency will become too long. Evan more, if the other nodes are waiting the executing result to go on, the whole execution time will delay unlimitedly.

Siegel[1996] proposed that to complete the distributed computing should consider these following factorials：1.The transferring time of data in different machines. 2. The cost of transferring work. 3. The parallel processing ability in a specified application. 4. The network loading status.

Also, Zaki[1997] mentioned that the probable factorials which affects the distributed system, includes：

- **data formats**：For different language mapping, the data format must have an agreement on their data format.
- **processors' features** ：Some CPUs are designed for different needs. Some are suitable for floating-operations, and some are suitable for processing vector data.
- **External loading**：In distributed system, besides the invocation method which we concern, the other usage of machine will affect the performance, especially when the external loading is huge.

2.3.1 The loading balance method

The loading balance methods can roughly divide into three kinds：static、dynamic and adaptive.

- Static loading balance：Static loading balance uses some simple system information, such as average processing time and cycle. According to the information and some adjusting formulas to assign the work on different machines. The benefit of this method is simple and doesn't need gather the information too often. But it will cause some low-usage machines, because of not adjusting system's efficacy dynamically.
- Dynamic loading balance：Dynamic loading balance uses the up-to-now system status as parameters to allocate works on different nodes. But the defect of this method is obviously, it will add the additional overhead. But if we can control the overhead in acceptable range, the performance will be much better than static one. Also, there are two dynamic loading balance method：central control v.s. distributed control. Central control is using a specified node to summarize the loading information and decide how to allocate works. Distributed control rests on invocators' need whether high-loading node transfer the work to the light-loading nodes（Sender-initiated）, or light-loading node automatically find the high-loading nodes to share their works（Receiver-initiated）.
- Adaptive loading balance：According to the circumstances, Shivaratri[1992] used work transferring and nodes' loading status. When the system is in

high-loading state, he used Sender-initiated strategy. When the system is in light-loading state, he used Receiver-initiated strategy.

In our design of work, we select “dynamic loading balance” for its dynamic characteristics. Because we restrict the work to be “none migrating task”, all task invocation is initiated by requesting one. No transferring of work will occur and therefore we don’t use “Adaptive loading balance”.

2.4 Gathering real-time information of each node

Since we select “Dynamic loading balance “ method. The key point is how to gather the real-time information to be the parameters for selecting executing node. One concern is not to increase the overhead of whole systems. Oltson et al. [19] proposed an algorithm used to approximate the real time information. The basic idea of it is that not all applications on the network need so precise value of information. Updating and transferring real-time information too frequently will cause to increase the overhead and decrease the performance. Oltson et al. claimed that the information could be reported as an “interval”, and the target is to lower down the cost. Oltson made a formulas to set the width of this interval.

2.4.1. The mechanism of interval shrinks

Since the information of real time data is changed by time, it is necessary to “shrink” the value interval. Two mechanisms are used to adjust the interval. One is “value-initiated” refresh, and the other is “query-initiated” refresh. “Value-initiated” refresh is issued by sender, which gathering each updating data(In this paper, “cache” the updating data). As long as the new updating data went out the value interval, this initial interval should shrink to cover the new updating data. Most of the time, this shrink will expand the interval width. “Query-initiated” refresh is issued by the receiver, who use this data for application use. When the receiver thought this data is not “precise” enough, it can issue the sender to “shrink” the interval. This kind of shrink makes the interval “narrow”.

2.4.2 The model and algorithm symbols

Table 1 Model and algorithm symbols

Symbol	Meaning	Note
Cvr	Cost of a value-initiated refresh	Used to determine cost factor ρ
Cqr	Cost of a query-initiated refresh	Used to determine cost factor ρ
ρ	Cost factor defined as 2* *Cvr/ Cqr*	Determines width adjustment probability
Ω	Cost rate (per time step)	Metric this algorithm minimizes
W	Width of a cached approximation	Set adaptively by our algorithm
*W**	Width that minimizes the cost Ω	This algorithm converges to *W* = *W**
Pvr	Probability of value-initiated refresh	Increases with precision
Pqr	Probability of query-initiated refresh	Decreases with precision

For a given cached approximation with width *W*, the probability of each type of refresh can be written as $Pvr = K_1 / W^2$ and $Pqr = K2 * W$, where *K1* and *K2* are model parameters that depend on the nature of data and updates, the frequency of queries, and the distribution of query precision requirements. Now we know how *Pvr* and *Pqr* depend on W, we can rewrite our cost rate Ω in terms of W：

$$\Omega(W) = C_{vr} \cdot K_1 / W^2 + C_{qr} \cdot K2 \cdot W$$

Our goal is to find the minimum value of Ω(W). Therefore we different this function to get the optimal value of Ω. The optimal value for *W* is

$$W^* = (2 \cdot C_{vr}/C_{qr} \cdot K1/K2)^{(1/3)} = (\rho \cdot K1/K2)^{(1/3)}$$

Setting the interval width *W* based on this formula for *W** is difficult unless update behavior and query/update workloads are stable and known in advance, since *K1* and *K2* depend on these factors. This approach is not to monitor these factors at run-time to set *K1* and *K2* but using other adjusting algorithm.

2.4.3 The experiments and derived results

Oltson et al.[19] thought that when querying to “shrink”. The cost of query is 2 times of value-initiated “shrink”, because it needs sending back the new value of interval. Based

on this assumption, $\rho = 1$. In this special case, the optimal values for W occurs exactly when two types of refreshed are equally likely, showed in Fig 4

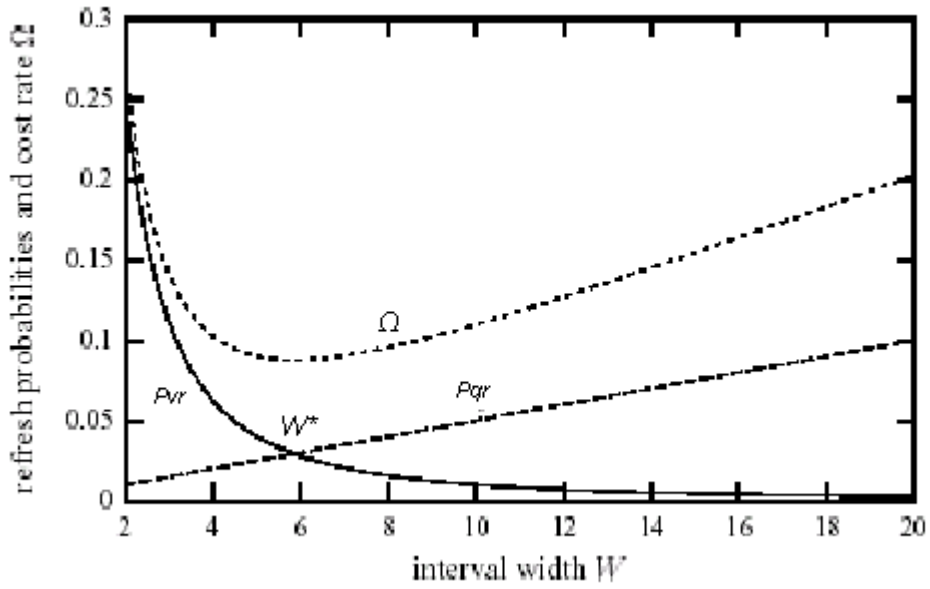

Figure 4 : Cost rate and refresh probability when $\rho = 1$ as functions of interval width

From this experiment, we found that when $P_{vr} = P_{qr}$,the optimal Ω is found. Therefore, we can compute K1 and K2 to find the optimal W^*. In the experiment, $K_1 = 1$ and $K_2 = 1/200$. W* is computed as 5.85. Noted that this experiment is based on a query period of 10 seconds, and the interval width is a relative value.

We will use this algorithm and experiment result to solve the precise problem in our updating information works later.

2.5 Synchronize the clock in each node

Since in our model, we should gather real-time information in each node. It is important to judge whether the information is out-of-date. Some synchronize works should be taken into consideration. Here we introduced Cheng Liao et al.[20] ‘s Global synchronizing clock algorithm to make the clock on each node be synchronized.

