An Approach to Mobile Software Robots for the WWW

Kazuhiro Kato, Member, IEEE Computer Society, Yuuichi Someya, Katsuya Matsubara, Kunihiko Toumura, and Hirotake Abe

Abstract—This paper describes a framework for developing mobile software robots by using the PLANET mobile object system, which is characterized by language-neutral layered architecture, the native code execution of mobile objects, and asynchronous object passing. We propose an approach to implementing mobile Web search robots that takes full advantage of these characteristics, and we base our discussion of its effectiveness on experiments conducted in the Internet environment. The results show that the PLANET approach to mobile Web search robots significantly reduces the amount of data transferred via the Internet and that it enables the robots to work more efficiently than the robots in the conventional stationary scheme whenever nontrivial amounts of HTML files are processed.

Index Terms—Internet, WWW, Web, mobile object, mobile agent, search robot, indexing robot.

1 INTRODUCTION

The recent widespread use of the Internet and the World Wide Web (WWW) has had a significant effect on the whole of human culture. People who like to provide information can easily do so by simply putting it on Web servers, and the users can easily access the information by using convenient Web browsers to request the servers to send it. The procedures for providing and accessing information are extremely easy ones, but the Web is, from the viewpoint of those concerned with network resource management, not an efficient system because it is an uncontrolled system.1 If all the users of the Web were human, their uncontrollability might not be such a serious problem. Actually, however, all Web users are not human beings. Typical nonhuman users are Web search robots, which are computer programs that traverse the Web’s hypertext structure, recurrently retrieving all documents that meet specified screening criteria [16]. Unconstrained by the limits inherent to biological systems, Web search robots have the potential to request so much that they can fill the bandwidth of the Internet and use all the computing power of the Web servers.

Software robots (sometimes called “agents”) play very important roles in the Web environment, and one of their most important tasks is resource discovery. A lot of useful information is stored on Web servers throughout the world, but it cannot be accessed unless the user specifies a URL (universal resource locator). To obtain the URLs for information, we need to use an information search functionality, and software robot technology is used to build the search indices automatically. Example robot systems are the WWW Worm [20], WebCrawler [25], Lycos [19], Harvest [2], AltaVista [26], and WISE [31]. Software robot technology is also used to manage server and client hosts, to analyze statistics, to maintain the consistency among hypertext documents, and to mirror data [16], [4].

Software robots in the Web are sometimes called spiders, wanderers, or Web worms. As Koster pointed out [16], however, these names might be misleading in that terms like “spider” and “wanderer” give the false impression that the robot itself moves and that the term “worm” might imply that the robot multiplies itself, like the infamous Internet worm [16]. In the engineering field, a robot is assumed to be a mobile machine programmed to perform a number of tasks, but the Web search robots available so far are all stationary.

We claim that mobile software robot technology is a key technology by which the Web environment can be controlled without losing the virtue of freedom. This claim can be illustrated, with regard to resource discovery, as follows. Almost all conventional Web-searching robots are stationary and they connect with target Web sites from the robot-base sites by using the application layer protocol, HTTP. A robot connects with a Web server, requests an HTML document, receives it, analyzes it, makes indices from it, extracts the URLs of other sites it should access, and accesses those sites. If we could use a mobile Web search robot, the robot would move to a target Web site, obtain and analyze HTML files, and extract URLs. Then the mobile robot would either come back to the robot base or move to another Web site to obtain more information.

This paper describes a framework for developing mobile software robots by using the PLANET mobile object system. The most notable feature of PLANET is that it was designed as middleware. That is, PLANET is in the middle layer between operating systems and programming language systems and is independent of both. Another notable
feature of PLANET is that it implements mobile objects by the direct execution of native codes instead of by interpreting bytecodes. These two features distinguish PLANET from other mobile object systems, since most other mobile object systems—such as Emerald [8], Telescript [30], Aglets [17], Voyager [7], Obliq [3], and Messengers [1]—take a language-centric and bytecode-interpretation approach.

The rest of the paper is organized as follows. Section 2 discusses design issues, Section 3 is an overview of the PLANET mobile object system, Section 4 explains how mobile Web search robots can be implemented with PLANET and discusses emerging issues, Section 5 presents experimental results, and Section 6 concludes the paper.

2 DESIGN ISSUES FOR MOBILE WEB SEARCH ROBOTS

2.1 Basic Terminology

A computer host or simply host is a computer on which an operating system runs. It may have one CPU or several. A computer site or simply site is one host or a set of hosts connected by a local, large-bandwidth network, and managed under a single administration policy.

A software robot is a software object that performs a given mission autonomously. In the rest of this paper the word “robot” will mean software robot. A stationary robot cannot move among computer sites but can use network communication to access remote information resources. A mobile robot can move to the computer site containing the target information and can access the information directly or closely.

A robot base is a computer site that manages the execution of robots. In the stationary robot scheme, robots are executed within a robot base and perform their missions by communicating with remote information servers via network communication links. In the mobile robot scheme, robots are sent from a robot base to remote information servers and come back to the robot base after performing their missions.

A Web search robot is a robot that autonomously searches information accumulated in the Web servers. Fig. 1 shows the conventional (i.e., stationary) scheme for Web search robots. A Web search robot resides in a robot base and communicates with Web servers by using the HTTP protocol. All the communication between the robot and the Web servers is via the Internet. In the mobile Web search robot scheme discussed in the following sections (see Fig. 2), robots are transferred from the robot base to sites near Web servers or to Web servers themselves. There the robots collect information by using local network communication or using interprocess communication and then return to the robot base.

2.2 Advantages of Mobile Software Robots

We expect mobile object computing technology to bring at least the following advantages to the Web world.

2.2.1 Effective Use of the Internet Bandwidth

The mobile Web search robot scheme can utilize the limited Internet bandwidth more effectively than the stationary scheme. A conventional stationary Web search robot requests all the HTML files or those updated from a particular time (perhaps the last time the robot requested
files). In some case, all the HTML files obtained might be useful, but in other cases only the files satisfying some criteria might be useful. For example, a Web server may for local convenience, store a large volume of free documents (e.g., RFC documents or those for free software) available in many other Web servers. This problem is currently dealt with by voluntary collaboration between the administrators of Web servers (so-called Webmasters) and the programmers of Web search robots. A Webmaster create a special file called “robots.txt” [15], [14], [4] and in it describes which files have information useful for robots. The programmer of a Web search robot programs his robot obtain “robots.txt” from target Web servers before searching Web files. If “robots.txt” is not prepared or not properly described, the robot has to obtain all the files through the Internet. 2

If we could utilize a mobile software robot technology, it would be possible to send only useful files by screening Web files at the Web server site: a Web search robot transferred to a target Web server site would check the contents of the files stored in the Web server, taking advantage of large bandwidth of the local network of the Web-server side, and would send back to the robot base only the files that seem to be useful. Furthermore, the robot could take advantage of any other bandwidth-saving techniques, such as word-filtering, archiving, and compressing.

2 If a Web search robot is programmed to work in a periodical interactive manner, the number of files transferred each searching time could be reduced by transferring only newly created files and files modified since the last search.

