Regular Paper

A Low-stretch Object Migration Scheme for Wide-area Environments

Ken Hironaka,† Kenjiro Taura† and Takashi Chikayama††

We propose a low-stretch scheme for locating mobile objects in wide-area computing environments. Locating mobile objects in distributed computing systems is a non-trivial problem and has been investigated for decades. The forwarding address algorithm, perhaps the most popular algorithm, requires the previous holder of the object to point to the successive holder, and to forward all requests along this pointer. However, this approach cannot provide any access stretch bounds for wide-area settings, and can incur unlimited communication overhead. This is unacceptable when a large number of objects simultaneously move or when numerous referencers attempt to access an object that has moved. We propose an active update method where nodes in the vicinity of the object's location are notified of its new location via localized update messages. Moreover, we will utilize the overlay topology information to minimize these messages. Referencers beyond the scope of the update will still be able to safely access the object. We will demonstrate that these updates maintain access stretches low even in wide-area settings.

1. Introduction

Wide-Area computing has become increasingly popular with the increase of network bandwidth and of the computational power of commodity computers. In such computing environments, distributed object-oriented programming is taking center stage. In particular, solutions like JavaRMI and CORBA have become popular choices for web-based services [1]. Java is becoming a favorite for grid-enabled high performance computing [2),3)]. The world's largest database is in fact, run by an object oriented database management system [4)]. The importance of object-oriented solutions is undeniable and in the future, it is important that such solutions can provide adaptivity to accommodate the dynamic nature of wide-area environments.

Object migration is one way in which distributed object-oriented systems provide adaptivity. Object migration allows increased performance by moving an object to a local destination and avoiding remote method invocations. It also enables adaptive load-balancing by distributing popular objects evenly and dynamically among nodes. It can also give more flexibility as nodes who want to leave or join a computation can do so by moving objects away or to itself.

Many object-oriented implementations [5)~8)] have adopted object migration as a tool, but have not answered the question of how it should be implemented in a wide-area setting. The question that needs to be answered in its scheme is, how can a mobile object be located? It is imperative that all existing remote and local object references stay valid even in the face of migration. Needless to say, providing location servers is not scalable in such settings.

One way in which an object can be accessed is by following the path of migration. However, an object can migrate across local networks. In a grid-enabled environment, wide-area communication costs hundreds and thousands of times more time compared to local-area communication. A simple senario is depicted in **Fig. 1**. In such cases, the access path becomes enormously large in comparison to the optimal path. Such an access overhead is unacceptable. Therefore, it is important that the *access stretch*, the ratio of the length of the access path to that of the optimal access path, is constantly low. The above approach does not satisfy this property.

The opposite approach is to update existing references actively so that all references will constantly reach the object via the optimal path. Though this will result in constantly low stretches, the message overhead is significant. An object movement of any kind will require a flooding of update messages. In the presence of a large number of mobile objects, this is clearly not scalable. Moreover, a localized change of location usually does not affect a very remote node in terms of locating the object; that node would still need to send an access message in the

† Graduate School of Information Science and Technology, The University of Tokyo
†† Graduate School of Frontier Sciences, The University of Tokyo

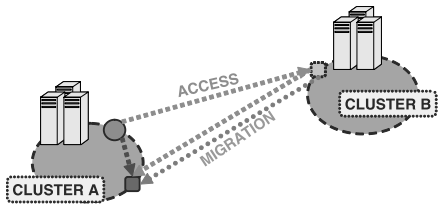


Fig. 1 Simply following the path of migration may force the accessor to traverse wide-area connections unnecessarily. Such traversals can result in access stretchs in the order of thousands.

same general direction. Thus, this method will encompass nodes that are largely unaffected by the modification.

Consequently, what is expected of an object migration scheme is a method that yields low access stretch from all nodes without flooding the network with update messages.