In Cheng Liao et al. ‘s algorithm, there exists an “Node 0”, who is responsible for sending global time and maintaining other nodes’ time to be precise enough. We summarized this algorithm in some steps：

At Node 0 as clock server :

1. Periodically sending its time to other nodes (we use X to mention a specified node)according to their resynchronized period（each node has different period）
2. Receiving X's time and compute its time difference between itself. （T*diff*）
3. Finding out which causes made T*diff* exceed the threshold（Network latency or Node X needs to be re-synchronized）
4. If the latency exceeds the threshold, sending again Node 0's time and receiving Node X's time to update the latency.
5. Sending back Node X its Time difference between Node 0
6. Listening from other nodes error message which contain the time difference exceeding its threshold.
7. If X has errors exceed threshold, Node 0 should narrow the re-synchronized time period.
8. Otherwise if Node 0 has not received any error massage from X for a long time, Node 0 should expand its re-synchronized time period.

At Node X（Which is not the time server）：

1. Receiving re-synchronized message periodically and sending back its time.
2. Receiving from Node 0 the time difference between itself.
3. If this difference exceed its threshold, sending the error message to Node 0 .
4. Calculating its drift rate to self-resynchronize
5. Applying drift rate to T*diff* every re-synchronizing period

After some experiments, this algorithm showed less error （each clock of nodes is "precise"）and more efficiency（system loading alleviate）.

2.6 WSDL

2.6.1 WSDL（Web Services Description Language）

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete

endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.

In this paper, we made the host's status to be published in its WSDL file. So, each requesting information node can get return messages via its defined WSDL file, no matter what language and platform used.

There are several main components should be defined in a WSDL file.

- Types

  Types are just wrappers for XML Schema definitions. The purpose of the types is to enable one to create all the XML definitions that will be referenced by messages when building up an actual protocol communication.

- Message

  Messages are defined at the global level and can be re-used. A message consists of parts. A part is just a pointer to a piece of XML Schema. WSDL currently supports referencing XML schema element declarations as well as simpleType and complexType declarations.

- Operation

The next level of object is the operation. An operation is how one expresses calling patterns. Each operation XML element can contain an input and output XML element as well as zero or more fault XML elements.

- portType

Operations XML elements are gathered together inside of a portType XML element.

- Transferring elements

The intentional goal of WSDL is to enable programs to be written without having to think about the actual protocols being used to move messages around. One can write a program expecting a particular portType and not care a whit about what is happening on the wire. These left two parts are doing such works :

- Binding

To actually figure out how to move a particular portType around one has to declare a WSDL binding. A binding provides instructions on how to encode and transport the operations and messages defined in a portType.

- Service

The last step, associating an address with a binding is carried out by a service. A service is a collection of ports. A port is simply an association of an address with a binding. This means that all the methods (operations) for a particular interface (portType) are addressed using the same URI..

2.7 Load distribution and balancing support in a workstation based distributed system

Arrrdondo D. et al.[18] proposed a loading-balance mechanism to equally distribute work in homogeneous distributed system. Their design is an user-supervised processor allocation scheduler, shows which information should be
collected and when and how to collect and disseminate it to support the user decision.

2.7.1 THE LOAD BALANCING SYSTEM (LBS)

The *load balancing system* should supply an information subsystem, a decision subsystem and an execution subsystem (See Fig. 5).

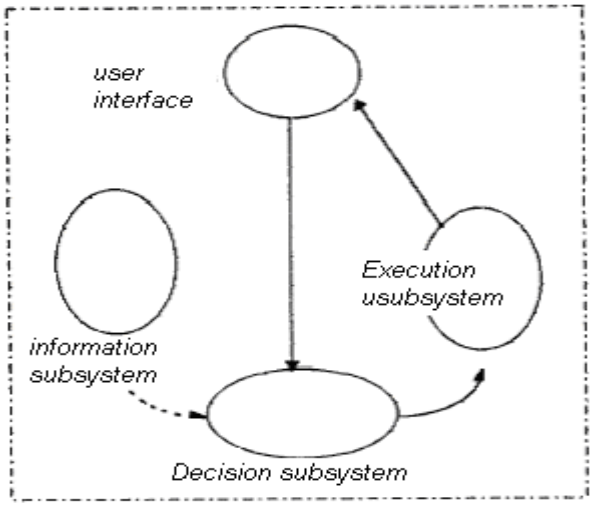

Figure 5 Load Balancing System

The *information subsystem* is in charge of collecting and maintaining global information about each available node in the network and its load condition.

The *decision subsystem,* by using the output of the information subsystem, is responsible to evaluate which is the most suitable node for the execution requested by the user.

The *execution subsystem* is in charge of executing the user requested program on the workstation designated by the decision subsystem.

The *user interface* is restricted to receive execution requests and to return results.

2.7.2 Performance metrics selection

Many performance parameters could be considered to decide what to collect when seeking for knowledge of system load. Conventional approaches use as a referential metric the number of processes waiting, locally, for their CPUs (ready queue length, *rql).*

On deciding which resources and metrics should be included, issues such as *reliable representation* of load condition, *computing overhead* and *distributed control* were extensively considered. we defined *mrm* consisting of a representative group of mettles related to basic resources, as follows:

$$mrm = < CAPP, CMC, DT, NT > \text{ where}$$

• *CAPP,* current available processing power in the node; is a ratio between the manufacturer declared processing speed (MIPS) and the number of processes currently serviced by the local CPU.

• *CMC,* current memory capacity, is the free memory size in the node.

• *DT,* disk transfers, is the amount of data actually transferred on each disk (internal I/O traffic).

• *NT,* network transfers, is the number of I/O packets on each network interface (external I/O traffic).

The decision based on this criteria is made by decision subsystem. It bases on what application user will request to evaluate the best execution node. It needs some historical data to support the decision process.

2.7.3 Data gathering strategies

Aiming to provide recently updated information on *system load condition* we proposed a *periodic-and-event-based* strategy. The basic actions within this strategy are:

• Every certain prefixed interval, a timer interrupt occurs and, each node broadcasts 5 its own state *(mrm)* to the rest of the nodes. Individual intervals are shifted along the global time to reduce collision rate.

•Each time a job arrives to the system, the entry node checks if every item in MRM was updated within that prefixed interval (ageing control procedure). If some node update is beyond this threshold then a request, for updated *mrm* values, will be sent to that node.

2.7.4 The assumptions and scenario of this design

In this design , there are some restrictions as following：

• Processes to be executed are non migrating processes. Once a process is initiated in a machine it remains there until completion (it is not allowed to move the process to another site).

• Processes created by remote tasks execute in the same machine as their parents.

• The network nodes are homogeneous in hardware and operating system.

• Nodes in the network are arranged as a computing pool. Many users, with equal privileges, can be logged to a single node and nobody is the workstation owner.

Briefly describe the over-all process of one request,

1. User request a remote execution
2. Its load balancing system will automatically assign which host to execute based on what application and the information it gathered.
3. Successfully request an application and get the return.
4. Piggybacking each other’s information.
5. The periodic gathering is embedded, as a Daemon.

2.8 Loading balancing in Visibroker’s smart agent

VisiBroker Edition's Smart Agent (osagent) is a dynamic, distributed directory service that provides facilities for both client applications and object implementations. The Smart Agent keeps track of objects that are available on a network, and locates objects for client applications at invocation time. The Smart Agent locates the specified implementation so that a connection can be established between the client and the implementation. The Smart Agent is designed to be a lightweight directory service with a flat, simple namespace, which can support a small number of well known objects within a local network.

2.8.1 The loading balance and scalability in smart agent

1. The Smart Agent implements load balancing using a simple round-robin algorithm on a per agent basis, not on an ORB domain basis. For load balancing between server replicas, when we have more than one Smart Agent in the ORB domain, make sure all servers are registered with the same Smart Agent.
2. When a Smart Agent is terminated, all servers that were registered with that agent attempt to locate another agent with which to register. This process is automatic, but may take up to two minutes for the server to perform this function. During that two minute window, the server is not registered in the ORB domain and therefore is not available to new clients.
3. Server registrations should be limited to less than 100 object instances or POAs per ORB domain.
4. The Smart Agent keeps track of all clients (not just CORBA servers), so every client creates a small load on the Smart Agent. Within any 10 minute period, the client population should generally not exceed 100 clients.