2.2.2 Server-Initiative Scheduling of Web Search Robots

Conventionally, the behavior of Web search robot cannot be controlled by Web servers. This is because the servers cannot distinguish requests that are made by a human user from requests that are made by a Web search robot. A request from a human user should be serviced as fast as possible, since providing useful information to human users is indeed the objective of the Web system. The behavior of human users is restricted by human biological constraints, but software robots are free of such constraints: their behaviors depends only on the way they are programmed. If the robots are programmed by careless programmers or selfish administrators of a search-robot base, the Web servers searched might be overwhelmed by the request load due to robots.

Access requests from human users (perhaps using Web browsers) could not be postponed in general, but those from robots could be; in this sense, the former is inherently synchronous and the latter is asynchronous. Asynchronous requests are inherently schedulable, and mobile software robot technology can implement this scheduling. With the technology, a robot program itself is transferred to a Web server site and is executed under the scheduling control of the Web server site. We call this type of scheduling server-initiative scheduling of Web search robots, and we call the conventional type of scheduling client-initiative scheduling. Server-initiative scheduling allows the scheduler of the server site to execute the robot program according to the convenience of the Web server site, for example, when there are few requests from human users.
The server-initiative scheduling benefits not only the Web servers but also the search-robot bases. First, it enables the robot-computing load to be distributed among Web servers. Second, it eases the programming of Web search robots. Server-initiative scheduling makes it possible for most real-time control for “well-behaved” robots to be performed automatically by the server-side scheduler.

2.2.3 Parallel Mobile Robots
Even in the conventional stationary robot scheme, physically parallel or pseudoparallel (by using time-slice scheduling by operating systems) execution of Web search robots is performed. It might be possible for the manager of the Web search robot base site to increase the parallelism until the Internet bandwidth available to the site is used up. He would, of course, not do so intentionally but he might do so accidentally.

With the mobile robot scheme, it is possible to send robots in parallel as illustrated in Fig. 2. As discussed in Section 2.2.1, mobile robots can use the Internet bandwidth effectively; and as discussed in Section 2.2.2, server-initiative scheduling is possible. The mobile robot scheme thus enables parallel execution of mobile robots in a controlled way without requiring as much network bandwidth as conventional schemes require.

2.2.4 Extensible Service Definition for Web Search Robots
In most Internet services, protocols above the transport layer are fairly simple and statically fixed. The HTTP, NNTP, and SMTP protocols are typical examples. This has helped simplify the structure of Internet service software but has forced the use of unsuitable protocols when a more sophisticated one should be used. Despite the wide use of Web search robots, they are still forced to use the conventional HTTP protocol originally designed for Web browsers operated by human users. It is not so easy, however, to extend or customize Internet service protocols because the Internet is an open system of quite large scale: a large number of people have to agree on any extension, and achieving a consensus generally takes a long time.

The mobile object technology provides a solution to this problem. The mobile robot base creates a server-side service object that extends functions and protocols so that they are convenient to their robots, and it submits the service object to server sites to be dynamically loaded when its robots work. In the case of the Web, for example, a service object may have functions for retaining information to be used at the next search, for directly accessing the Web files by way of the operating system’s (local or remote) file access functions, or for providing higher abstractions that can be used in determining which files to access (such as, “give recently created files or modified files”).

2.2.5 Load Distribution
The mobile robot scheme distributes the robot-processing load: the computation load on a Web robot base site is transferred to a Web server site. Is this a reasonable or acceptable thing from the viewpoint of a Web server site? This question is perhaps beyond the scope of this paper, but here we present just one relevant consideration: A paradigm brought by the Internet is resource sharing: packets interchanged with the Internet protocol are passed through many computing and network resources, and people now accept the mutual sharing of the resources. This acceptance might be extended to the resource load imposed by mobile Web search robots. Consider the effect brought by the “load migration.” By sparing some computation power of a Web server site, valuable Internet bandwidth could be conserved, and the quality of Web search systems could be enhanced. This could be advantageous to everyone: the information providers, information users, information brokers managing robot bases, and all the other Internet users who share its bandwidth.

2.3 Technical Requirements

2.3.1 Sending Robots
A software robot that is represented as a software object to collect or search information in Web servers ought to be sent from a robot base to a Web server. Then it or a part of it including the collected information ought to be sent back to the robot base. Sometime a robot sent to a Web server might be requested or advised by the robot base or the Web server to visit another Web server and thus might visit several Web server sites. Object sending is similar to message passing except that an object may include program code and computational state as well as data, whereas a message generally includes only data. Hence we use the terms sending and passing interchangeably.

The easiest approach to moving objects is an interpreter-based approach. This is because interpreters inherently treat program codes, data, and computational state as internal data structures and because implementers can provide object mobility relatively easily by extending the interpreters to exchange the internal data structures between them. The approach can also deal easily with hardware heterogeneity. Most mobile object systems adopt this approach despite its inefficiency.

2.3.2 Identification
A software robot must be sent to a specific target Web server site, and the receiving Web server must know who the robot is and by whom the robot has been sent. Functions to identify robots, Web server sites, and robot base sites are therefore needed.

2.3.3 Scheduling
The scheduling issues can be classified into two kinds: robot-base-side scheduling and Web-server-side scheduling. The robot-base-side scheduler will send multiple search robots to multiple sites in parallel and will wait for the search results. Parallel execution of multiple search robots is a technique ordinarily employed in the conventional (stationary) Web search robot system, but it is limited by the CPU power of the Web base and by the network bandwidth between the robot base and the target Web servers. In the mobile Web robot approach, these limitations are overcome: mobile Web robots can be sent to a few Web server sites in parallel without being limited by the CPU power and network bandwidth.
2.3.4 Security
The security issue would be the most significant issue for the administrator of a Web server site deciding whether the site should cooperate with a mobile Web robot system. In the ordinary conventional distributed systems, all of the services provided by the servers and all of the users are fixed and the administrator of a distributed system can precisely determine them. In the Web systems, on the other hand, nobody can determine them. Thus some mechanism for the Web server to protect its resources from incorrect or malicious access by visiting Web search robots is required. And, some robot bases may not want to have the contents of their Web search robots peeked or leaked, so it might also be necessary to protect the robot itself.

2.3.5 Efficiency
Another reason for the administrator of a Web server site to hesitate to cooperate with a mobile Web robot system would be execution load due to Web search robots that may repetitively process large volumes of information stored in the Web site. Furthermore, execution performance is relevant to the quality of the retrieval of a Web search system because if a robot can execute n% faster, it can visit a target Web site n% more frequently or can visit n% more Web sites. Efficient execution of robots is thus a very important issue in designing mobile Web robot systems.

2.3.6 Interoperability
Web servers are installed on diverse kinds of computer hardware and was various operating systems. A mobile Web robot system should therefore support multiprocessor interoperability. An interpreter-based approach is a promising one, but it will sacrifice execution performance.

3 Overview of the Planet Mobile Object System
PLANET is a mobile object system designed by the authors to serve as a platform on which Internet applications can be developed, and the mobile Web search robot system we describe here is the first nontrivial application developed with PLANET.

