이동 객체 기반 병렬 및 분산 응용 수행을 위한 전역 프레임워크

(A Global Framework for Parallel and Distributed Applications with Mobile Objects)

한 연 희 † 박 찬 열 † 황 종 선 †† 정 영 식 †††
(Youn-Hee Han)(Chan Yeol Park)(Chong-Sun Hwang)(Young-Sik Jeong)

요 약 월드 와이드 웹은 가장 커다란 가상 시스템이 되고 있다. 최근의 연구 분야에서, 많은 계산량을 지닌 응용을 수행시키기 위해 월드 와이드 웹에 존재하는 여러 휴지 호스트들을 이용하는 아이디어가 등장하고 있으며, 이러한 새로운 컴퓨팅 패러다임을 전역 컴퓨팅이라고 부른다. 우리는 이 논문에서 Tiger라 불리우는 이동 객체 기반 전역 컴퓨팅 프레임워크를 구현하여 제시한다. Tiger의 첫 번째 목표는 객체들의 분산, 전달, 이동과 계산행위의 동시성을 지원하는 객체 지향 프로그래밍 라이브러리를 제시하는 것이다. 이 프로그래밍 라이브러리는 프로그래머에게 분산 및 이동 객체에 대한 접근, 위치 및 이동 투명성을 제공한다. Tiger의 두 번째 목표는 전역 컴퓨팅의 요구 조건인 확장성 및 자원, 위치 관리를 지원하는 것이다. Tiger 시스템과 제공하는 프로그래밍 라이브러리는 프로그래머로 하여금 전역적으로 확장된 컴퓨팅 자원을 활용하여 객체 지향 병렬 및 분산 응용을 쉽게 작성하게 해준다. 또한, 우리는 병렬 프랙탈 이미지 처리 및 유전자 뉴로 퍼지 알고리즘과 같은 매우 많은 연산량을 지닌 응용을 Tiger 시스템에 적용하여 성능 향상 정도를 보인다.

Abstract The World Wide Web has become the largest virtual system that is almost universal in scope. In recent research, it has become effective to utilize idle hosts existing in the World Wide Web for running applications that require a substantial amount of computation. This novel computing paradigm has been referred to as the advent of global computing. In this paper, we implement and propose a mobile object-based global computing framework called Tiger, whose primary goal is to present novel object-oriented programming libraries that support distribution, dispatching, migration of objects and concurrency among computational activities. The programming libraries provide programmers with access, location and migration transparency for distributed and mobile objects. Tiger's second goal is to provide a system supporting requisites for a global computing environment - scalability, resource and location management. The Tiger system and the programming libraries provided allow a programmer to easily develop an object-oriented parallel and distributed application using globally extended computing resources. We also present the improvement in performance gained by conducting the experiment with highly intensive computations such as parallel fractal image processing and genetic-neuro-fuzzy algorithms.

· 이 연구는 정보통신부의 대학기초연구 지원사업에 의하여 수행되었음.
† 비 회 원 : 고려대학교 컴퓨터학과
yhhan@disys.korea.ac.kr
chan@disys.korea.ac.kr
†† 종신회원 : 고려대학교 컴퓨터학과 교수
hwang@disys.korea.ac.kr
††† 종신회원 : 원광대학교 컴퓨터정보통신공학부 교수
ysjeong@wonkwang.ac.kr
논문접수 : 1999년 12월 27일
심사완료 : 2000년 9월 9일

1. Introduction

Utilization of resources available to a network of workstations has served well to gain enough computing resources in order to execute a computationally intensive application [1, 2, 3]. However, a significant administration is required for them, so that administrators have to manually download

codes, install and configure the system. This restricts the size of the participating hosting group and the benefit of parallel and distributed computations.

Recently, the World Wide Web (hereinafter referred to as the Web) has become the largest virtual system that is almost universal in scope. There have been substantial changes in this Internet era, which have resulted in the proliferation of low-priced powerful hosts connected by high-speed links. At any given moment, however, many hosts are idle. An appealing idea is to utilize these hosts to run applications that require a substantial amount of computation. Such a novel computing paradigm has been called Global Computing.

Some of the obstacles common to global computing include the heterogeneity of the participating hosts, difficulties in administering distributed applications, and security concerns of users. The Java language and Java applets have successfully addressed some of these problems, since the Java executing environment genuinely supports platform-independent portability. The growing number of Java-capable browsers is able to load applets remotely so that administrative difficulties are reduced. The browsers execute applets without valid security certificates in a trusted environment, which alleviates some of the users' security concerns. Furthermore, Java is a simple, robust, multithreaded language and is designed to support distributed applications. Therefore, Java and Java applets with Java-capable Web browsers have become good candidates for the construction of a global computing platform [4, 5].

The object-oriented paradigm is ideal for parallel and distributed computations, because users can naturally treat objects as the distributing and dispatching units that are executed concurrently. Objects can become a set of autonomous communicating entities that express intensive computation. With current state-of-the-art technology, however, developing parallel and distributed applications using globally extended computing resources requires special knowledge beyond that needed to develop an application to run on a single host. Our first challenge is to provide the global parallel and distributed programming libraries that can combine distribution, dispatching, migration of objects and concurrency among computational activities into existing sequential programs. Global parallel and distributed programming can be simplified by combining the libraries into the classical Java language. The libraries provide programmers with a globally extended virtual address space, and allow them to write the programs in a shared-memory style.

Since numerous hosts that contribute CPU resources participate in a global environment, the global infrastructure employed must allow the systems growth without the user's awareness. Furthermore, the hosts manifest different performances and memory capacity, which may change variously in the duration of the long execution time. Therefore, it is necessary to move numerous objects among the globally extended computing resources provided in order to efficiently execute a parallel and distributed application with optimal use of resources. Because of such a migration, it is important to achieve transparency through a proper location management scheme. Our second challenge is to deal with three requisites for object-oriented global systems - scalability, global resource management, and location management for mobile objects.