2.9 Comparing Loading balancing algorithm between each other

From the simply review in Section 2.8 and 2.9, we can get some ideas of different kinds of loading balance strategy. Our design in this paper is aimed to solve the

"cross-platform" problem in Section 2.8 and the "scalability" problem in Section 2.9. See Table 2

In table 2, we can see what the main difference between each other. Cross-platform means the whole system requires each node with the same platform and executing language or not. Central work means if this method need some "central processing" work to allocate the tasks. Distributing strategy is according to Zaki[1997] to label them. Others are some description that is important to affect our results.

Table 2 The comparison of each models

charact. / methods	Cross-platform	Central work	Distributing strategy	Others
Arrrdondo D. et al.	No	No	Dynamically, using nodes' loading information to allocate work	1. No mention about how to gathering "dynamical" data of each node 2. Emphasizing on evolutionally setting the criteria of decision making.
Visibroker's smart agnet	Yes	In smart agent to store each node's information	Dynamic, Simple Round-Robin	1. It takes time to make different smart agents to communicate 2. Smart agent should be initiate earlier than registering objects' interfaces
Our Design	Yes	Only Synchronized work	Dynamically, using approximate real-time information to allocate work	1. Using simplified criteria of selecting execution host. 2. Considering the

				real-time information gathering and synchronized problem

III. Model of study

3.1 Define the problem

What we want to emphasize is simple question：How can we dynamically assign which hosts to execute specified tasks？In the published ORB , this problem is solved by some “central processing” work. Visibroker’s smart agent is an example for it. Every host registers its own objects to this agent by object’s name. When one invocates the object, it first gets the object IOR through this agent. The smart agent will use “Round-Robin” algorithm to “dynamically” assign object’s IOR on different hosts to the invocater.

3.1.1 Main Problem

To solve this “central” and “not so dynamically” allocation policy, we would like to propose a new algorithm for this problem. The main purpose is：

How to alleviate the loading of whole system and therefore improve the performance without third party’s work？

We use the dynamic loading method (Shivaratri[1992])to solve loading-balance problem. Dynamic loading method, which fastest reflects the instant system information, can be used to allocate works more appropriately. But here comes another problem：How to dynamically assign tasks？What kind of information is necessary to decide the executing node？

3.1.2 Data Gathering problem

We use resource information based on what application the clients request. Just as Kunz ‘s[1991] claim, different applications will affect different resource usage. So we

used a simple application to choose the resource threshold, that is, big number factorial computing.

Big number computing operation is most effected by CPU's usage（ratio）and RAM available. It uses dynamic information. To simplify the information gathering work, we use only CPU usage as the parameter o choose the executing host.

3.1.3 Dynamic information

Since we decide to gather the "CPU Usage" as the base to select the executing host, we should gather this data "up-to-date". CPU usage is very dynamical information and hard to be updated and transmitting too often. We use the algorithm Olston et al. proposed---the interval, to approximate the value.

3.1.4 Synchronizing work

The information we gather should be "up-to-date". So we need a universal clock for judge if the data is out-of-date. Here we use Arrrdondo D. et al.'s algorithm. This algorithm will be less-cost.

3.2The assumption and solution to simulation restriction

To focus on the problem we want to improve. Some assumptions are made below

- The computation work is done within single host and won't be transferred to other hosts. This work is known as non migrating process.
- Every host has ways to gathering their system information, no matter what OS or language it uses.
- The interval width data is computed in advanced and could be known to each node.
- The Clock server is stable and the re-synchronizing work is done implicitly steady. All hosts can self-resynchronize by itself for a long period.

To simulate heterogeneous environment within a single machine. We introduced a software,"VMWare". It is used to run more than one operating systems at one single machine. By using this software, we can simulate at least 2 virtual machines in one real

machine.

3.3 The information gathering algorithm

We made the information gathering work as an embed process. And it also contains the clock synchronizing work. The steps are as followings :

1. Joining the computing network, first synchronize the clock with the clock server.
2. Setting the time interval of each updating and network latency of one transmission .
3. Getting its own information currently and caching them to compute the average in each time interval.
4. Periodically sending out its “interval” information of status.
5. Listening other nodes’ messages and judging whether the data is out-of-date or accepted.
6. Storing the “approximate” real-time data of other nodes in its local file for further selecting execution node work.
7. If the message other nodes sent cannot be accepted, it can automatically request a message about others status information.
8. Go back to step 3

3.4The scenario and algorithm of one invocation

The steps of one invocation are as following,

1. Looking up the file which stores status information of each node
2. First selecting one most suitable host to execute its invocation.
3. Invocating to this most suitable host with piggybacking its own status information

4. The executing host returns the result and piggybacking its own status information.

5. Both nodes could look up its status file to update this file.

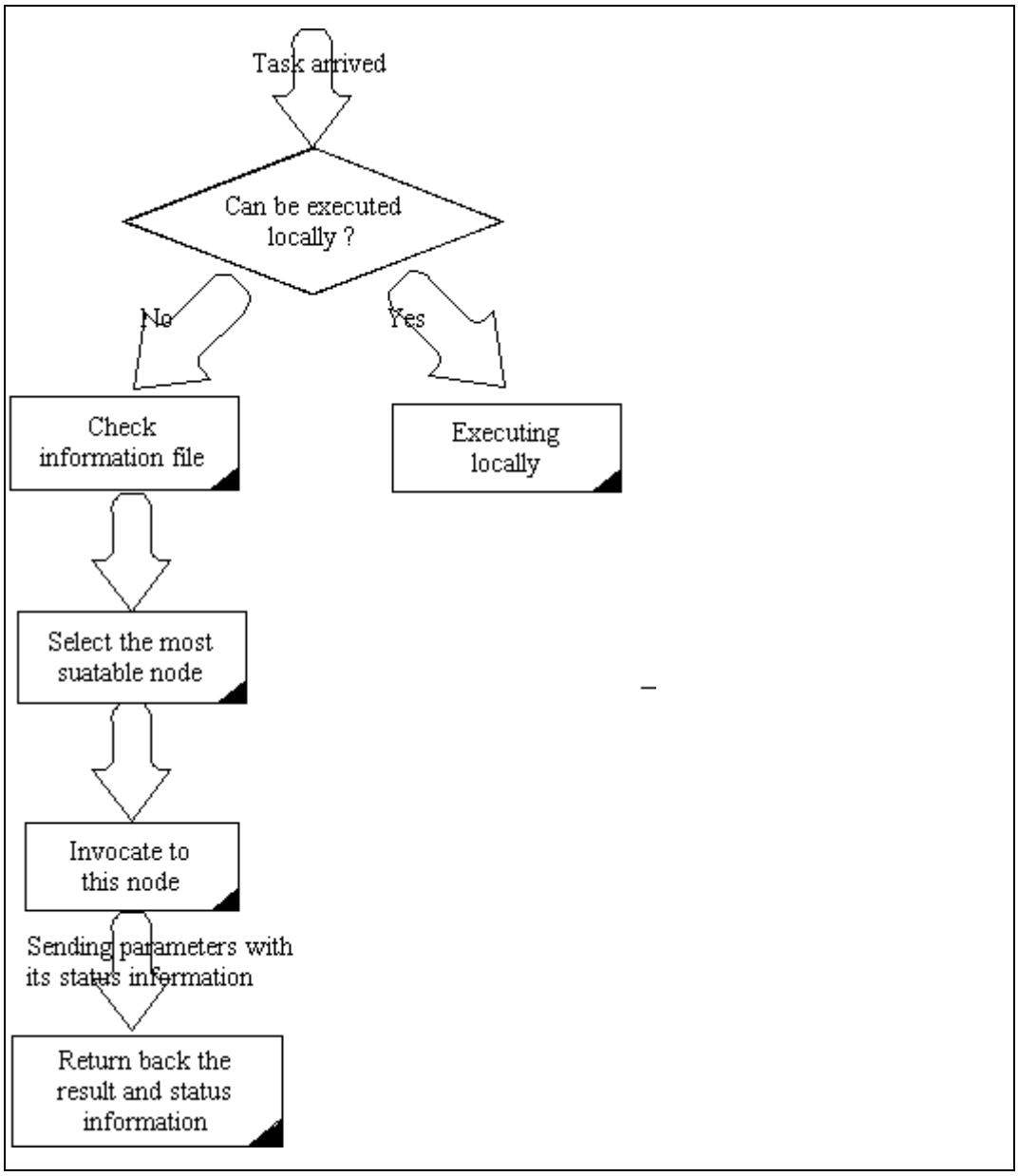

Figure 6 The flow of one invocation

3.5Main idea of this architecture

In this flow, each host in the serving distributed environment will update their information of each node periodically and check its validity. Therefore it can invocate the method to known suitable host.（Here we use URL Naming）After the execution,

both sides get the updating information about each other. Every time when it invocates other services or the same service, it can use this information to invocate light-loading machines, like machine learning.