3.1 Basic Approaches
3.1.1 Layered Architecture
Most mobile object systems require application programmers to use a brand-new, object-system-specific programming language or a particular language (such as Java) and do not support the mobility of objects written in other languages—including conventional programming languages such as C, C++, Fortran, and Pascal. Thus we can say that those systems take a language-centric approach. Why should application programmers be forced to use a programming language that might be quite different from the languages ordinarily used in the application field? Why should they be forced to give up the software libraries they usually use? PLANET takes language-neutral approach and is designed as middleware located between the operating system layer and the programming language layer. Virtually any programming language processor can be ported to the PLANET environment.

3.1.2 Native Mobile Objects
Most mobile object systems are based on the interpreter-based execution of either scripting code or bytecode, and this basis greatly simplifies the implementation of mobile objects and the management of platform-heterogeneity. PLANET, however, supports object mobility without assuming either a particular programming language design or a bytecode design. PLANET provides functions to unload a memory segment containing a mobile object from a virtual address space, to transfer that segment to another computer site, and to load it into a virtual address space there. In the transferring, conversions needed to assure object mobility are made. The only conversions supported by the current working version of PLANET is a dynamic and iterative relocation of address references (so-called pointers) that makes the memory segment loadable to any vacant portion of a virtual address space. This conversion is sufficient in a homogeneous environment, but a heterogeneous-hardware environment requires conversions of program codes, data representation, and computational state information (so-called threads). Currently, we are implementing these conversions by extending the integration techniques of virtual memory manipulation and data conversion [13], [32].

3.1.3 Asynchronous Object Passing
Most mobile object systems use a synchronous object-passing mechanism. That is, an object sender and an object receiver must be synchronized with respect to transferring an object. PLANET instead uses an asynchronous object-passing mechanism. An asynchronous one has at least two advantages, especially in the Internet environment. First, a synchronous object-passing mechanism does not allow the sender to finish the sending operation until the receiver performs the receiving operation, but in the Internet environment the receiver might not want to receive object and might not even be running at that time. The asynchronous object-passing mechanism, on the other hand, enables the sending and the receiving operations to be performed independently. The execution scheduling of the sender and the receiver thus also can be performed independently. The mobile Web search robot system described in this paper make use of this advantage. The second advantage is that the asynchronous object-passing style naturally incorporates indirect object-passing style, by which we mean that the sender sends an object through a kind of named port and does not explicitly specify the receiver and that the receiver receives the object through the port by specifying the port’s name. For the access control of objects in ports in a distributed computing environment, the capability concept can be incorporated into ports [24], [28]. The indirect object-passing style allows flexibility in implementing of a mobile object system such as a mobile Web search robot because the determination of the relationship of the sender and the receiver is deferred until the receiver performs the receiving operation.
3.1.4 Protection Domains Implemented with Virtual Address Spaces

In the language-centric interpreter-based approach taken in most other mobile object systems, protection mechanisms are relatively easy to implement. One reason for this is that the language-centric approach enables protection-violation primitives to be excluded from the language designs. Another reason is that basically all the interpretations of program statements or expressions can be checked for protection during interpretation. Because PLANET takes an approach that does not depend any particular language designs or on bytecode designs, each computer site receiving mobile objects has to be responsible for its own protection. To make this possible, PLANET utilizes virtual-memory-related techniques (available in most modern operating systems such as Solaris, FreeBSD, Linux, and Windows NT) in the following way:

- PLANET utilizes a virtual address space to implement a protection domain. The execution of native-code objects is sealed within a protection domain, and interference between protection domains is completely excluded. PLANET enables multiple mobile objects to be loaded to a virtual address space provided that the mobile objects belong to the same protection domain.
- Because of the existence of protection domains, communication between mobile objects is classified into two kinds: intra- and inter-protection-domain communications. Inter-protection-domain object communication is supported by combining inter-process communication functions provided by operating systems and stub-generation techniques in the language-layer.
- All operations acting outside a protection domain including the above-mentioned inter-protection-domain communication, ought to be performed through system calls to the operating system kernel. PLANET hooks the system call issues from each protection domain and validates the issues by using a policy module called protection domain verifier.
- PLANET adopts the remote memory-mapping architecture, which combines the virtual memory management technique and the network communication technique, to efficiently and transparently load/unload objects into/from protection domains.

3.2 Logical Structure

One major concern in designing PLANET is clear separation between logical structures and physical structures of it. The logical structure of PLANET is described using five basic abstractions: (mobile) object, place, protection domain, distributed shared repository, and object port (see Fig. 3 and Fig. 4).

3.2.1 Object

An object is an entity that may encapsulate data, the program, and the computational state (so-called thread). As illustrated in Fig. 5, objects are classified into four types according to the data segments included in the objects. A basic type segment includes data for basic data types such as integers, floating point numbers, strings, bitmap image data, etc. A structured data segment includes references (pointers) to data within the same segment. A program code segment includes codes that manipulate data in the objects. And a computational state segment includes all the information required to maintain the state of computation (i.e., a program counter, CPU registers and other CPU-state materials, and a stack).

3.2.2 Place

A place is an abstraction for the computational resources through which objects do their computation. A very simple
example of a place is a computer host with one or more CPUs and memories. Another simple example of a place is a LAN that has several computer hosts.

3.2.3 Protection Domain

Since a place may be visited by a number of inherently enigmatic or malicious objects, it must have a protection mechanism. The protection mechanism should have at least two functionalities: the computational resources of a place must not be affected by accidental or intentional violations of access rights, and the objects and activities in a place must be protected against illegal access by each other. A protection domain is an abstraction for controlled object accesses. When an object is loaded into a place, one protection domain has to be specified; then, if no access right violation is determined by the system, the object is loaded to the protection domain. Any number of objects can be loaded to a protection domain provided that access right violation does not occur and enough computational resources are available for it. The objects loaded to a protection domain exist in the protection domain until they are explicitly unloaded from it. The system grants access rights on a protection-domain basis, and the access rights granted to a protection domain are shared by all the objects within it. The level of protection is uniform within a protection domain, so object interaction within a single protection domain does not require domain-switching and thus can be executed efficiently.

Each protection domain is associated with a protection-domain verifier that checks to see whether the system calls issued from the domain are in accordance with the protection policy set by the administrator of the place or by the owner user of the protection domain. Only calls that have been validated are actually issued.

3.2.4 Distributed Shared Repository

The distributed shared repository (DSR) [12] is an abstraction for the worldwide persistent object store: objects put in the
DSR become accessible from all places and are guaranteed to persist until they are explicitly removed. Because all the load and unload operations are performed between protection domains and the DSR, the sender of an object can send the object regardless of the receiver’s state. Similarly, the receiver can receive an object regardless of this sender’s state. The naming and access control functions of the DSR are provided by using object ports.

3.2.5 Object Port
An object port is a named and access-controlled queue for objects in the DSR (Fig. 4). When a sender unloads an object to the DSR, the sender must specify an object port into which the unloaded object is to be placed and must have a capability for write-access to that object port. Similarly, when a receiver loads an object from the DSR, the receiver must specify the object port from which the loaded object is to be obtained and must have a read-access capability for that object port. A capability is obtained by specifying a location-independent name given to an object port and by authenticating the sender or receiver.

3.3 Physical Structure
The logical structure of PLANET is fairly simple, but the physical structure of our current implementation is rather sophisticated. The current implementation runs on SPARC workstations and the Solaris 2.5.x operating system. It is written in about 40,000 lines of C program code and two hundred lines of the SPARC assembly program code.