This paper is organized as follows: Section 2 discusses related works. A description of our system and object model is given in Section 3. This is followed by a detailed discussion about parallel and distributed programming using global resources in Section 4. Functional requisites such as resource and location management are presented in Section 5. Experimental results are then presented in Section 6, and conclusions are presented in Section 7.

2. Related Works

Several approaches have been proposed recently

to provide a Java-based distributed and global computing framework. In this section, we discuss some of their characteristics and functional deficiencies.

ATLAS[6] is the earliest infrastructure supporting global computing using Java. It ensures scalability using a hierarchy of managers. However, it may raise some problems regarding portability, since its implementation uses native libraries. ParaWeb[7] provides a framework for utilizing Internet resources for parallel computing. It allows a Java program using Java threads to be executed in parallel on a shared-memory environment consisting of several hosts on the Internet. However, its runtime system is implemented by modifying the Java interpreter, while its new class libraries provide a message-passing framework for sending and receiving messages to and from threads spawned on remote machines. ATLAS and ParaWeb, therefore, may be more suitable to a LAN environment.

SuperWeb[8] provides an infrastructure for global computing with emphasis on ease of CPU donors' participation. It discusses various technical challenges, such as interoperability, execution speed, security, correctness and communication, associated with a global computing environment. However, it does not supply a programming model for a parallel and distributed application.

Also, POPCORN[9] provides any programmer connected to the Web with one huge virtual machine. It is noted that a market-based mechanism of trade in CPU time is provided. It supports a special programming paradigm such that it eliminates any need for explicit use of any other types of concurrency and event-driven model. However, the paradigm requires programmers to learn new programming skills, because there are many gaps between classical object-oriented programming and POPCORN programming.

DJM[10] provides a novel model of global computing with introducing applet helper mechanism, which allows applets to communicate with any host and act as servers in the system without lowering the security level of the Java applet technology. Its programming paradigm is like that of CORBA and JavaRMI. Access and location transparency between the local object and the remote object is provided to a programmer. However, since it does not support any resource management mechanism, it is possible that some heavily loaded nodes are present.

Charlotte[11] provides a distributed shared memory and uses a fork-join model for parallel programming. Its distinctive feature is the eager scheduling of tasks where a task may be submitted to several servers to provide for fault tolerance and ensure timely execution. This allows a task to be resubmitted to a different server, in case the original server fails. But compared to simpler message-passing systems, maintaining correct memory semantics requires a high overhead. Moreover, it requires programmers to learn a new programming paradigm like that of POPCORN.

Java//[12] provides a framework for the development of metacomputing applications. The most important feature of Java// is that it provides very smooth transition among sequential, multithreaded and distributed programming. Given a sequential Java program, it only takes minor modifications for the programmer to make it ready for metacomputing. Dejay[13] is a new Java-based programming language that unifies concurrency and distribution into a single mechanism. This allows simplification of the development of distributed or concurrent programs. HORB[14] extends the Java language by providing a remote procedure call mechanism. Although HORB can run in Java applet using a Web browser, the node running the applet can only connect to its originating host and acts as a client of the system. This restricts HORB users to the traditional client-server programming model. Moreover, since users of Java//, Dejay and HORB's must specify the address of a target server for remote objects, location transparency is not supported for them. They likewise do not support any resource management mechanism.

Table 1 Comparison of Related Works

	ATLAS	ParaWeb	SuperWeb	POPCORN	DJM	Charlotte	Java//	Dejay	HORB	*Tiger*
World-Wide Scalability	×	×	Support	Support	×	Support	×	×	Support	***Support***
Progamming Paradigm	×	Difficult	Easy	Difficult	Easy	Difficult	Easy	Easy	Easy	***Easy***
Location Transparency	×	Support	×	Support	Support	Support	×	×	×	***Support***
Object Migration	×	×	×	×	×	×	×	Support	×	***Support***
Resource Management	Support	×	×	×	Support	Support	×	×	×	***Support***
Location Management	×	×	×	×	×	×	×	×	×	***Support***

3. System and Object Models

3.1 System Model

Our system model consists of six kinds of major components: users, brokers, hosting applets, gateways, regions, and a manager (Fig. 1).

- *Users* wish to use extra computing power in order to run parallel and distributed applications with large computations.
- *Brokers* manage the user applications and coordinate all communication between Tiger and the applications. It is necessary for a user to execute a broker before executing his or her application.
- *Hosting applets* allow their CPU resources to be used by other users. Their general form is a Java applet in a Java-enabled web browser.

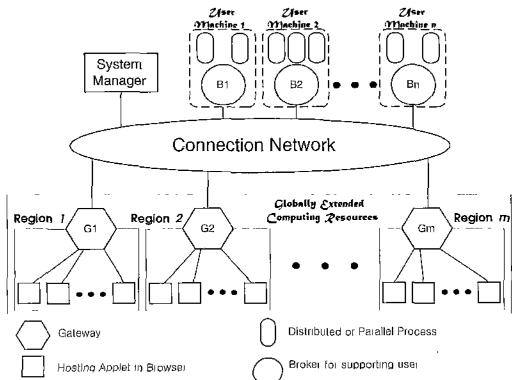

Fig. 1 The Tiger System Architecture

- *Gateways* manage parts of all hosting applets and coordinate all communication between hosting applets and other components in the Tiger system. We assume that each gateway serves exactly one region.
- *Regions* consist of hosting applets managed by a gateway. Regions are generated by grouping hosting applets that yield similar duration for communication to and from a gateway managing them.
- *A manager* registers and manages all participating brokers and gateways.

When CPU donors make their resources available to Tiger by browsing the HTML document on the part of the manager, the web server in the manager redirects the request to the gateway with the shortest period for communication to and from the CPU donor's machine. Therefore, the web server in the gateway sends the hosting applet code to the participating browsers.