Hence in this paper, we propose a mobile object location scheme that utilizes localized dissemination of update messages without rely on object access monitoring. Participating nodes will construct an overlay network and routing among them will be performed using shortest-path routing. We utilize communication latency for the metric. The messages will be propagated by the most current holder of the object. The message propagation is localized by limiting the distance it can travel over the overlay network. Our scheme is able to give provably low stretch bounds on certain situations. Our scheme satisfies the following properties for any overlay topology.

(1) The object access stretch is constantly low on average
(2) The update messages are localized to the local area network of the object's location

The organization of this paper is as follows. In Section 2, we will introduce related work. In Section 3, we will explain our proposal. In Section 4, we will evaluate our scheme. Finally in Section 5, we will conclude.

2. Related Work

2.1 Forwarding Addresses

A very popular object tracking method has been analyzed by Fowler [9]. This has been adopted in many implementations [1),5),7),8),10)]. Even the more recent Java implementations like JavaParty [5] and Voyager [1] employ this scheme.

The method makes use of forwarding addresses to keep track of mobile objects. A *forwarding address* for an object is a data structure that contains the following

object id: a unique network-wide identifier for an object

timestamp: a number that represents the freshness of the forwarding address, where a larger number indicates a fresher piece of information.

location: location information of where the object was when the forwarding address was created

When an object migrates from a node, the contents of the node's object is replaced by a forwarding address. This means, that the timestamp will be incremented to the newest value, and that the location will point to the node to which the object migrated. An attempt to access the object at an old location will reveal a newer location. The access will be forwarded to the newer location, where a similar attempt to access is made. The above steps will continue until the access is finally forwarded to the correct location. In effect, this will create a *forwarding chain* from the accessing node to the hosting node. The object reference of the node from where the access originated will be updated after a successful object access. Henceforth an access will go directly to the correct location.

The forwarding address protocol is effective when migration is limited, or when mobility is confined to a small group of nodes. Fowler has given a proof that with the path compressing protocol, the average stretch of accessing a remote object is $O(\log N / \log(a/m))$ where N is the number of nodes the object visits in its lifetime, a is the number of times it is accessed, and m is the number of times it moves [9].

Many of the more modern distributed object oriented systems, including Java Party [5] and ProActive [11], implement migrant object tracking using this method.

However, Fowler's analysis of the forwarding address model assumes the following.

- all nodes are directly connected to each other
- the cost of all inter-node communication is equal

None of the above can be assumed for the modern wide-area environment. In such setting, following a forwarding chain may incur unnecessary traversals of wide-area connections. In such a situation, the access stretch can extend unconditionally.

2.2 Extension to the Forwaring Protocol

In Ref. 12), Moreau extends on Fowler's work. Moreau uses the same forwarding address protocol by adding an *inform* message. The message contains the object's location and the timestamp it had when it was at the location. When a node receives an inform message, its routing table is updated if the message is more up-to-date.

Moreau does not specify *when* inform messages should be sent, or *to where* the messages need to be sent. The work is left to the distributed system that implements the algorithm.

2.3 Referencer Updates: Thor

Thor is a distributed object-oriented database where objects are stored persistently at highly available servers called object repositories (ORs) [13]. In Thor, references to an object is a location dependent piece of information called *xref*s which contains

- the OR id
- the local address within the OR

When a reference makes an access, the request is forwarded to the corresponding OR, which executes the access using the local object ID. When an object moves to another OR, the corresponding *xref*s in the system must be updated. The source OR propagates the new *xref* to all ORs that have references to the migrated object. This selective propagation is made possible by the *inlist* on each OR. The distributed garbage collector maintains this list. An OR's inlist contains a list of ORs that hold references to objects on it. When an OR receives an update message, the update is saved in a *location table*, in which each entry contains

- old xref
- new xref

Whenever a distributed garbage collector scans references in an OR, the local location table will be used and old *xref*s are overwritten with newer *xref*s. However, object references themselves are not updated eagerly, and so accesses may be made to an old location. To accomodate this, the source OR of a migration also maintains a *serrogate* which effectively forward the access request to the location to where the object migrated.