Fig. 6 shows a typical physical structure for PLANET. A place consists of one or more client sites, and a client site has protection domains, an object cache, and a name cache. A protection domain is implemented as a virtual address space. The object cache is used by the remote memory-mapping mechanism. The name cache is used by the system to map the logical names of object ports to the corresponding physical names efficiently. To implement the name mapping we extensively use the prefix table technique [29] to make the mapping scalable to the number of client sites.

The DSR is logically one space but is physically composed of multiple servers. In Fig. 6, one DSR server is connected to the LAN of Place A and the other two DSR servers are connected to the Internet. Since the name space for the DSR is unique and location-independent, the connection topology of the client sites and DSR servers do not affect the logical name specifications.

In PLANET, objects are loaded to protection domains and unloaded to object ports in the DSR through the use of a remote memory-mapping mechanism, an extension of the memory-mapping file access mechanism for network environments. A memory-mapping file access mechanism is a combination of file I/O and demand-paged virtual-memory access: Rather than offering read and write operations, a memory-mapping interface provides access to disk storage via an array of bytes in main memory (see Chapter 12 in [27]). The mechanism has several advantages in the implementation of mobile objects. One is that communication between a place client and a DSR server is implicit: Once an object stored on a DSR server site has been specified to be loaded to a virtual address space that implements a protection domain at a place client site, each portion of the object is transferred when the portion is accessed by the CPU on the virtual address space. The transfer is done automatically by PLANET’s runtime system, and it is not necessary for programmers to control the transfer. A second advantage is that network communication...
is minimized, since the portions transferred to a client site are locally cached and subsequent access to the portions does not require data communication. Furthermore, only the modified portion is written back to the DSR server. Still another advantage is that the same representation of objects can be used both when an object is loaded to a protection domain and when it is unloaded to the DSR. Finally, the problem of double buffering between a user address space and the kernel space is avoided as a result of the local memory-mapping mechanism. More details of the remote memory-mapping mechanism of PLANET are in [18].

4 IMPLEMENTING MOBILE WEB SEARCH ROBOTS WITH PLANET

4.1 Basic Scheme

Fig. 7 illustrates the basic scheme of a mobile Web search robot system designed with PLANET. Remember that a site (or place) may consist of a single computer or a set of computers interconnected by a network with a large bandwidth. In the following description of the basic scheme we use “object” and “robot” interchangeably.

1. **Sending a robot.** The robot base site (“robot base”) obtains an object-sending right to its corresponding object port of the target Web server (b could be an open name declared by each Web server site). Then, the robot base sends a Web-searching robot to the object port b.

2. **Receiving a robot.** When the Web server wants to provide a Web search service, it receives a Web search robot from the object port a.

3. **Obtaining Web information.** The Web search robot executes in a protection domain prepared by the Web server site and obtains Web information (contents). The Web search robot may perform data processing such as filtering, archiving, and compressing.

4. **Return the result.** After the Web search robot collects sufficient information (or some precondition is satisfied), the robot is unloaded and sent back to the object port a (the robot knows the name in
Obtaining the result. The robot base receives the robot (or the data part of it) from object port \(a\).

The DSR shown in Fig. 7 has two servers, and they can be located either in one host or in different hosts in different places. The administrator of the Web server can implicitly specify which DSR server should be used to receive a robot by setting the object port \(a\) or \(b\) in Fig. 7.

**4.2 Introducing Web Search Server and Robot Suspension/Reactivation**

We can, as shown in Fig. 8, extend the basic scheme to incorporate a search server in a Web server site. A Web search server stationed in a Web server site can provide Web-information-providing service to visiting Web robots. The reason for introducing a Web search server is to provide higher-level services than those that can be provided with only a Web server. As noted in Section 2.2, the HTTP protocol and HTTP servers were originally designed to work with Web browsers manipulated by human users and are thus not particularly suitable for use with Web search robots. By introducing a Web search server, we can easily extend the functionality of Web servers. Typical uses of a Web search server are as follows:

- A search server interacts with the Web server in the Web server site through the standard HTTP protocol periodically and then calculate the “differences” between the periods. The difference information may include information about the creation, modification, and deletion of the Web contents. When a robot visits the Web server site, the search server asks the robot when the robot had last visited the site and passes only the contents that have changed since then. More generally, a visiting Web search robot may leave in a search server some information that will be used the next time the robot visits.
- If the Web search server and the Web server are located in the same host, the Web search server can access the Web contents files directly. This bypasses the Web server and thus can be expected to significantly increase the speed with which Web contents are accessed. Furthermore, the search server can utilize the meta-information (create/update/access dates, file size, owners, etc.) provided by the file system. Even if the search server and Web server are not located in the same host, the search server might obtain similar advantages by using a distributed file system such as the NFS.

Another technique, illustrated in Steps 3 and 4 of Fig. 8, is the suspension/reactivation of robots. Suspension means that the execution of a robot is stopped and it as well as its state is stored in the DSR. Reactivation means that the stored robot is loaded and started again. Typical uses of this technique are when the robot is suspended by the Web server site so that the system can be shutdown or simply to decrease the system load. Another typical use of the technique is in the implementation of a “semimobile” robot, one that is sent to a Web server site, remains there, and performs its work periodically, thus eliminating the overhead of transferring the robot.
4.3 Forwarding and Parallel Search

The use of a mobile Web search robots may concern multiple Web server sites. One typical pattern is the serial navigation shown in Fig. 9 and another is the parallel navigation shown in Fig. 10. The serial scheme might be required when the information regarding the site to be visited is obtained only dynamically (i.e., after the robot is sent from the robot base). If the information is known statically (i.e., before the robot is sent), the parallel scheme could be used. By combining these patterns, complex navigation patterns can be organized.

5 EXPERIMENTS

We examined the effectiveness of our mobile Web search robot scheme experimentally. Section 5.1 describes the experimental environment, Section 5.2 compares the mobile and stationary schemes, and Section 5.3 compares the
native-code execution approach taken in PLANET with bytecode-interpreter and just-in-time-compiler approach taken in many other mobile object systems. We use as a typical system a Java-based mobile object system. Lastly, Section 5.4 shows the effect of parallel computing with mobile Web search robots.

5.1 Experimental Environment

The experimental environment consisted of Internet nodes of the University of Tsukuba, the University of Tokyo, and Kyoto University (see Fig. 11). Fig. 12 shows the effective network bandwidth and the straight line distance between each pair of computer sites (i.e., Tsukuba-1, Tsukuba-2, Tokyo-1, and Kyoto-1). SINET [6] in Fig. 11 is the Science Information Network managed by the National Center for Science Information Systems of the Ministry of Education, Science and Culture of Japan. It played the central role of the Internet in the experiments.

We implemented a mobile Web robot system with PLANET according to the scheme described in Section 3. We installed PLANET on the experimental environment as shown in Fig. 13. In the installation, there were three computing hosts: Kyoto-1, Tsukuba-1, and Tsukuba-2. Kyoto-1 was a robot base and contained not only a protection domain for the robot but also a DSR server including an object port named a.3 Tsukuba-1 was a Web server site and contained protection domains for robots sent from robot bases and also contained a DSR server including an object port named b. Tsukuba-2 was another Web server site and contained a Web server (Apache 1.3.1) executed as an ordinary

---

3. In general, a client and a server of PLANET are not necessarily located in the same computer host.
Unix processes. The search server communicated with the Web server using the standard HTTP protocol to obtain the Web contents.