3.6 Splitting functions to subsystems

If we make whole system implemented under CORBA architecture, it may cause the clients' store too many operation stub codes. And it's hard to define the status information data type（CORBA uses **Any** for unknown data source type）. So, we combined the WSDL concept into this work. CORBA is used to execute invocations and transferring whole information file（which stored as text file and can be transferring as string）. And a host's WSDL service about its own status information is published on the same URL（but using different file name）.See fig 7 for detail functions and subsystems.

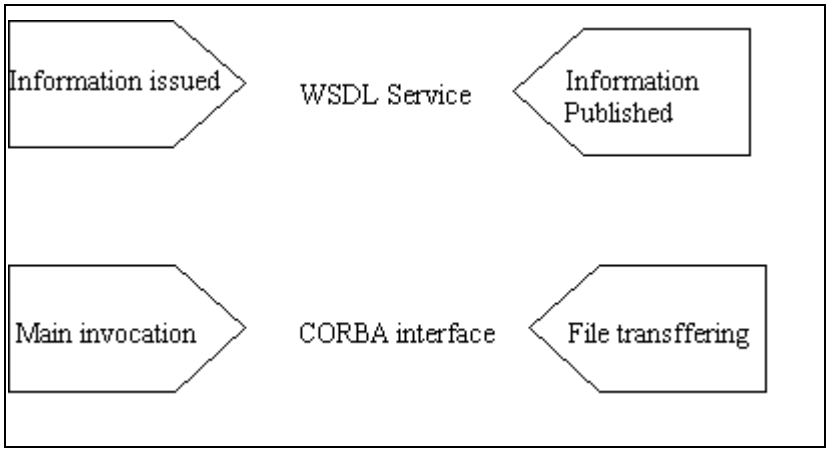

Figure 7 The functions and splitting service

4 Implementation and experiments

4.1Hardware and Software used

I only listed information which will affect computing results

4.1.1 Hardware

CPU : Pentium 4 2400 MHZ

RAM : 256 MB DDR RAM

4.1.2 Software and implementing language

OS：Windows 2000 professional

ORB：Visibroker ORB for java

The ORB provider also packed IDL compiler together, which will automatically generated servant and stub code

Web Server for both URL Naming and WSDL published：Gatekeeper （published by Borland）. We need a web server which supports HTTP PUT commands for URL Naming service.

Implementing language and tool kits：Java with J2SDK

4.2 Interface Definition Language

We need at least three basic interfaces to handle the request.

- Information Transferring interface：Since we stored information in a file,a new joining node can invocate the information file as a whole. This interface's methods should be at every node. The IDL returns a string , it's information file's content. The parameters include：information it used and its own value. As fig 8 shows part of codes :

```
interface filetransfer {
 string file(in out string infotype, in status);
 .....
}
```

Figure 8 part of codes in information data transferring interface (IDL)

- Real processing interface：This interface is defined what method we want to invocate , as mentioned previously, big number factorial computation operation.
- URL Naming interface：This interface is used to get the IOR of a specified object through its URL string and ior file name.

4.3IDL Compiled files

By using Visibroker IDL compiler, one interface will generate seven files. For filetransfer interface, they are：

- _filetransferstub.java：Stub code for the filetransfer object on the client side.
- filetransfer.java：The filetransfer interface declaration.
- filetransferHelper.java: Declares the filetransferHelper class, which defines helpful utility methods.
- filetransferHolder.java: Declares the filetransferHolder class, which provides a holder for passing filetransfer objects.
- filetransferOperation.java: This interface provides declares the method signatures defined in the filetransfer interface in the idl file.
- filetransferPOA.java: POA servant code (implementation base code) for the filetransfer object implementation on the server side.
- filetransferPOATie.java: Class used to implement the filetransfer object on the server side using the tie mechanism.

When implementing all interfaces, we don’t need to change the auto-generated code. We only need to implement server side operations. And to make server and client become runnable, we need to write server side and client side codes.

4.4Big number factorial computation

We should define the factorial computing IDL. Fig 9 is what we generated server side implementing interface by class diagram. And Fig 10 is client side runnable application which uses IDL auto-generated stub code.

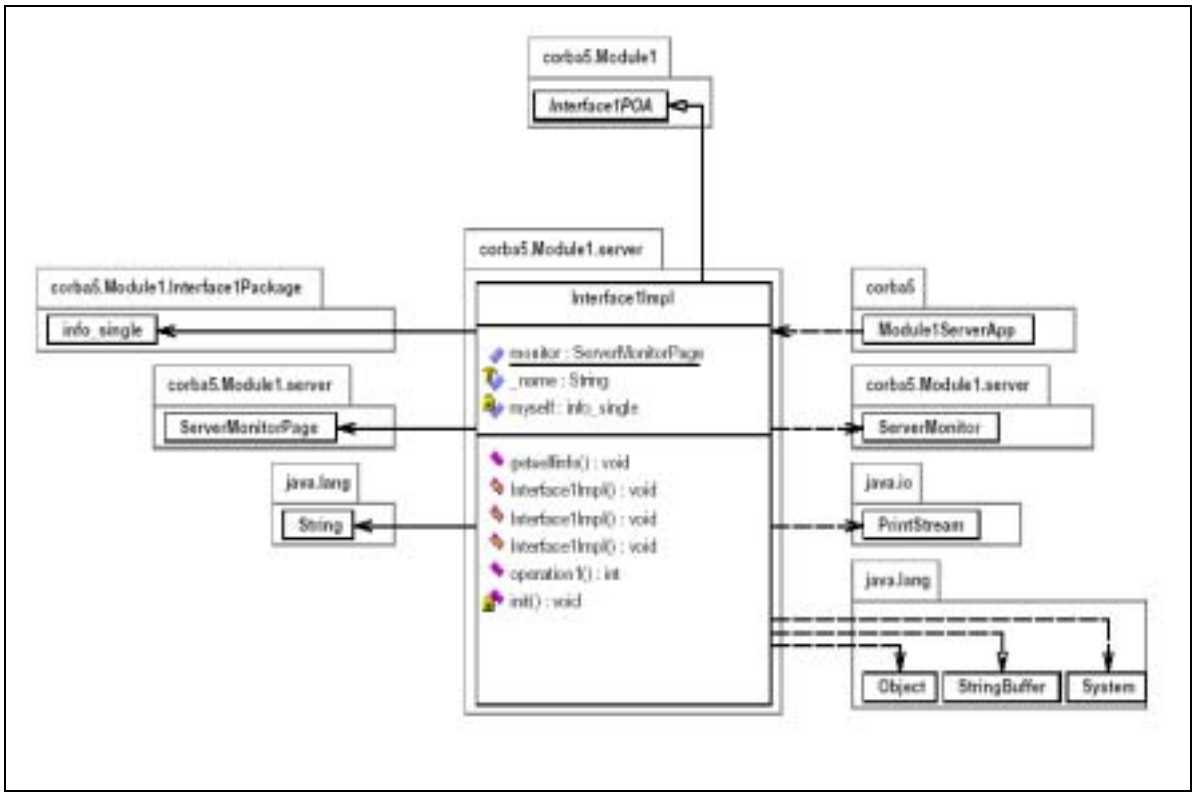

Figure 9 Server side skeleton implementation class diagram

Fig 9 shows an operation 1 which is computation work. And this implementation is inhered from POA interface.(Portable Object Adapter). And we include this skeleton in our server application.