The primary function of the manager consists of resource management for all regions and location management for all mobile objects in the Tiger system. The primary function of gateways, on the other hand, consists of resource management for hosting applets managed in its associated region, and location management for the mobile objects visiting the associated region.

There are two important reasons our model must provide gateways. The first reason is to the model

must facilitate the distribution of the manager's massive load. Because of the heavy network traffic generated by various brokers and hosts, the manager may experience a bottleneck. To reduce traffic on the manager, one natural solution is to distribute the manager functionsThe second reason results from severe limitations on the capabilities of Java applets, since they are used to execute programs from the Internet without valid security certificates. Applets cannot create a server socket to accept any incoming connection, while Java-capable browsers do not allow applets from establishing a network connection except to the machine from where they were loaded.

Using gateways as the intermediate message-exchange nodes, we allow an applet to communicate with any applet within the same region (Fig. 2(a)) or in different regions (Fig. 2(b)). Both broker and gateway mediate message exchanges between user applications and hosting applets (Fig. 2(c)). Two different user applications can likewise communicate through each associated broker (Fig. 2(d)).

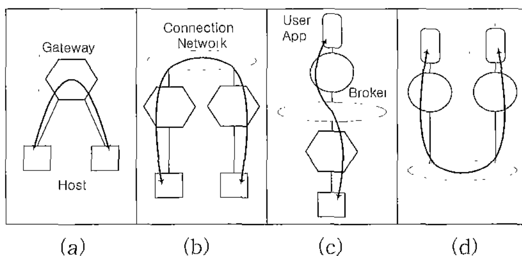

Fig. 2 The Communication Model

3.2 Triple-Object Model

Tiger is designed as an object-oriented global system which expresses computations as autonomous communicating entities referred to as objects. We use a triple-object model in Tiger consisting of normal (local) objects, distributed objects and mobile objects. Normal objects are similar to standard Java objects. Distributed objects are placed in other user machines rather than in the one where local objects are placed. Like distributed objects in CORBA or JavaRMI, they are remotely accessible and their locations are fixed to the user machine where they are created. Mobile objects are similar to distributed objects except that they can change their current location from a user machine or a hosting applet to other hosting applets. The distributed object and mobile object are the basic units for distribution and concurrency. Users have to specify explicitly which objects are distributed or migrated.

A requested method call on a distributed or mobile object is either executed or queued depending upon an object's state at the time of its arrival. A distributed or mobile object can be in one of three states: dormant, active, and waiting. The one dormant is not executing any method currently, and there is no requested method call in the message queue. The one in active state is currently executing a method, while the one in waiting state is waiting for a specific response to a method call issued while in the active state. When an acceptable response arrives, the object will return to the active state. In the Tiger system, only dormant objects can be moved.

4. Parallel and Distributed Programming

When designing an object-oriented application, programmers start with high-level abstractions and turn them into objects and classes. Programmers are usually eager in modeling and dealing with algorithmic issues about the application. Our programming model concentrates on a clear separation between high-level design and lower-level implementation issues such as distribution, dispatching, migration of objects, and controlling concurrent activities. When dealing with globally distributed objects, the power of the Tiger lies in that any client process or thread in the local machine can directly interact with the server object that runs on a globally distributed user machine and a hosting applet through a remote method call, although the server object may migrate among them.

4.1 TigerObject Interfaces

A server object must declare its service via an interface by extending the TigerObject interface. Each remote method is declared in the interface. Like JavaRMI, client stub and server skeletons are generated from this interface and the implementation of client and server object is done using this interface.

4.2 Objects-distributing Mechanisms

Tiger provides us with three application programming interfaces (hereinafter referred to as the APIs) related with distributing server objects.

- *turnToDistributedObj(TigerObject obj, String name)*

This converts an existing local object, namely *obj*, into a distributed object at any time after its creation. The distributed object becomes remotely accessible and its location is fixed. The stub object is registered to the *naming subsystem* together with a given argument *name*. The stub acts as a handle for clients to reference the remote server object. The client can call methods on the stub, which are routed by the client broker to the server broker, where the skeleton executes the method upcall on the actual server object (Fig. 3(a)). This mechanism is similar to that in the CORBA and JavaRMI.

- *turnToMobileObj(TigerObject obj, String name)*
- *turnToMobileObj(TigerObject obj, String name, Region destRegion)*

Unlike CORBA and JavaRMI, on the other hand, Tiger allows the local or distributed server object, namely *obj*, to migrate from the generating machine to a hosting applet in a globally extended computing resource. Programmers do not present such information, or they can present only regional information, such as *destRegion*. If regional information is presented, the object's location can be changed only within the region. However, if the object is not allocated, its location is not fixed and it can re-migrate to any hosting applet across all regions without the users awareness. It is noted that a client process or thread does not know the detailed location of the target hosting applet. Therefore, Tiger provides users with location and migration transparency. Through such migration mechanism, we can reduce the burden on a server machine and gain performance benefits (Fig. 3(b)).

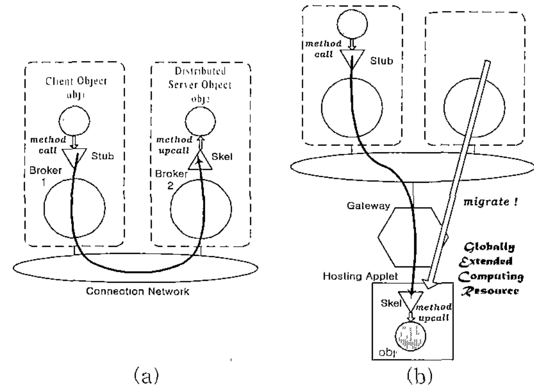

Fig. 3 Object Distributing Mechanisms

4.3 Parallel Objects-dispatching Mechanisms

Object-oriented parallel programs are largely divided into two parts: the main control program, which provides a whole body for solving a given problem; and the TigerObject, which describes tasks to be executed in parallel. The main control program must dispatch a number of TigerObjects into the globally extended computing resource and call remote methods on the dispatched objects. It can likewise watch loads imposed on each region, and balance those using object migration. Tiger provides us with the two APIs related to dispatching parallel TigerObjects.