The specification of the platforms of the computer sites are listed in Table 1. The robot base site (Kyoto-1) has the most powerful CPU and large main memory. Thus this is advantageous to the conventional stationary scheme since in the mobile scheme the robot is executed in less powerful Web server sites.

To describe the program of the mobile object implementing the Web search robot, we used the PLANET/C++ programming language. This is a porting of the C++ programming language to the PLANET environment. The language, two mobile objects in the same place to communicate with each other without regarding the existence of the protection domain barrier. That is, mobile objects loaded to the same protection domain and those loaded to different protection domains can communicate each other without modifying the program code [11], [9]. The language processor of PLANET/C++ is implemented using Open C++ [5] (version 2.5b1). The processor translates a user program written in PLANET/C++ into a C++ program that issues local method calls, local/remote interprocess communications, and PLANET’s system primitives. Then the generated program is translated into an executable code with an ordinary C++ compiler. The C++ compiler we used in the experiments was a GNU C Compiler (gcc version 2.8.0), and we specified the default compiler-optimization setting.

Using PLANET/C++, we described a simple mobile Web search robot system as follows (see Fig. 13):

- **Step A: Send a Robot.** The robot base site (Kyoto-1) sends a robot to the Web server site (Tsukuba-1) via the object port named b.
- **Step B: Obtain Files.** The robot loaded to the protection domain for a visiting robot in the Web server site obtains HTML files by communicating with the Web search server (see Fig. 14). The search server obtains the requested files through the HTTP protocol. The protection-domain verifier associated with the protection domain for the

Fig. 13. The experimental environment for PLANET.
visiting robot assures, by watching the system calls issued from the protection domain, and that the visiting robot communicates with only the Web search server.

- **Step C: Process Files.** For each obtained file the robot extracts URLs to find those pointing the files stored in the same Web server and recurrently obtains the files corresponding to the found URLs until the robot obtains the specified amount of HTML files. The initial “root” URL is given to the robot as an initial value by the robot base. All the collected files are archived into one file.

- **Step D: Send Back the Robot.** The robot containing the obtained set of URLs and HTML files is sent back to the Web robot base.

---

**TABLE 1**
The Platforms Used in the Experiments

<table>
<thead>
<tr>
<th></th>
<th>robot base site</th>
<th>Web server site</th>
<th>Web server site</th>
</tr>
</thead>
<tbody>
<tr>
<td>host name</td>
<td><em>Kyoto-1</em></td>
<td><em>Tsukuba-1</em></td>
<td><em>Tsukuba-2</em></td>
</tr>
<tr>
<td>machine</td>
<td>Sun Ultra-60</td>
<td>Sun Ultra-30</td>
<td>Sun Ultra-30</td>
</tr>
<tr>
<td>CPU name</td>
<td>UltraSPARC-II</td>
<td>UltraSPARC-II</td>
<td>UltraSPARC-II</td>
</tr>
<tr>
<td>CPU clock (MHz)</td>
<td>360</td>
<td>296</td>
<td>296</td>
</tr>
<tr>
<td>SPECint95</td>
<td>16.1</td>
<td>12.1</td>
<td>12.1</td>
</tr>
<tr>
<td>SPECfp95</td>
<td>23.5</td>
<td>18.3</td>
<td>18.3</td>
</tr>
<tr>
<td>main memory size (MB)</td>
<td>640</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>operating system</td>
<td>Solaris 2.6</td>
<td>Solaris 2.5.1</td>
<td>Solaris 2.5.1</td>
</tr>
</tbody>
</table>

---

*Fig. 14. The experimental environment for a stationary robot. The robot base obtains 26.9 MB\text{Bytes} HTML files through the Internet by using the HTTP protocol.*

---

5. In the experiment, all the system calls (except for `getmsg` and `write`) were hooked. The `getmsg` and `write` system calls, which were issued a lot of times to communicate with the Web search server, were safe because such calls can be issued only to the previously checked communication ports.
The implementation of PLANET we used incorporates data-compression into its remote memory-mapping mechanism, and the GZIP algorithm is used in the current implementation of PLANET. The compression and uncompression are performed automatically by PLANET and are transparent to the programmers. All the messages used for communications between every protection domain and the DSR are in a compressed data representation, and all the objects stored in a DSR are in the compressed form.

To compare the mobile and stationary scheme, we performed the similar computation based on the conventional stationary scheme. In other words, the stationary robot performed only Steps B and C. Another difference was that the robot remained in the robot base site. The robot was implemented as an ordinary Unix process, and thus there were no protection-domain-related issues. Two more differences were that all files were obtained via the Internet and that data compression was not used. The settings are quite similar to those in the common conventional Web robot base sites. The program for the stationary robot was written in C++ and was compiled with the above-mentioned C++ compiler, again with the default compiler-optimization setting specified.

5.2 Mobile Robot vs. Conventional Stationary Robot

Now, we present our experimental results on comparing the mobile scheme and the conventional stationary scheme to implement Web search robots.

5.2.1 Reduced Network Communication via the Internet

First, we show how the amount of Internet communication was reduced by the mobile scheme. We measured the total amount of communication packets transferred, via the Internet between the Web robot base site and the Web server site in order to obtain 26.9, 13.6, and 2.7 MBytes of Web content. The amounts, counted by the number of bytes transferred as IP packets, are shown in Fig. 15, Fig. 16, and Fig. 17. The amounts transferred by the PLANET mobile Web search robot were 1/2.24, 1/1.89, and 1/1.15 of those transferred by the stationary robot, respectively. This savings are mainly due to the effect of the data compression mechanism of PLANET. For example, 26.9 MBytes of HTML files were reduced to 5.9 MBytes. That is, the compression ratio was 1/4.56. Another reason for the reduction of the amount was the reduction of the number of transferred files: In the
conventional stationary scheme, a lot of HTML files were transferred, but in the mobile scheme very few files (including the Web robot itself and the archived result file) were transferred. The reduction of the number of transferred files helps reduce the amount of file-accessing information transferred and thus also the total number of bytes transferred. When smaller amounts of HTML files are transferred (see Fig. 17), the overhead for sending robots in the mobile scheme becomes relatively large.

In the experiment, the robot did not perform information filtering. If it had, however, the mobile scheme would have shown further improvement in performance, since the mobile scheme would enable the filtering to be performed in the Web server site and would thus reduce the number of bytes transferred via the Internet.

5.2.2 Efficiency of Robot Execution

We also compared the processing time of Web search robots. For each scheme, we measured the time from the beginning until the result was obtained by the Web robot base site. Note that the physical behavior of the two schemes were slightly different because of the logical and physical structures of PLANET; the DSR of PLANET incorporates the functionality of making objects (including the obtained results) persistent, and is implemented with the remote memory-mapping technique—thus, sending robots to object ports \( a \) and \( b \) in DSR caused disk I/O. In the experiments of the mobile scheme, Disk-I/O were required when: i. unloading from the robot base protection domain to object port \( a \); ii. loading from object port \( b \) to the protection domain for a visiting robot; iii. unloading from the protection domain to object port \( a \); and iv. loading from object port \( a \) to the robot base protection domain, though iv. does not cause actual I/O since the returned result was not “touched” in the robot base protection domain in the experiment. Such extra-I/O were not required in the stationary scheme.