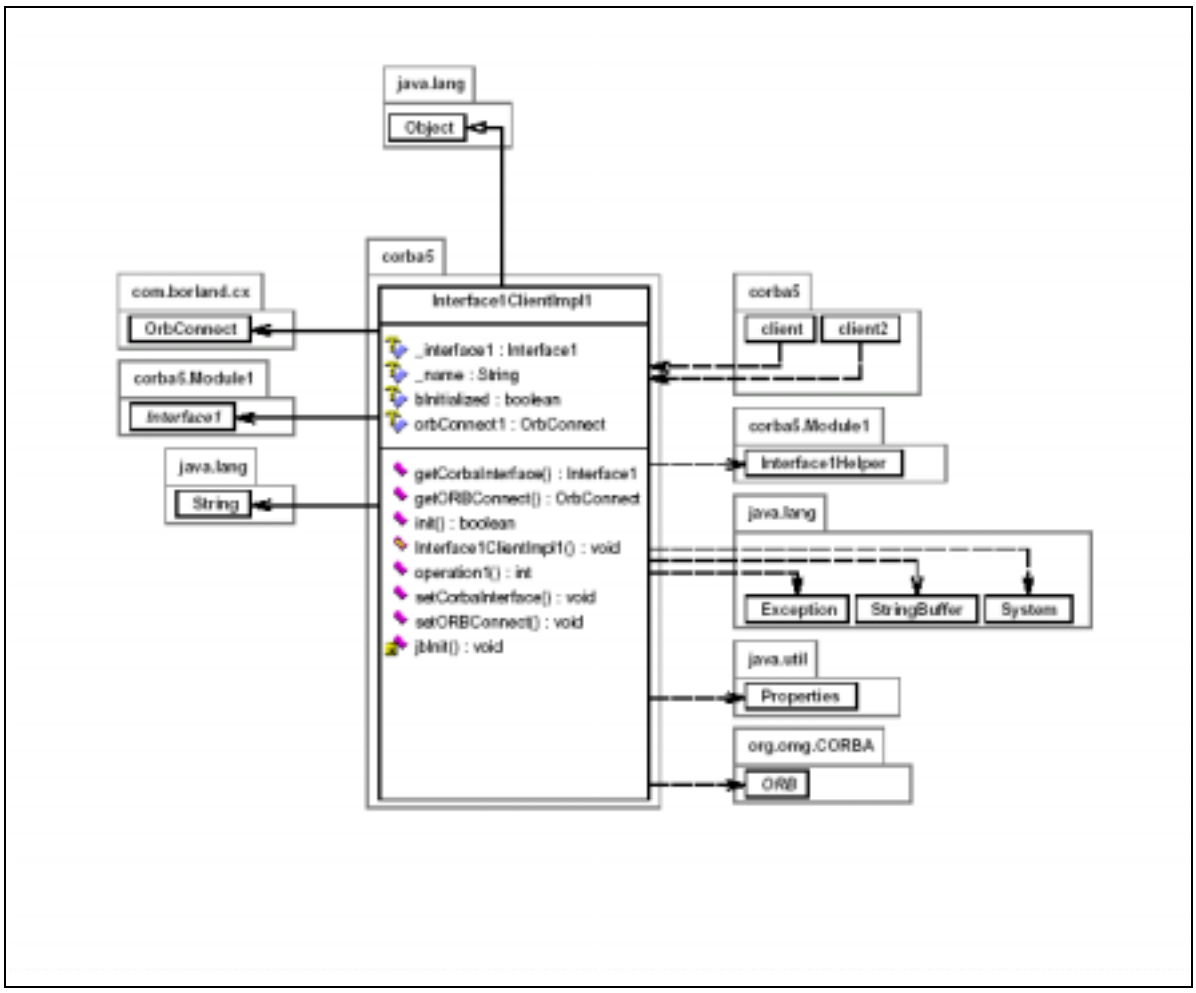

Figure 10 Client side stub application

In fig 10 we can see such method as “setCORBAInterface()” and “setORBConnect()”, which is to bind the object. And “Operation1()” is only a dump function which contains no implementation but only invocate server’s method.

And this information is kept as a host’s Host_info. This information will be published through WSDL service.

4.5 Define the host information

We want to use the WSDL service to publish local information. So we must write the information we provide as a server skeleton and transfer it to an readable WSDL file. Fig 11 shows this information structure and its methods in JAVA.

We define 4 important values in one data set, hostname、datatype、value、width, which means the host、what kind of information it gathered、the average value in a specified time interval and the time interval of one publishing.

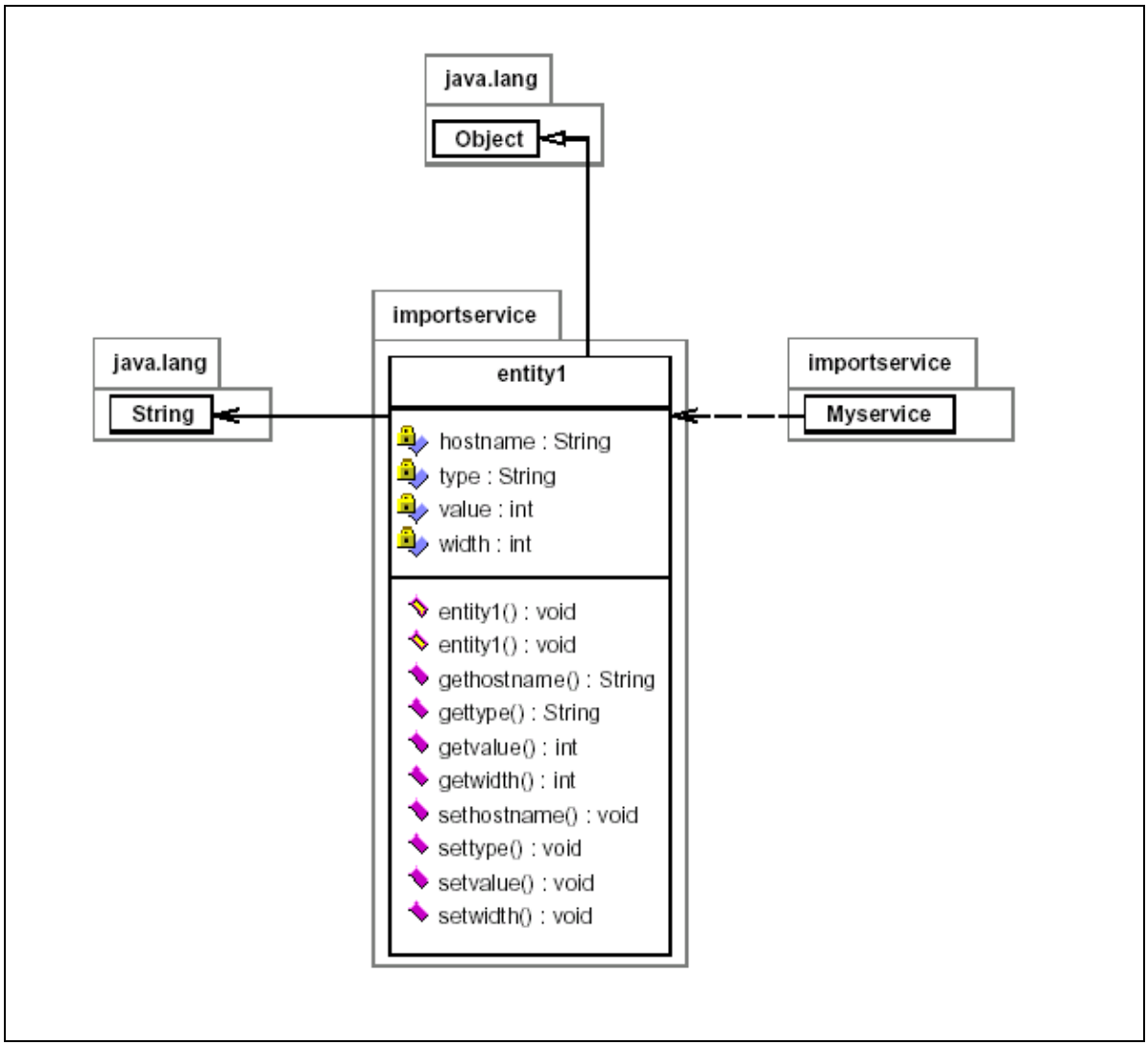

Figure11 Information structure

And then we publish it to the web server, which also serves URL Naming service.