- *turnToMobileObj(TigerObject obj)*
- *turnToMobileObj(TigerObject obj, Region destRegion)*

These methods dispatch an existing local object, namely *obj*, into a hosting applet in the globally extended resource. This object then becomes remotely accessible from the main control program (Fig. 4). It is not required to provide the name of the object since the object is used only by the main control program. Tiger itself internally allocates the globally unique identifier to the object. It is noted that its location is not fixed in the dispatched hosting applet and that it can re-migrate

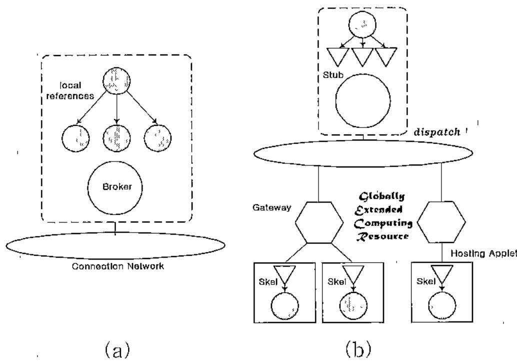

Fig. 4 Parallel Object Dispatching Mechanisms

to other hosting applets.

4.4 Getting Distributed or Mobile Objects Back

- *returnToLocalObj(TigerObject obj)*

This method reverts a distributed or mobile object into a local object. If the argument *obj* indicates a distributed (not mobile) server object, a remote client process or thread cannot call any remote method on the server object. On the other hand, if the argument *obj* indicates a mobile object, the object is eliminated from the remote hosting applet where the object is currently located. The stub and skeleton associated with the object are eliminated from the client and server machines, respectively. The object is drawn to the local machine and all method calls on the object are henceforth done within the local address space.

4.5 Remote Method Call and Concurrency

Concurrency can be used to perform a computation in parallel on several machines or on one machine simulating parallelism. In Java, threads only serve as a mechanism to express concurrency on a single host, but they do not allow concurrency between remote hosts. Therefore, there exists a huge gap between multithreaded and distributed concurrent programming.

Although method calls in Java are only synchronous, there are three concurrency- enhanced mechanisms provided to call a remote method in the Tiger model: synchronous, asynchronous, and one-way. A synchronous remote method call is typically similar to the method call in Java. The calling object must be blocked until the result is returned. Its form is the same as that of the

```
class MatrixMultiply {
  public static void main(String[] args) {
          Matrix mA, mB, resultM;
          // code of initializing mA, mB, and resultM
          ...
          TaskObj[][] obj = new TaskObj[10][10];
          for (int i = 0 ; i < 10 ; i++) {
             for (int j = 0 ; j < 10 ; j++) {
                      Matrix subA = mA.getSubMatrix(i * 10, 0, 10, 100);
                      Matrix subB = mB.getSubMatrix(0, j * 10 , 100, 10);
                      obj[i][j] = new TaskObj(i, j, subA, subB);
                      obj[i][j] = (TaskObj)Tiger.turnToMobileObj(obj[i][j]);
             }
          }
          AsyncReplySet replySet = new AsyncReplySet();
          AsyncReply[][] reply = new AsyncReply[10][10];
          for (int i = 0 ; i < 10 ; i++) {
             for (int j = 0 ; j < 10 ; j++) {
                      reply[i][j] = Tiger.asyncMethodCall(task[i][j], "multiplyMatrix", null);
                      replySet.add(reply[i][j]);
             z}
          }
          for (int i = 0 ; i < 10 ; i++) {
             for (int j = 0 ; j < 10 ; j++) {
                      Matrix m = (Matrix)reply[i][j].getResult();
                      resultM.setSubMatrix(i * 10, j * 10, m);
             }
          }
  }
}
```

Fig. 5 A Reference Code

standard method call. The asynchronous remote method call has a future style, in which the caller may proceed until the result is needed. For this duration, the caller is blocked until the result becomes available. If the result has been supplied, the caller resumes and continues. To support the asynchronous mode, we provide asyncMethodCall (TigerObject obj, String methodName, Object[] args) API, AsyncReply and AsyncReplySet classes of which the usage is shown in Figure 5. By using these classes and API, it is possible to issue several asynchronous calls to remote objects that are executed in parallel. Lastly, the one-way remote method call is also asynchronous. The caller, however, will not retain anything related by this method call, and the party being called will never have to reply to it. This is referred to as the fire-and-forget style.

To distinguish among these three mechanisms, each remote method, which is declared in a user-defined interface extending the TigerObject interface, must throw one among SyncMethodCallException, AsyncMethodCallException, and onewayMethodCallException. These three exceptions specify what the method's behavior style is.

Figure 5 presents the referential code using Tiger programming libraries that multiply the two 100×100 matrices. The program creates one hundred *TaskObj*-type mobile objects, and send one hundred asynchronous methods, *"multiplyMatrix,"* to the objects.

5. Functional Requisites

On the Web, all hosts wherein hosting applets reside, may manifest different performance and memory capacity, which may change variously during the long execution time. Therefore, it is essential that Tiger has a resource management scheme in treating these dispositions on the Web. In Tiger, since an object is the migrant unit of computation, object migration can be a suitable way to balance the workload on all the hosting applets. At a given periodic interval, Tiger moves objects from overloaded hosting applets to under-loaded hosting applets. There may be numerous method call requests during the migration of many objects. Therefore, in order to effectively provide users with migration transparency and support the mobility of mobile objects, a proper location management scheme is required.