We conducted the experiments in which the obtained HTML file sizes were 26.9, 13.6, and 2.7 MBytes. The results are listed in Table 2A, Table 2B, and Table 2C and are illustrated in the first and second bars in Fig. 18, Fig. 19, and Fig. 20, respectively. These are stacked bar graphs and each stacking element corresponds to the processing time spent in performing Steps A-D. Note that the computer performance of host \( \text{Kyoto}-1 \) for the Web robot base site and host \( \text{Tsukuba}-1 \) for the robot execution in the Web server site is not the same, and that the former one is 1.3 times faster and has five times as much main memory (Table 1). If we could use the \( \text{Kyoto}-1 \) machine in \( \text{Tsukuba}-1 \), the processing time of the PLANET mobile robot would be even shorter.

The time for sending robot in Step A was the extra-time cost necessary only in the mobile scheme. This required 14, 21, and 24 percent of each total time. When larger HTML files (i.e., 26.9 or 13.6 MBytes) were obtained, the extra-time cost was compensated for by the reduced time-cost of Steps B–D. When the smallest (2.7 MBytes) HTML files were obtained, it was not. This is the main reason the mobile scheme shows better performance than the stationary scheme does in Fig. 18 and Fig. 19, but not in Fig. 20. There are two basic reasons for the compensation. First, in Step B, the mobile robot can obtain HTML files via the local network within the Web server sites rather than via the Internet. This shortens the processing time because communication via a local network is usually much faster than that via the Internet. Indeed, we measured the effective network bandwidths (throughputs) of the local network and the Internet used in the experiments and found that the effective network bandwidth between the Web robot base site in Kyoto University and the Web server site in University of Tsukuba was 2.2 Mbps, whereas that within the Web server site was 88.4 Mbps. Thus, the local network communication was about 40 times faster in the experiment, and this greatly reduced the time-cost of obtaining the HTML files. Second, the data-compression mechanism of PLANET effectively reduced the number of bytes transferred via Internet in Step D as discussed in Section 5.2.1, and this also effectively reduced the time-cost.

6. If the host used for the Web robot base in the stationary scheme had had less memory, I/O would have been necessary to receive the large results.
5.3 Native Code Approach vs. Bytecode Interpreter and JIT-Compiler Approaches

To compare the native-code-based approach for object mobility used in PLANET and the interpreter-based approach used in most other mobile object systems, we implemented a mobile Web robot system with Java. We used Java for the following three reasons. First, it has an most advanced and sophisticated bytecode interpreter and a just-in-time (JIT) compiler. Second, it is being widely used to implement network-related and Web-related applications in Internet and intranet environments. Third, Java itself has class (program code) passing functionality and is used by many researchers as a basis for implementation of mobile object systems (see Fig. 21).

Fig. 18. The processing times for Web search robots obtaining 26.9 MBytes HTML files. In the PLANET mobile robot (I), the compression time-cost was included in Step D since the current implementation of PLANET incorporate data-compression into the remote-memory mapping mechanism. This is also true of the results shown in Fig. 19 and Fig. 20.

TABLE 2A
Processing Time (Seconds) that Web Search Robots Required for Obtaining HTML Files, Corresponding to Fig. 18 (The Numbers in Parentheses are Standard Deviations)

<table>
<thead>
<tr>
<th></th>
<th>I. Planet mobile robot</th>
<th>II. conventional stationary robot</th>
<th>III. Java mobile (interpreter)</th>
<th>IV. Java mobile robot (JIT compiler)</th>
<th>V. Java mobile robot (JIT compiler, no compression)</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.9-MBytes files (corresponding to Fig. 18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A. send a robot</td>
<td>28.02 (0.91)</td>
<td>-</td>
<td>0.99 (0.52)</td>
<td>0.59 (0.41)</td>
<td>1.24 (0.26)</td>
</tr>
<tr>
<td>B. obtain files</td>
<td>32.19 (1.51)</td>
<td>267.17 (5.74)</td>
<td>139.95 (9.73)</td>
<td>132.50 (10.65)</td>
<td>129.06 (7.33)</td>
</tr>
<tr>
<td>C. process files</td>
<td>20.98 (0.42)</td>
<td>10.65 (0.18)</td>
<td>483.70 (14.99)</td>
<td>232.44 (37.18)</td>
<td>249.87 (19.92)</td>
</tr>
<tr>
<td>C’. compress files</td>
<td>-</td>
<td>-</td>
<td>27.58 (1.01)</td>
<td>20.25 (1.34)</td>
<td>-</td>
</tr>
<tr>
<td>D. send back the result</td>
<td>38.64 (1.63)</td>
<td>-</td>
<td>26.65 (1.04)</td>
<td>21.92 (0.90)</td>
<td>165.47 (42.35)</td>
</tr>
<tr>
<td>total</td>
<td>119.83 (2.27)</td>
<td>277.82 (5.90)</td>
<td>678.89 (25.57)</td>
<td>407.71 (41.95)</td>
<td>545.64 (37.97)</td>
</tr>
</tbody>
</table>
We implemented the logical structure of PLANET described in Section 3.2 with Voyager [23], [7] (version 1.0.1) (see Fig. 21). Voyager is developed by ObjectSpace Inc. and is one of the most sophisticated and easy-to-use Java-based mobile object systems. Voyager’s object-passing model is based on a synchronous one, so we emulated PLANET’s asynchronous object-passing model by representing an object port with a Unix process including a Java virtual machine.\footnote{Voyager’s object-passing model is based on a synchronous one, so we emulated PLANET’s asynchronous object-passing model by representing an object port with a Unix process including a Java virtual machine.}

8. This technique is similar to the well-known technique used to emulate an asynchronous message-passing model in the framework of a synchronous message-passing model.

We implemented the logical structure of PLANET described in Section 3.2 with Voyager [23], [7] (version 1.0.1) (see Fig. 21). Voyager is developed by ObjectSpace Inc. and is one of the most sophisticated and easy-to-use Java-based mobile object systems.\footnote{Voyager’s object-passing model is based on a synchronous one, so we emulated PLANET’s asynchronous object-passing model by representing an object port with a Unix process including a Java virtual machine.} Voyager’s object-passing model is based on a synchronous one, so we emulated PLANET’s asynchronous object-passing model by representing an object port with a Unix process including a Java virtual machine.\footnote{This technique is similar to the well-known technique used to emulate an asynchronous message-passing model in the framework of a synchronous message-passing model.}

### TABLE 2B
Processing Time (Seconds) that Web Search Robots Required for Obtaining HTML Files, Corresponding to Fig. 19 (The Numbers in Parentheses are Standard Deviations)