The WSDL file is shown in Fig 12. We can see that it defines the **message** return by asking this information. And **message** contains information what we mentioned before. A client once saw this wsdl file, it can know how to operate information it gets and where to bind this service.（Binding work and operation stub is written separately in a package, as Fig 12. Fig 13 shows piece of codes for binding and stub）.

After a client get the message, it can use WSDL defined pattern to parse the message.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://importservice" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap" >
 <wsdl:types>
  <schema targetNamespace=http://importservice
xmlns="http://www.w3.org/2001/XMLSchema"><import
namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="entity1"><sequence><element name="hostname" nillable="true"
type="xsd:string"/><element name="type" nillable="true" type="xsd:string"/>
      <element name="value" type="xsd:int"/>
      <element name="time " type="xsd:int"/></sequence></complexType></schema>
 </wsdl:types>
   <wsdl:message name="getentity1Response">
      <wsdl:part name="getentity1Return" type="intf:entity1"/></wsdl:message>
   <wsdl:message name="getentity1Request"></wsdl:message>
   <wsdl:portType name="Myservice">
      <wsdl:operation name="getentity1">
         <wsdl:input message="intf:getentity1Request" name="getentity1Request"/>
         <wsdl:output message="intf:getentity1Response" name="getentity1Response"/>
      </wsdl:operation></wsdl:portType>
   <wsdl:binding name="MyserviceSoapBinding" type="intf:Myservice">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="getentity1"> <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getentity1Request">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://importservice" use="encoded"/></wsdl:input><wsdl:output
name="getentity1Response">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
```

```
namespace="http://importservice" use="encoded"/>
</wsdl:output></wsdl:operation></wsdl:binding>
<wsdl:service name="MyserviceService">
<wsdl:port binding="intf:MyserviceSoapBinding" name="Myservice">
<wsdlsoap:address location="http://localhost:8080/myservice/services/Myservice"/>
</wsdl:port></wsdl:service></wsdl:definitions>
```

Figure 12 WSDL file, defined the operation and type

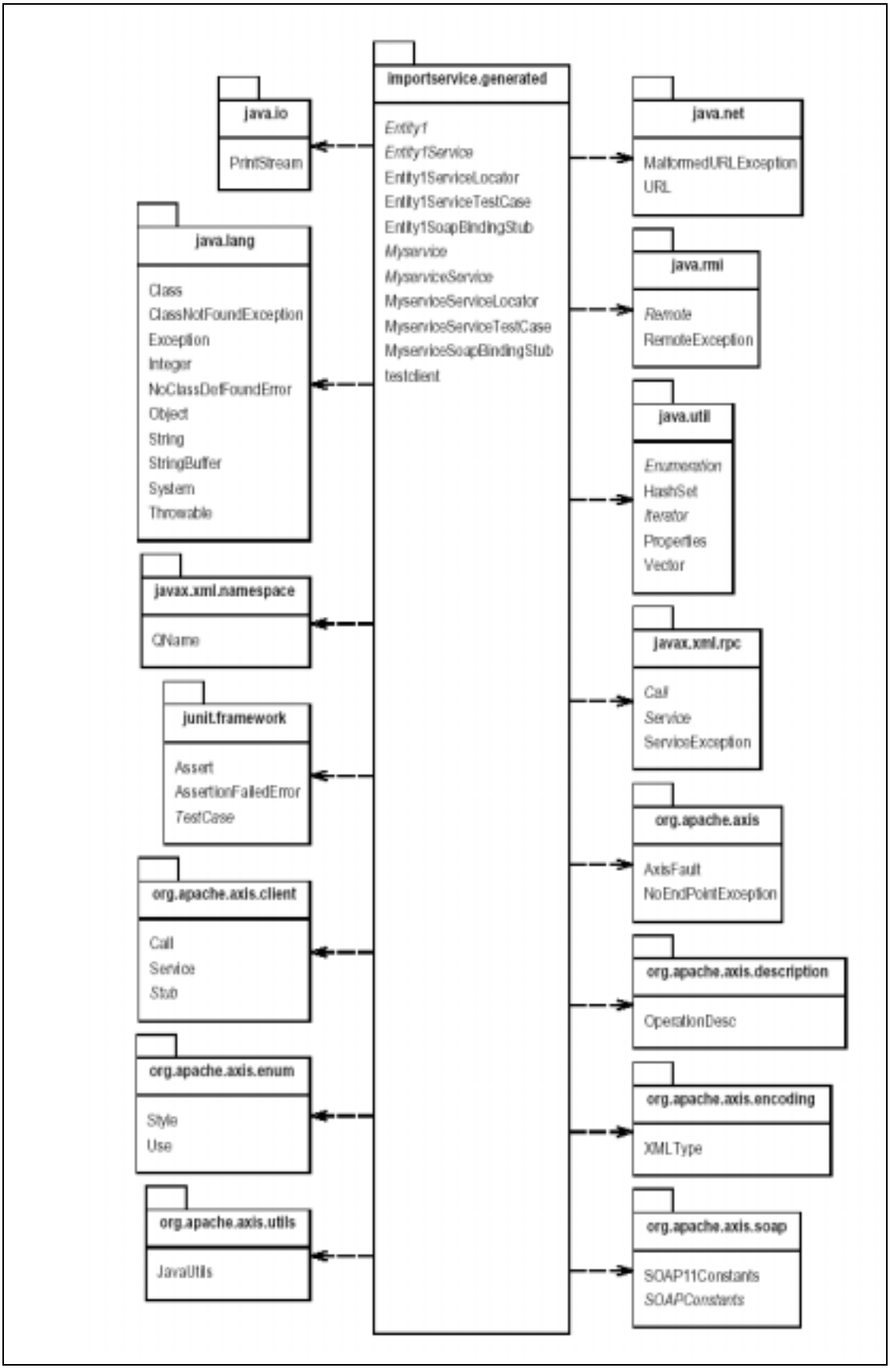

Figure 13 Generated stub codes for client side

From Fig 13, we can see many Entity1Service… method, these methods are stub codes for client to bind with Server's published WSDL service. Such as Locator, it is used to located the service defined in WSDL's <address location > element. And BindingStub is doing the work which parse the server's returned message.

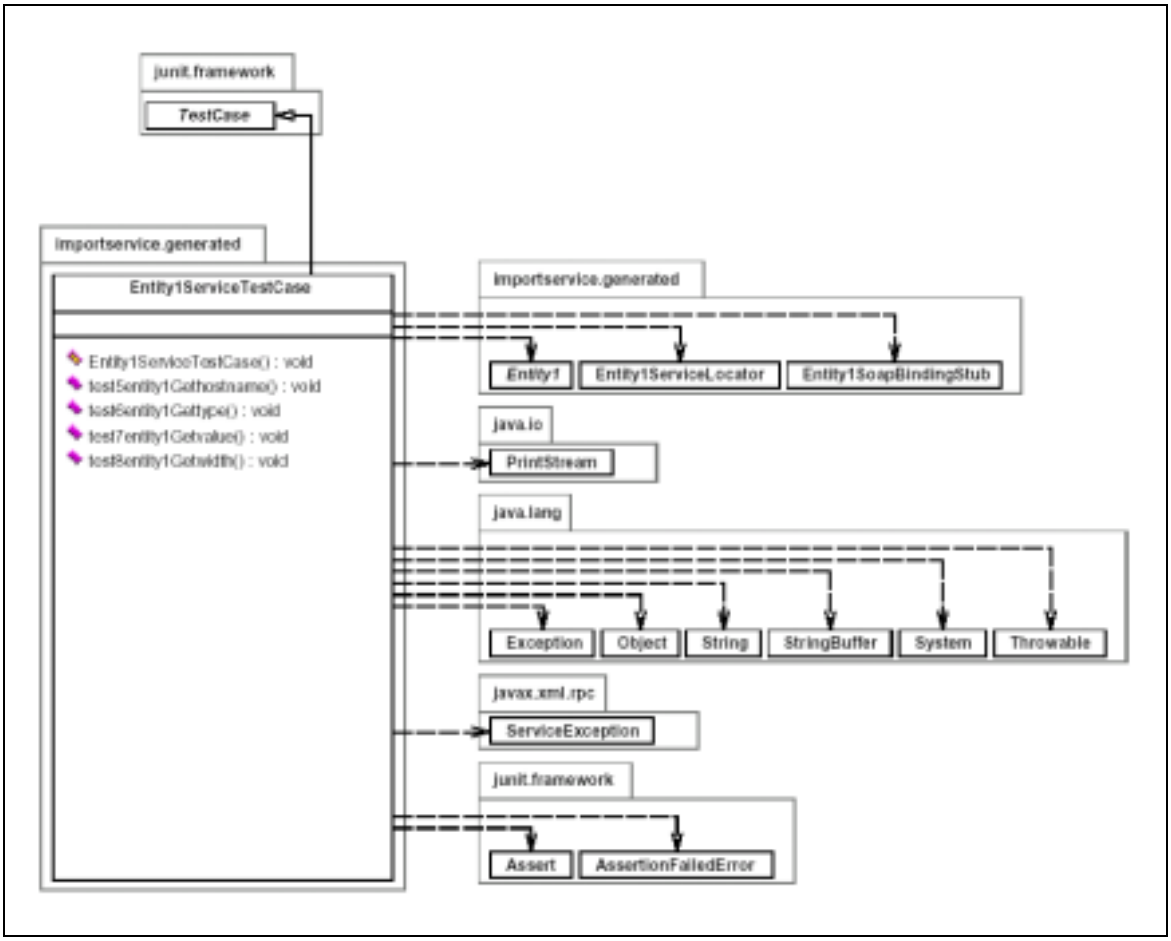

Figure 14 A test case to use a WSDL service

This test case uses Locator to get service entity and then uses BindingStub to parse the message its get.

4.6 URL Naming interface

Object's Interoperable Object Reference (IOR) can be obtained via many mechanisms, such as naming or directory services. But it may cause these machines block the scalable growth and make "distributed" become "central". Using the URL

Naming allows us to associate a URL (Uniform Resource Locator) with an object's IOR. Once a URL has been bound to an object, client applications can obtain a reference to the object by specifying the URL as a string instead of the object's name. The URL Naming Service is a simple mechanism that lets a server object associate its IOR with a URL in the form of a string in a file. The URL Naming Service supports the http URL scheme for registering objects and locating an object by the URL. Also, it enables client applications to locate objects provided by any vendor.

4.6.1 The IDL of URL Naming

URL Naming service has its own IDL definition. Therefore it can be used under CORBA. We should use this IDL to understanding how to invocate this method.

It defines a server how to register a service to a URL string and a client how to locate a service via URL string. It also defines exceptions raised when the service not exists(when request a service) or invalid URL(when register a service or request a service). Fig 15 shows URL Naming definitions.