5.1 Resource Management

Tiger uses two important parameters for resource management: *Object Population Density(OPD)* and *Call to Performance Rate(CPR)* for all regions and hosting applets. Suppose that C_{R_i} and $C_{H_{ij}}$ are the memory capacities of a region i and a hosting applet j in region i, respectively. P_{R_i} and $P_{H_{ij}}$ are the computational processing rates of a region i and a hosting applet j in region i, respectively. When the number of hosting applets in region i is defined by $NumH_i$, $C_{R_i}=\sum_{j=1}^{NumH_i} C_{H_{ij}}$ and $P_{R_i}=\sum_{j=1}^{NumH_i} P_{H_{ij}}$. The computational processing rate is estimated by solving a dense 100×100 linear system of equations using the Linpack library package[15], and its measure is MFLOPS(million floating point instructions per second).

Both performance of a region i and that of a hosting applet j in region i are given by $Perf_{R_i}=\alpha C_{R_i}+\beta P_{R_i}$ and $Perf_{H_{ij}}=\alpha C_{H_{ij}}+\beta P_{H_{ij}}$, respectively, where α and β are scale parameters. The OPD of a region i and that of a hosting applet j in region i are defined by

$$OPD_{R_i}=\frac{NO_{R_i}}{Perf_{R_i}} \quad (1)$$

$$OPD_{H_{ij}}=\frac{NO_{H_{ij}}}{Perf_{H_{ij}}} \quad (2)$$

where the number of objects in region i is given by $NO_{R_i}=\sum_{j=1}^{NumH_i} NO_{H_{ij}}$ and $NO_{H_{ij}}$ is the number of objects in a hosting applet j in region i.

On the other hand, the CPR of region i and that of a hosting applet j in region i are defined by

$$CPR_{R_i}=\frac{NC_{R_i}}{Perf_{R_i}} \quad (3)$$

$$CPR_{H_{ij}}=\frac{NC_{H_{ij}}}{Perf_{H_{ij}}} \quad (4)$$

where the number of call arrivals to region i is

given by $NC_{R_i} = \sum_{j=1}^{NumH_i} NC_{H_{ij}}$ and $NC_{H_{ij}}$ is the number of call arrivals to a hosting applet j in region i.

Tiger resource management is carried out by *the global resource manager* and *the regional resource managers*. Dispatching the mobile object into globally extended resources, the global resource manager selects the region with the lowest OPD_R value. When the mobile object enters a region, the specific regional resource manager selects the hosting applet with the lowest $OPD_{H_{ij}}$ value. At a given periodic interval, the global resource manager moves some mobile objects from a region to another in order to keep both CPR_Rs for all regions in equilibrium. All regional resource managers likewise move some mobile objects from a hosting applet to another in the same region in order to keep $CPR_{H_{ij}}$s for all hosting applets within each region in equilibrium..

In addition, Tiger presents the user-level global resource management tools, which can be used with mobile objects turned by the objects-distribution or objects-dispatching APIs with the *destRegion* argument. The tools allow a user to get both OPD_R and CPR_R of a region and to measure the time needed to execute an asynchronous method for each region. Figure 6 describes an example where user-level resource management tools are used. Initially, the two TigerRegion, *regions[0]* and *regions[1]*, are selected as the regions with the lowest OPD_R value. The two TigerObjats, *foo* and *bar*, are dispatched to *regions[0]* and *regions[1]*, respectively. After two asynchronous methods with the same computing quantity, *factorial*, are called at the *foo* and *bar*, respectively, the returned *AsyncReply* objects, *reply1* and *reply2* are added to the *AsyncReplySet* object, *replySet*. When results for the remote methods arrive, the processing times for the calling asynchronous methods are determined and stored in the AsyncReply objects. Users can inquire about each processing time by using the *getProcessingTime* methods on both *reply1* and *reply2*. It is the user's responsibility to establish a basis on which whether Tiger moves an object from an over-loaded hosting applet to an under-loaded hosting applet or not would depend.

```
...
FactorialTigerObject foo = new FactorialTigerObject();
FactorialTigerObject bar = new FactorialTigerObject();
TigerRegion[] regions = Tiger.getRegionsSortedByLowOPD();
foo = Tiger.turnMobileObj(foo, regions[0]);
bar = Tiger.turnMobileObj(bar, regions[1]);
AsyncReplySet replySet = new AsyncReplySet();
AsyncReply reply1, reply2;
reply1 = Tiger.asyncMethodCall(foo, "factorial", arg1);
reply2 = Tiger.asyncMethodCall(bar, "factorial", arg2);
replySet.add(reply1);
replySet.add(reply2);
while(!replySet.isAllAvailable());
if (reply1.getProcessingTime() > 2*reply2.getProcessingTime())
   foo = Tiger.turnMobileObj(foo, regions[1]),
```

Fig. 6 Example of user-level load balancing scheme

5.2 Location Management

Our current scheme for location management is based on a two-level hierarchy such that two types of manager, the *Home Location Manager (HLM)* and the *Visitor Location Manager (VLM)*, are involved in tracking a mobile object. There is an HLM in each broker, while a mobile object is permanently registered in the HLM associated with a broker where the mobile object is created. The broker serves as the *home broker* of the mobile object. The mobile object chooses the IP address of the broker as its *home address*. Each VLM is associated with a gateway and stores the location information of the mobile objects visiting its associated region. Whenever a mobile object moves into a new region, the mobile object's HLM is informed of the location change so that it keeps exact track of the current VLM.

Our location management includes two major tasks: *location registration* and *method call delivery*. Location registration procedures update the location managers (HLM and VLM) and *authenticate that up-to-date location information of* a mobile object is available. When a remote method

call for the mobile object is initiated, method call delivery procedures locate a mobile object based on the information available in the managers and deliver the method call to the target mobile object.