<table>
<thead>
<tr>
<th></th>
<th>I. Planet mobile robot</th>
<th>II. conventional stationary robot</th>
<th>III. Java mobile robot (interpreter)</th>
<th>IV. Java mobile robot (JIT compiler)</th>
<th>V. Java mobile robot (JIT compiler, no compression)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.6-MBytes files (corresponding to Fig. 19)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A. send a robot</td>
<td>15.24 (0.91)</td>
<td>-</td>
<td>0.82 (0.51)</td>
<td>0.58 (0.40)</td>
<td>0.63 (0.38)</td>
</tr>
<tr>
<td>B. obtain files</td>
<td>14.65 (0.42)</td>
<td>98.96 (1.33)</td>
<td>51.78 (3.99)</td>
<td>50.17 (4.47)</td>
<td>46.60 (1.13)</td>
</tr>
<tr>
<td>C. process files</td>
<td>11.59 (0.33)</td>
<td>4.78 (0.15)</td>
<td>207.97 (5.60)</td>
<td>102.13 (4.59)</td>
<td>103.65 (4.20)</td>
</tr>
<tr>
<td>C’. compress files</td>
<td>-</td>
<td>-</td>
<td>13.07 (0.71)</td>
<td>11.27 (0.70)</td>
<td>-</td>
</tr>
<tr>
<td>D. send back the result</td>
<td>23.19 (1.10)</td>
<td>-</td>
<td>11.51 (0.77)</td>
<td>10.30 (0.53)</td>
<td>60.11 (9.82)</td>
</tr>
<tr>
<td>total</td>
<td>64.67 (2.99)</td>
<td>103.75 (1.35)</td>
<td>285.16 (10.56)</td>
<td>174.45 (9.83)</td>
<td>210.78 (7.58)</td>
</tr>
</tbody>
</table>

Fig. 19. The processing times for Web search robots obtaining 13.6 MBytes HTML files.
Voyager passes objects through the ordinary TCP/IP link, so it behaves quite unlike from PLANET, which passes objects by using the remote memory-mapping technique. Thus, sending an object to an object port causes disk I/O in PLANET but does not in the emulation with Java and Voyager. We implemented an experimental environment similar to that shown in Fig. 13 using the same platform specifications listed in Table 1. The programs were compiled with the compiler of JDK 1.1.6 [21] and executed with the bytecode interpreter of JDK 1.1.6 and with the JIT compiler of JRE 1.1.6 [22]. We used the DEFLATE data-compression algorithm provided in the standard library (java.util.zip.Deflater) of JDK 1.1.6. (The process of data

### TABLE 2C
Processing Time (Seconds) that Web Search Robots Required for Obtaining HTML Files, Corresponding to Fig. 20 (The Numbers in Parentheses are Standard Deviations)

<table>
<thead>
<tr>
<th></th>
<th>I. Planet mobile robot</th>
<th>II. conventional stationary robot</th>
<th>III. Java mobile robot (interpreter)</th>
<th>IV. Java mobile robot (JIT compiler)</th>
<th>V. Java mobile robot (JIT compiler, no compression)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. send a robot</td>
<td>4.64 (0.35)</td>
<td>-</td>
<td>1.04 (0.84)</td>
<td>0.57 (0.39)</td>
<td>0.81 (0.80)</td>
</tr>
<tr>
<td>B. obtain files</td>
<td>2.60 (0.40)</td>
<td>17.64 (0.44)</td>
<td>7.58 (0.61)</td>
<td>7.34 (0.66)</td>
<td>6.69 (0.44)</td>
</tr>
<tr>
<td>C. process files</td>
<td>5.90 (0.61)</td>
<td>1.19 (0.20)</td>
<td>40.02 (0.37)</td>
<td>18.73 (0.54)</td>
<td>18.87 (0.18)</td>
</tr>
<tr>
<td>C’. compress files</td>
<td>-</td>
<td>-</td>
<td>2.67 (0.27)</td>
<td>2.58 (0.21)</td>
<td>-</td>
</tr>
<tr>
<td>D. send back the result</td>
<td>11.54 (1.04)</td>
<td>-</td>
<td>2.37 (0.39)</td>
<td>2.30 (0.34)</td>
<td>8.37 (0.74)</td>
</tr>
<tr>
<td>total</td>
<td>24.68 (1.73)</td>
<td>18.83 (0.57)</td>
<td>53.68 (2.18)</td>
<td>31.51 (1.94)</td>
<td>34.75 (2.00)</td>
</tr>
</tbody>
</table>

Fig. 20. The processing times for Web search robots obtaining 2.7 MBytes HTML files.
compression is referred to Step C’ in the following.) To minimize the overhead in string manipulation, we used StringBuffer-class objects to edit strings.

Using this environment, we conducted experiments similar to those described in Section 5.1. For the execution of Java byte codes, we used the bytecode interpreter and JIT compiler. To examine the effect of data-compression, we also conducted experiments in which data-compression was omitted and the bytecode was executed with the JIT compiler. The results are shown by the third, fourth, and fifth bars in Fig. 18, Fig. 19, and Fig. 20.

A result immediately seen from the figures is that the Java-based mobile Web robot system was inferior, with regard to processing time, to the conventional stationary scheme in all the cases. Since in the experiment the hosts for the Web server site (Tsukuba-1 and Tsukuba-2) were less powerful than the host for the Web robot site (Kyoto-1) the execution performance of both the Java and PLANET mobile robots could be improved if we could use more powerful machines in the Web server site. Although the Java-based approach is slower, it is advantageous in that it uses less of the Internet bandwidth.

Now, we discuss the experimental results comparing PLANET’s native-code-based approach with the Java-based bytecode-based approach. The processing time of PLANET’s approach is, as shown in Fig. 18 and Fig. 19, much faster when the larger HTML files are obtained, but PLANET approach is only slightly better than the Java JIT-compiler approach when small HTML files are obtained as shown in Fig. 20. The reasons for this can be understood by examining the time for every step. The PLANET approach required more time for sending the robot in Step A and sending it back in Step D than does the Java approach. This is because the size of the PLANET robot is greater than that of the Java robots, and the current implementation of the remote memory-mapping mechanism of PLANET requires nontrivial overhead for initialization and finishing. The time needed for obtaining files in Step B is almost the same for both approaches. This is because the processing is communication-intensive, hence the time difference between the native-code execution and the bytecode execution did not appear. The time to process files in Step B and C was the most major source of the processing time difference. Step B is communication-intensive processing and Step C is CPU-intensive processing. The reason of slowness of Java’s communication observed in Step B would be the consequence of the platform-independent I/O model and I/O library of Java, and the slowness of Java in Step C would be the consequence of the bytecode approach. Thus, we can say that the Web search robot was both communication-intensive and computation-intensive, and the slower computation and communication speed of Java bytecode (compared with those of the native code) increased the total execution time of the robot.