```
module URLNaming {
exception InvalidURL{string reason;};
exception CommFailure{string reason;};
exception ReqFailure{string reason;};
exception AlreadyExists{string reason;};
abstract interface Resolver {        // Read Operations
Object locate(in string url_s)
raises (InvalidURL, CommFailure, ReqFailure);    // Write Operations
void force_register_url(in string url_s, in Object obj)
raises (InvalidURL, CommFailure, ReqFailure);
void register_url(in string url_s, in Object obj)
raises (InvalidURL, CommFailure, ReqFailure, AlreadyExists);
};
};
```

Figure .15 IDL Definition of URL Naming

5 Simulation Results and Analysis

5.1 WSDL binding time

We should first define the time interval of each updating and the accepted latency of transferring status information. We measured the binding time of WSDL service to figure out the latency. Table 2 shows the average binding time for each 10 trials.

Table 3 Average WSDL binding time

Trials	0~10	11~20	21~30	31~40	41~50
Time in sec.	1.973	3.317	1.572	2.93	3.24

This information should be taken for deeper consideration, because we bind to “local” web server. If we use the real network environment, the time will grow grandly.

5.2 The value interval

We must simulate some patterns of value-initiate and query-initiated value interval shrinks to decide the most suitable value interval we use. First, we observe that the value of CPUusage performing a random walk between 10% interval . We made it as a uniform distribution and simulated a workload having query period T_q = 2 sec. Also, we think the case of ρ =1. From times of simulation, the W converges to 3.11%. Therefore we define the value interval as 3.11%.

The simulating result is shown as Fig 19.

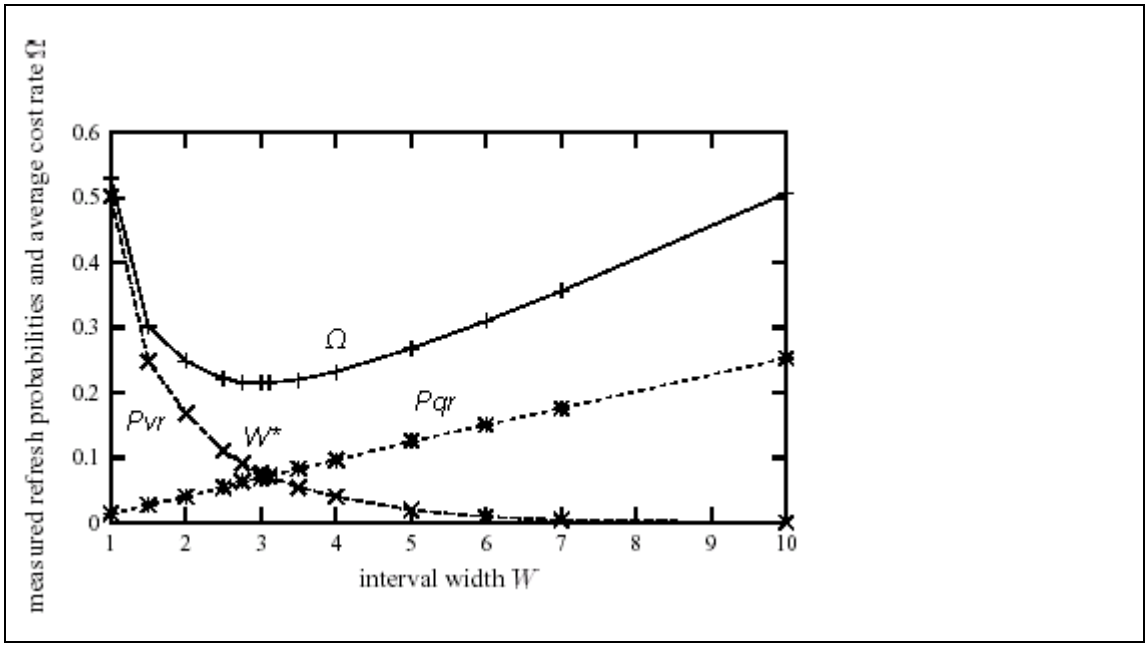

Figure 19 Simulation for optimal width

5.3 Time for binding an object with its URL String

First, I tested single character return by using URL Naming service（which involve no computation）. And the time is shown in Table 3.

Table 4 Average binding time of URL Naming service

Trials	0~10	11~20	21~30	31~40	41~50
Time	3.05	2.149	2.38	1.792	2.01

Compare with URL Naming, we have Visibroker smart agent computing time for the same service.

Table 5 Average binding time of Visibroker Smart Agent directory service

Trials	0~10	11~20	21~30	31~40	41~50
Time	2.34	3.314	3.18	3.562	3.671

We can find that smart agent did not perform better than URL Naming.Because smart agent first finds its repository to find the mapping IOR of an object.

Real network should also be taken into consideration here. Would Smart Agent out-perform than URL Naming in real environment？Or this kind of “central” processing will cause bottleneck of network traffic and then delay the service？

5.4 Overall execution time of big number factorial computation

We used the designed model to test the total time. Compare with Visibroker’s smart agent and locally execution time（without any mechanism on loading balance）. This test is tested by 2 servant objects and 1 invocation.

The invocation completed time is shown as follows.(Average of each 10 trials)

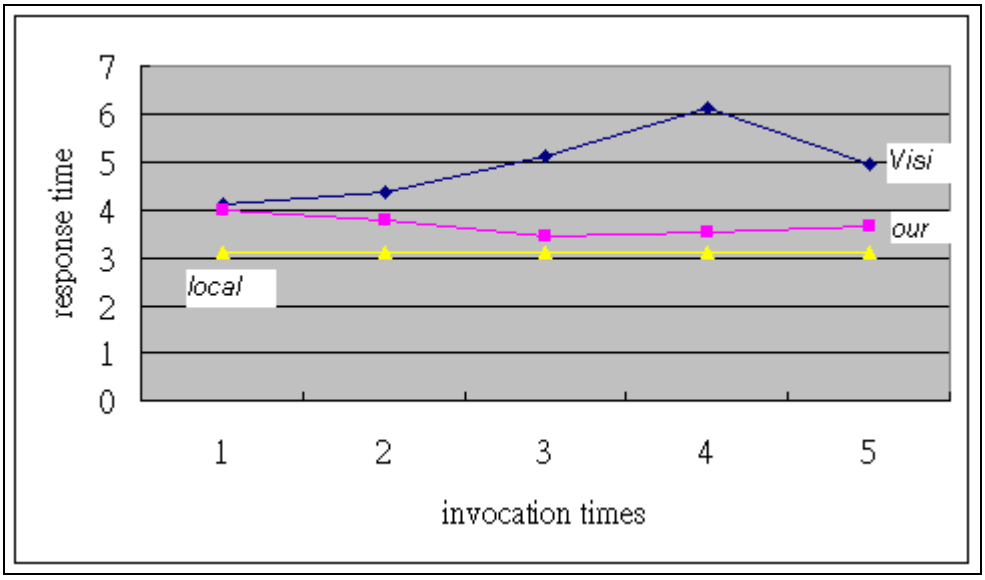

Figure 20　The execution time of computation

Summarized the result as table 5（Time of two mechanism model – Time of locally execution）

Table 6 Summary of the results

Trials	1-10	11-20	21-30	31-40	41-50
Time(Visibroker)	1	1.26	2.01	3	1.85
Time(My model)	0.86	0.66	0.35	0.42	0.55

Because we ran the WSDL service at the same time, it will take additional system load. We also recorded system load information within one single node in Table 7.

Table 7. The records of System’s resource usage

Service / CPU usage	Min Usage (%)	Max Usage	Steady State
CORBA servant only	21	100	24
With WSDL service	24	100	26

In Table 7 we can find that, when CORBA object is waiting for invocation, its average CPU usage is 24 %. The max value appeared when initiating this service. When we run the WSDL service simultaneously, it won't increase too much overhead of the system. Note that the max value is observed when initiating service or the system's task manager process takes place.

5.5 Ease of Programming

Besides the quantified analysis, we should also make the qualified analysis. In Visibroker's smart agent, there are several characteristics in programming with Visibroker (both skeleton and stub). We made it compare with our model.

Table 8 Qualified analysis of the model

Characteristics / Models	Know objects name in advance at client side	Have Defined IDL in advance at client side	Additional information of other nodes	Splitting of loading balance function with main invocation
Visibroker	Yes	Yes	No(But Smart Agent's location)	Yes. Loading balance is done by smart agent
My Model	Yes	Yes	Yes	Partially. The loading balance function should be taken when starting each invocation

From Table 8, we can see that our model's usability is almost as good as Visibroker's. The only different is our model should do some gathering work. But this

work is embedded in each host, so it won't make the programmer confused. And when we want to invocate as a client, we must first evaluate the loading status in client side's file. Although this evaluation work is done automatically, we should write into our stub code first. This may cause programmers' inconvenience.

5.6 Analysis of simulation

Using the proposed model, the executed time is better than Visibroker's smart agent. But the following problems should taken into consideration：