Figure 7(a) illustrates the location registration procedure when a mobile object moves to a hosting applet within the same region. The following is the ordered list of tasks performed during the location registration.

1) The gateway locks the field associated with the mobile object in the VLM such that all external call delivery requests must wait until the lock is released.

2) The gateway sends *the object migration message* to the hosting applet where the target object is located.

3) The target object migrates to the new hosting applet indicated in the object migration message. The gateway is used as the intermediate node for object-forwarding.

4) The new hosting applet sends an acknowledgement message notifying that object migration was done successfully.

5) The gateway records the new hosting applets' address for the mobile objects and unlocks the field associated with the mobile object in the VLM.

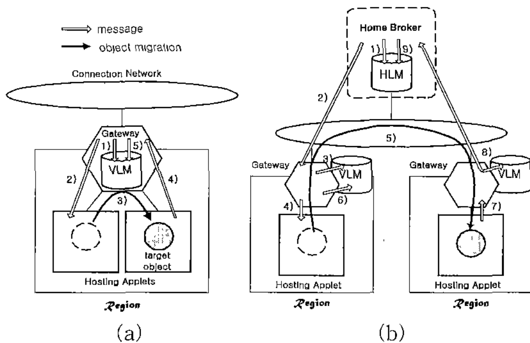

Fig. 7 Location Registration Procedures

Figure 7(b) illustrates the location registration procedure when a mobile object moves to a hosting applet within a new region. The following is the ordered list of tasks performed during the location registration process.

1) The home broker locks the field associated with the mobile object in the associated HLM such that all external call delivery requests wait until the lock is released.

2) The home broker sends the object migration message to the gateway managing the region where the target object is located.

3) The gateway locks the field associated with the mobile object in the VLM such that any external call delivery request waits until the lock is released.

4) The gateway sends the object migration message to the hosting applet where the target object is located.

5) The target object migrates to the new hosting applet, which is selected by the new gateway managing the region as indicated in the object migration message.

6) The old gateway sets up a forwarding pointer by recording the new gateway's address in the VLM and unlocks the field associated with the mobile object in the VLM.

7) The new hosting applet sends an acknowledgement message notifying that object migration was done successfully.

8) The new gateway records the new hosting applets' address for the mobile objects in the VLM and sends the migration complete message to the home broker.

9) The home broker records the new gateway's address and unlocks the field associated with the mobile object in the HLM.

When a remote method calling is requested by a user application, two major steps for method call delivery are involved for the stub to locate the skeleton associated with the implementation object: determining the visiting region of the called mobile object, and locating the current hosting applet within the region.

In Tiger, the method call delivery uses triangle routing involving the following ordered list of tasks. Triangle routing means that the message from a calling stub to a called skeleton must first be routed via the mobile object's home broker.

1) A calling stub sends a method call request to the home broker using the called skeleton's home address.

2) The HLM in the home broker determines the current location of the called skeleton.

3) If the skeleton exists in the home broker's node, the home broker delivers the method call request to the called skeleton and the method call delivery procedures are successfully carried out.

4) If the skeleton does not exist in the home broker's node, the HLM determines the serving VLM of the called skeleton and the home broker relays the method call request to the gateway associated with the VLM.

5) The VLM determines the serving hosting applet of the called skeleton and relays the method call request to the hosting applet.

6) The hosting applet delivers the method call request into the called skeleton and the method call delivery procedures are achieved successfully.

Figure 8 shows the method call delivery procedure when the mobile object locates in a hosting applet. This is similar to the triangle routing scheme in mobile IP [16].

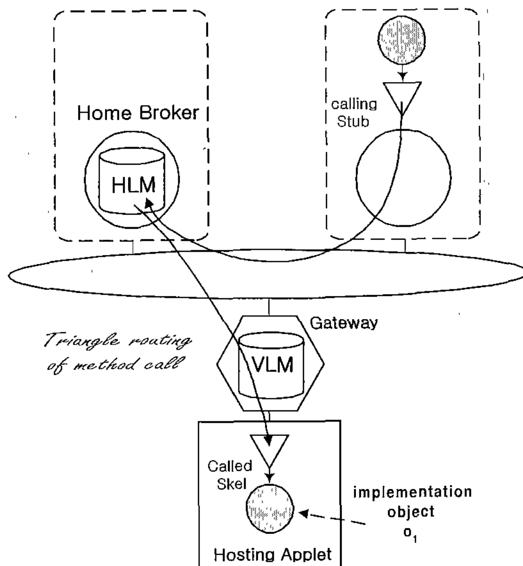

Fig. 8 Method Call Delivery Procedures

6. Experimental Results

We conducted experiments to illustrate the speedup achieved by Tiger. The target applications for the experiments are fractal image processing and genetic-neuro-fuzzy algorithm. The system manager runs on Pentium 333Mhz while three gateways run on Pentium 200, 233, 266Mhz using Java virtual machine in JDK 1.2.1. Ten hosting applets run in Netscape Communicator 4.5 or Internet Explorer 4.0 on heterogeneous machines connected by 10Mb/s Ethernet.

6.1 Fractal Image Processing

A popular representation of fractal image processing lies within the Mandelbrot set. The Mandelbrot set is the set of all points that remain bounded for every iteration of $z = z \times z + c$ on the complex plane, where the initial value of z is 0 and c is a constant. The size of the bit-image is 256×256 pixels. Each independent subtask has a responsibility for coloring 256×8 pixels with considerable floating point computation. Therefore, the number of subtasks is 32. Our parallel fractal image processing is initiated by dispatching sixteen Tiger objects having the asynchronous remote method able to generate a Mandelbrot set. A calling of the remote method corresponds to a subtask. There are 2 calls by each object. The return value of the method is the coloring information for 256×8 pixel.