Many efforts are currently being devoted to speeding the execution of Java bytecodes, and its execution speed is therefore expected to be gradually improved. Java, however, seems to have a handicap with regard to Web search computing, which requires a lot of string processing. The disadvantage is that Java has to treat a string as an object and inhibits direct accesses to the contents of strings because of its language design principle. With the C or C++ languages, on the other hand, the programmer can access the contents of strings directly and perform string-related operations efficiently.
because in those languages a string is represented by a simple byte sequence on memory and the languages do not inhibit direct memory accesses. Of course, if we could write string processing in a special way—for example, using special library for string manipulation or using a specially tuned Java virtual machine—the performance might be much improved. 9

5.4 Parallel Computation with Mobile Web Search Robots

The mobile object computing can naturally incorporate parallelism, as suggested by Fig. 2 and as discussed in Section 2.2 and Section 4.3. We, therefore, conducted a simple experiment in order to demonstrate the effect of parallel computation with PLANET mobile robots. We added host Tokyo-1 to the experimental environment shown in Fig. 13. The configuration of the added site was similar to that of the University of Tsukuba site except that it consisted of only one host (Tokyo-1) and all the components (the DSR server, protection domains for visiting robots and the search server, and the Web server) were located in the host. See Table 1 for the specifications of the platform of the host. The Web contents stored in Tsukuba-1 and Tokyo-1 were the same.

Using this expanded environments, we conducted a parallel experiment. The robot base site sent robots in parallel to both Web server sites. Each robot performed the same computations performed by the single mobile robot in the experiment described in Section 5.1 and obtained 26.9 MBytes HTML files. The results are shown in Fig. 22 in the form of a time chart. Generally speaking, drawing a time chart for processing in a distributed environment is hard because adjusting the internal clocks of the hosts is very hard. So, we reconstructed the distributed computing by using the log that records the times when each event occurs and analyzing the causality between the events. The time chart in the figure was drawn as follows. The time 0.0 in the horizontal axis represents the (virtual) time when the program at the robot base site started. We thus see that the base site send robots to the University of Tsukuba and the University of Tokyo at 0.63 and 5.20, respectively. And the log in the site at the University of Tokyo revealed that obtaining and processing files in the site required 184.56 sec and that the sending the robot back required 40.78 sec. Using the causality of the robot-sending and the timing data, we can write the lower half of the figure. The upper half can be drawn similarly.

From Fig. 22, we can clearly see the parallel and overlapped execution of the two mobile robots. Parallel execution of robots is performed even in a conventional stationary robot system, but the execution speed is limited by the CPU power of the robot base site and the effective bandwidth of nearby network. Thus the use of parallel mobile Web search robots has the potential to break through these limits and provides better performance while using less network bandwidth.

6 CONCLUSION

We have explained the reasons for using mobile object technologies to implement Web search robots and have described our approach using the PLANET mobile object system. This system is characterized by language-neutral layered architecture, native code execution of mobile objects, asynchronous object passing via DSR, and

9. There are several implementations of native-code generation compiler for Java, but such implementations lose the advantage of multiplatform interoperability of bytecodes.
protection domains implemented with virtual address spaces. We discussed the design issues that need to be addressed when using mobile object technologies and we examined the effects of these technologies by conducting in the Internet environment. The results show that the PLANET approach to mobile Web search robots significantly reduces the amount of data transferred via the Internet and that it enables the robots to work more efficiently than the robots in the conventional stationary scheme whenever nontrivial amounts of HTML files are processed. We also showed the results of experiments comparing PLANET's binary-code-based approach with Java's bytecode-based approach. The readymade multiplatform-interoperability of Java is attractive, but the execution performance of Java-based mobile Web search robots was not as good as that of PLANET-based robots when larger amounts of HTML files were processed. Improvements in JIT-compilation and efficient string-handling technique in Java might soon improve this situation.

The design of a mobile Web robot system raises many interesting research issues that we shall tackle. First, a robot base may send a few robots simultaneously and we should therefore study the ways that a robot base should schedule and control the parallelism of its Web robots. Second, because a Web server can be visited by a few robots simultaneously, we should study the effective scheduling of robot execution in a Web server. Third, the Web search server objects in the experiments described in Section 5 were not designed to be intelligent, but these kinds of objects can in principle be made more intelligent and efficient, by caching some information [10] on the Web server, for example, or by providing higher-level abstractions. Fourth, Web search robots might be given the ability to collaborate with each other and exchange their collected information. The DSR could become an effective tool for such a functionality.

To become a more useful platform, PLANET should be enhanced in the following aspects. First, it should support interoperability among heterogeneous platforms. Second, the execution speed of its object-passing mechanism based on remote memory-mapping technique should be increased. A promising way to do this is to incorporate the prefetching technique into the remote memory-mapping technique. Third, a method to ease the description of protection-domain verifiers should be developed.

We have already begun work on these enhancements, and we are developing a parallel, distributed, and mobile Web search system. The most recent information about PLANET and the Web search system is available at http://www.osss.is.tsukuba.ac.jp/~planet/.

ACKNOWLEDGMENTS
We thank Professor Akinori Yonezawa and Dr. Kenjiro Taura of the University of Tokyo and Professor Atsushi Ohori of Kyoto University for cooperating with us in conducting the experiments. We would also like to thank Shigeru Chiba of the University of Tsukuba who developed Open C++ [5], a useful extensible C++ language processor. We used Open C++ to implement PLANET/C++ and it contributed to shortening the development time. We appreciate the comments, criticisms, and suggestions made by the anonymous reviewers of the earlier version of the paper. They were very helpful.

This work was partly supported by Precursory Research for Embryonic Science and Technology of the Japan Science and Technology Corporation; the Parallel and Distributed Consortium (Category for Programming Languages); the Support Center for Advanced Telecommunications Technology Research Foundation; the Advanced Database Systems for Integration of Media and User Environments of Grant-in-Aid for Scientific Research on Priority Areas of the Ministry of Education, Science, Sports and Culture; and the Wide-Area Distributed Programming System Based on Mobile Object Computing of Grant-in-Aid for Scientific Research (A) of the same ministry.

REFERENCES

Kazuhiko Kato received his BE and ME degrees from the University of Tsukuba, Japan, in 1985 and 1987, respectively; and his DSci degree from the University of Tokyo in 1997. He is currently an associate professor at the Institute of Information Sciences and Electronics of the University of Tsukuba. Since 1997, he has also been a member of Precursory Research for Embryonic Science and Technology of the Japan Science and Technology Corporation. From 1989 to 1993, he was a research associate in the Department of Information Sciences, Faculty of Sciences, University of Tokyo. His current research interests include distributed systems, operating systems, programming languages, and persistent object management. He is associated with the Editorial Board of Transactions on IPSJ and Computer Software of the JSSST. He is a member of five learned societies, including the ACM and the IEEE Computer Society.

Yuichi Someya received his BE degree from the University of Tsukuba, Japan, in 1997. He is currently a graduate student in the Doctoral Program in Engineering at the University of Tsukuba. His research interests include distributed systems and programming languages.

Katsuya Matsubara received his BE and ME degrees from the University of Tsukuba, Japan, in 1994 and 1996, respectively. He is currently a research associate at the Institute of Information Sciences and Electronics at the University of Tsukuba. His research interests include distributed systems and operating systems.

Kunihiro Toumura received his BE and ME degrees from the University of Tsukuba, Japan, in 1995 and 1997, respectively. He is currently a graduate student in the Doctoral Program in Engineering at the University of Tsukuba. His research interests include operating systems, programming languages, and mobile agent systems.

Hirotake Abe is a bachelor student in the College of Information Sciences at the University of Tsukuba in Japan. His research interests include cooperative systems, autonomous agents, and network security.