- If the system's scale grows, the efficiency will perform better in our model. And, Visibroker ORB contains some loading-balance algorithm（round-robin）, and it may works fine in small scale.
- The parameters of choosing execution hosts：Though we would say that CPU usage will affect computation most, other factorials will affect its performance, too, such as memory usage, cache in CPU etc.
- Difference between real world and single machine simulation：In both Visibroker's model and my model, it's necessary to consider the network status. Asking for directory service to re-direct to true IOR may take longer time than directly asking service through its IOR as URL String.

6. Conclusion and future works

6.1 Conclusion

In the above experiments and tests, we can find that some results are not as good as we expect. Because in Visibroker, they developed many none-application layer mechanisms to loading-balance work e. In our model, we only took application-layer mechanism to do these works. That may cause different performance.

We have discussed some relevant aspects of load distribution and balancing in distributed systems. The paper presented two main components for supporting them, which attempt to face some of the known problems.

- The User-Supervised Load Balancing method presented here is part of an incremental development to be extended as an automatic Load Balancing System.
- Transparent remote process execution constitute by itself a research area in distributed systems which is intimately related to those systems supporting load balancing. The design of the remote execution subsystem proposed here is an initial step which in a straightforward and simple manner permits to understand and face most of the relate problems.

6.2 Future works

In our experiment environment, we didn’t take many factorials into consideration, such as network’s bandwidth. Also, this invocated method can only be completed in single replica without status transferring. But in the real-world, many tasks in distributed systems are completed by many hosts’ coordination. If we take these into our experiments, the results may be closer to the reality.

References：

[1] M. Cukier, J. Ren, C. Sabis, W.H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr, and R. Schantz. AQuA：”An adaptive architecture that provides dependable distributed objects.” In *Proceedings of IEEE 17th Symposium on Reliable Distributed Systems*, pages 245-253, West Lafayette, IN, October 1998.

[2] P. Felber, R. Guerraoui, and A.Schiper. “The implementation of CORBA object group service.” *Theory and Practice of Object Systems*, 4(2):93-105,1998.

[3] http://263.aka.org.cn/Lectures/002/Lecture-2.1.9/Lecture-2.1.9/Corba.html

[4] http://java.sun.com/j2se/1.4.2/docs/api/org/omg/CORBA/package-summary.html

[5] http://www-2.cs.cmu.edu/~priya/fault-tolerance-resources.html

[6] Florin Sultan, Thu Nguyen, Liviu Iftode. “Fault-Tolerant Distributed Shared Memory” *Proceedings of the IEEE/ACM SC2000 Conference* .Scalable November, 2000

[7] Greg Bronevetsky, Nadiel Marques, Keshav Pingali, Paul Stodghill.” Collective Operations in Application-level Fault-tolerant MPI” *Proceedings of the 17th annual international conference on Supercomputing* , June,2003

[8] G.Cabillic,G.Muller,I.Puaut.”The Performance of Consistent Checkpointing in Distributed Shared Memory Systems.Proc.” *14th Symposium on Reliable Distributed Systems* pp.96-105,September 1995.

[9] L.E. Moser and P.M Melliar-Smith. “Strongly Consistent Replication and Recovery of Fault-Tolerant CORBA Applications”. *Theory and Practice of Object Systems*, 4(2):81--92, 1998 .

[10] L.E. Moser, P.M. Melliar-Smith, and P. Narasimhan. “Consistent object replication in the Eternal system”. *Theory and Practice of Object Systems*, 4(2):81--92, 1998.

[11] Alessio Bechini, Pierfrancesco Foglia and Cosimo Antonio Prete. “Use of CORBA/RMI Gateway : Characterization of Communication Overhead”. *Middleware performance analysis*. pp150-157, 2002/

[12] http://www.cs.huji.ac.il/support/docs/java/vbroker/vbj/pg/frames/vbj_toc.htm

[13] http://www.goland.org/Tech/wsdl.htm

[14]http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnexxml/html/xml10152001.asp

[15] Chris Olston, Jing Jiang, and Jennifer Widom “Adaptive Filters for Continuous Queries over Distributed Data Streams” *Proceedings of the 2002 ACM SIGMOD international conference on Management of data* June 2002

[16] Miron Livny, Myron Melman “Load balancing in homogeneous broadcast distributed systems” *Proceedings of the Computer Network Performance Symposium P. 47 – 55, 1982*

[17] A Hac, T Johnson “A study of dynamic load balancing in a distributed system” *Proceedings of the ACM SIGCOMM conference on Communications architectures & protocols. p.348-356 1986*

[18] D. Arredondo, M. Errecalde, F. Piccoli, M. Printista, R. Gallard, s. Flores “Load distribution and balancing support in a workstation-based distributed system” *ACM SIGOPS Operating Systems Review* Volume 31 , April 1997

[19] C. Olston, B. T. Loo, and J. Widom. “Adaptive precision setting for cached approximate values.” In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 355–366, Santa Barbara, California, May 2001.

[20] Cheng Liao, Margaret Martonosi, Douglas W. Clark. “An Adaptive Globally-Synchronizing Clock Algorithm and its Implementation on a Myrinet-based PC Cluster”.