We measured the time needed to generate the Mandelbrot set by changing the number of hosting applets, and the results are presented in Figure 9. The figure shows that, the more the number of

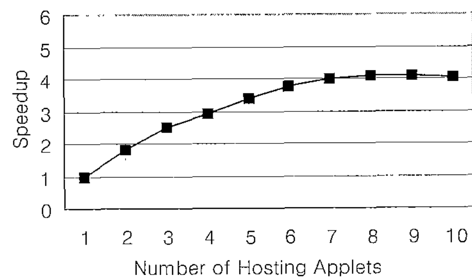

Fig. 9 Speedup for the number of hosting applets

hosting applets increases, the more the speedup increases. The speedup, however, decreases whenthe number of hosting applets is over nine. This is attributed to the low-bandwidth of networks, the overhead of communication, and short execution time of the remote method. Therefore, when users construct a remote method, they must make inputs and outputs of the method compact. However, the method's remote execution time must be long, if possible.

6.2 Genetic-neuro-fuzzy Algorithms

Genetic-neuro-fuzzy algorithms are a hybrid method for neuro-fuzzy systems based on genetic algorithms, used to find the global solution for neuro-fuzzy system parameters. They always begin by randomly generating an initial population after they encode the parameter into chromosomes. Then, they run iteratively repeating the following processes until they arrive at a predetermined ending conditions: *extracting fuzzy rules*, *self-tuning*, *fitness evaluation*, *reproduction*, and *performing genetic operators (crossover and mutation)*. It requires much computational time to construct a fuzzy system from a chromosome. The communication time, however, hardly affects the total processing time. Therefore, Tiger is suitable for executing genetic-neuro-fuzzy algorithms.

A major characteristic of our genetic-neuro-fuzzy algorithm is that the capability-based adaptive load balancing is supported to reduce total working time for obtaining an optimal fuzzy system. Such load balancing is done using Tiger's user-level global resource management tools. Let T_i be the time that is taken to execute the operations of chromosomes allocated to Region i and NF_i be the sum of the number of fuzzy rules processed in each chromosome allocated to the Region i. Then, the number of chromosomes which will be allocated to Region i at next generation, N_i, is defined by

$$N_i = N_c \cdot \frac{C_i}{\sum_{i=0}^{N_f-1} C_i} \quad , \text{ where } \quad C_i = \frac{NF_i}{T_i} \qquad (5)$$

In equation (5), N_c is the total number of chromosomes given in the system, N_f is the number of Regions currently participating in the system, and C_i is the capability, the number of fuzzy rules processed in unit time, of Regions i.

Table 2 The parameters for our genetic-neuro-fuzzy algorithms

population size	50	prob. of crossover	0.3
number of generation	50	prob. of mutation	0.15
chromosome size	48	learning iteration	50

Using equation (5), we can determine the number of chromosomes allocated to each region in the next generation, and can move some chromosomes to other regions using Tiger's APIs. The goal of our algorithm is to make a fuzzy system that can approximate the three input nonlinear functions defined by

$$output = (1 + x^{0.5} + y^{-1} + z^{-1.5})^2 \qquad (6)$$

A total of 216 training data are sampled uniformly from input ranges [1.6] × [1.6] × [1.6]. The parameters for the genetic-neuro-fuzzy algorithms used in the experiment are summarized in Table 2.

Figure 10 presents the speedup curve according to the change in the number of hosting applets, while Figure 11 shows the efficiency of capability-based adaptive load balancing scheme. In Figure 11, *NOLB* represents a scheme that does not use load balancing, while *CALB* represents the use of capability-based load balancing. We conducted the experiments on eight hosting applets and ten hosting applets, respectively. When there

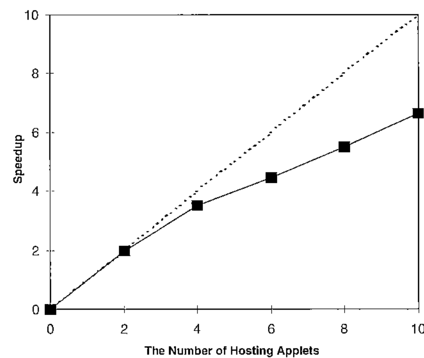

Fig. 10 Speedup for the number of hosting applets

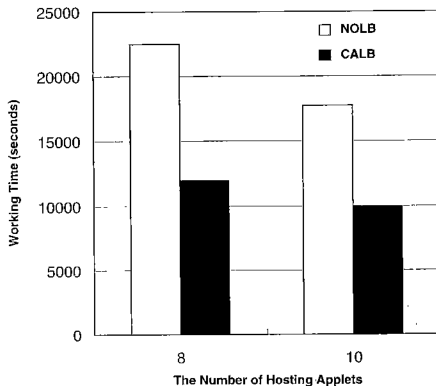

Fig. 11 The efficiency of load balancing

are 8 hosting applets, *CALB* provides performance 1.89 times better than *NOLB*. Similarly, when there are 10 hosting applets, *CALB* yields performance 1.80 times better than *NOLB*. Detailed experiments of our genetic-neuro-fuzzy algorithms using Tiger are described in [17].

7. Conclusions

We have designed and implemented Tiger, a global computing infrastructure, that is able to use computing resources of numerous machines connected to the Web. Tiger provides noble object-oriented programming libraries supporting distribution, dispatching, migration of object and concurrency among computational activities. These libraries, together with Tiger's infrastructure, allow a programmer to easily develop an object-oriented parallel and distributed application using globally extended computing resources. We did not present details of the resource management scheme here, but we believe that two parameters, OPD and CPR, are good standard measures for resource management in object-oriented global computing supporting object migration. The location management scheme is based on a two-level hierarchy such that two types of database, GLM and RLM, are suitable for keeping track of numerous mobile objects locations within globally extended computing resources.

We are currently working on an extended version of Tiger that supports a mechanism with fault tolerance, result verification and user privacy. We believe that the future version of Tiger will be a robust and high-performance global computing infrastructure.

Acknowledgements

We are thankful to our research assistants for helping with the implementation of this large-scale system, Tiger. They are Seung-Woo Lee, Min-SiK Lee and Mu-Hyun Kim, who are leading members of Crypto Club in Dept. of Mathematics in Korea University.

Reference

[1] T.E. Anderson, D.E. Culler, and D.A. Paterson, "A case for NOW(Network of and Workstations)," *IEEE Micro*, vol.15 no.1, pp. 54-64, February 1995.

[2] N.J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J.N. Seizovic, and W.S. Myrinet, "A gigabit-per-second local area network," *IEEE Micro*, vol.15 no.1, pp. 29-36, February 1995.

[3] T.M. Warschko, J.M. Blum, and W.F. Tichy, "The ParaStation project: using workstations as building blocks for parallel computing," *Proc. Intl. Conf. on PDPTA'96*, Sunnyvale, CA, pp. 375-386, August 1972.

[4] K.M. Chandy, B. Dimitrov, H. Le, J. Mandleson, M. Richardson, A. Rifkin, P.A.G. Sivilotti, W. Tanaka, and L. Weisman, "A world-wide distributed system using Java and the Internet," *Proc. of the 5th IEEE Intl. symposium on high performance distributed computing*, Syracuse, NY, August 1996.

[5] A. Baratloo, M. Karaul, H. Karl, and Z.M. Kedem, "An infrastructure for network computing with Java applets," *Proc. of ACM workshop on Java for high-performance network computing*, Palo Alto, California , February 1998.

[6] J.E. Baldeshwieler, R.D. Blumofe, and E.A. Brewer, "ATLAS : An infrastructure for global computing," *Proc. of the 7th ACM SIGOPS european workshop on system support for world wide applications*, September 1996.

[7] T. Brecht, H. Sandhu, M. Shan, and J. Talbot, "ParaWeb: Towards world-wide supercomputing," *Proc. of the 7th ACM SIGOPS european*

workshop on system support for world wide applications, pp. 181-188, September 1996.

[8] A. Alexandrov, M. Ibel, K. E. Schauser, and C. Scheiman, "SuperWeb: Research issues in Java-based global computing," *Proc. of Concurrency: Practice and Experience*, Wiley, June 1997.

[9] N. Nisan, S. London, O. Regev, N. Camiel, "Globally distributed computation over the internet - the POPCORN project," *Proc. of the 18th Int'l Conf. on distributed computing systems*, Amsterdam, Netherlands, pp. 591-602, May 1998.

[10] K.S. Leung, K.H. Lee, and Y.Y. Wong, "DJM: A global distributed virtual machine on the Internet," *Software-Prectice and Experience*, vol. 28(12), pp. 1269-1297, October 1998.

[11] A. Baratloo, M. Karaul, Z. Kedem, and P. Wyckoff, "Charlotte : metacomputing on the web," *Proc. of the 9th Int'l Conf. on Parallel and Distributed Computing Systems*, September 1996.

[12] D. Caromel, W. Klauser, J. Vayssiere, "Towards seamless computing and metacomputing in Java," *Proc. of concurrency practice and experience*, pp. 1043-1061, September 1998.

[13] M. Boger, F. Wienberg, W. Lamersdorf, "Dejay: Unifying concurrency and distribution to achieve a distributed Java," *Proc. of TOOLS99, Nancy*, France, June 1999.

[14] H. Satoshi, "HORB: Distributed execution of Java programs, Worldwide computing and its applications," *Springer Lecture Notes in Computer Science 1274*, pp. 29-42, 1997.

[15] J. Dongarra, Performance of varios computers using standard linear equations software, August 1998. (http://www.netlib.org/benchmark/performance.ps)

[16] J.D. Solomon, *Mobile IP - The Internet Unplugged*, Prentice-Hall, Inc., 1998.

[17] Y.H. Han, J.M. Gil and C.S. Hwang, "A Web based parallel framework for genetic-neuro-fuzzy algorithms," *Proc. of Int'l conf. on computational intelligent for modelling, control and automation (CIMCA'99)*, pp. 98-103, Vienna, Austria, Feb. 1999.

한 연 희

1996년 고려대학교 수학과 졸업(학사). 1998년 고려대학교 대학원 컴퓨터학과 졸업(이학석사). 1998년 ~ 현재 고려대학교 대학원 컴퓨터학과 박사과정 재학중. 관심분야는 이동컴퓨팅, 분산시스템, 분산객체 등임.

박 찬 열

1993년 고려대학교 수학과 졸업(학사). 1995년 고려대학교 대학원 컴퓨터학과 졸업(이학석사). 1995년 9월 ~ 현재 고려대학교 대학원 컴퓨터학과 박사과정 재학중. 관심분야는 분산시스템, 이동컴퓨팅, 결함허용시스템 등임.

황 종 선

1978년 Univ. of Georgia, Statistics and Computer Science 박사. 1978년 South Carolina Lander 주립대학교 조교수. 1981년 한국표준연구소 전자계산실 실장. 1995년 한국정보과학회 회장. 1982년 ~ 현재 고려대학교 컴퓨터학과 교수. 1996년 ~ 현재 고려대학교 컴퓨터과학기술대학원 원장. 관심분야는 알고리즘, 분산시스템, 데이타베이스 등임.

정 영 식

1987년 고려대학교 수학과 졸업. 1989년 고려대학교 대학원 석사학위취득(전산학). 1993년 고려대학교 대학원 박사학위취득(전산학). 1997년 1월 ~ 1998년 1월 미시간주립대학교 교환교수. 1993년 9월 ~ 현재 원광대학교 컴퓨터정보통신학부 조교수. 관심분야는 병렬분산처리, 멀티미디어 CAI, 컴퓨터 시뮬레이션 등임.