Thor is unique in that it selectively propagates the update only to the nodes that are concerned. However, this method is dependent on a distributed garbage collector, and this by itself fails to address the general problem for object location. This also assumes that the garbage collector runs often enough to update old references in a timely manner. As a result, this method also resorts to the forwarding address protocol.

2.4 DSDV

Destination-Sequence Distance-Vector Routing protocol is a modification of the Bellman-Ford routing mechanism [14]. It realizes shortest-route routing for mobile networks where nodes move freely. In addition, the participating nodes also do not have to know all the nodes in advance.

Each node broadcasts its routing table periodically to neighboring nodes within its range. This eager exchange of routing information is classified as *proactive* routing. Each node maintains a routing table that consists of entries for each destination. Each entry contains

fwd a node to which messages to the destination are forwarded

nhops the number of hops required from the local node to the destination node

seq a sequence number that implies the freshness of the information

When a node receives a routing table in a message, the node compares it with that of its own. If the message contains new information, its routing table is overwritten and broadcasted again. A piece of information in a table is *new* if for any entry

- the sequence number is newer
- the sequence number is the same, but *nhops* is smaller than its own

This protocol can adapt to dynamic mobile networks because nodes advertise routing tables periodically. It's adaptability depends on how frequently each node broadcasts messages.

It is possible to disseminate routing information to each object in this manner. Each node will constantly be aware of object's freshest location and thus will constantly access the object using the optimal access path.

However, all nodes will receive an update for any migration of any object. The network will be flooded with update messages.

2.5 AODV

Ad-hoc On-demand Distance-Vector Routing protocol is a modification to DSDV [15]. While nodes exchanged messages eagerly in DSDV, they are done lazily in AODV. This method is classified as *reactive* routing. Unlike DSDV, shortest-path discovery is done on demand, when a message needs to be sent to a mobile

destination. At this time, a node broadcasts to neighboring nodes and requests a path to the destination. If neighboring nodes do not have a path to the destination, they too broadcast. This continues until a node, possibly the destination node itself, responds with the requested route information. The information is conveyed back to the initial inquirer in the reverse path of the broadcasts. Because unnecessary broadcast messages are not exchanged actively, the amount of routing messages can be reduced significantly. However, because routing is done on-demand, the protocol does not provide good response. It is unacceptable if an object access requires such a large latency even before the access is performed.

2.6 SHARP

SHARP is a *hybrid* protocol that combines proactive and reactive routing [16]. This protocol takes advantage of the fact that most internode communication happens among nodes close by. Each node constructs a *zone*, a group of near-by nodes with which it eagerly exchanges routing information. A node resorts to reactive routing with nodes outside its zone. SHARP goes furthur by adaptively adjusting zone size using network traffic information. Though this method strives to incooperate the benefits of both proactive and reactive routing, its heuristics on deciding zone size is still suspect.

3. Our Proposal

3.1 The Problem Setting

Our scheme functions in a dynamic network. We do not prohibit dynamic joining and leaving of nodes, nor do we disallow dynamic changes of connections. However in the presence of dynamic reconfiguration of nodes and connections, we cannot theoretically guarantee low access stretch. Thus for the purpose of our analysis, we assume a *static* overlay topology. In other words,

constant number of nodes: the nodes will stay constant throughout the computation, no nodes will leave or join the computation

static links: there will be no changing of edges between any pair of nodes during the computation

3.2 The Overview

In our proposal, we utilize a combination of update messages and inter-node routing to locate an object. At start up, all nodes construct a routing table to all destinations on the overlay network. In each table entry for a given destination, each node keeps the next hop, and the distance to that destination. The metric for distance is the communication latency over the overlay network. By doing so, shortest-path routing will be realized among all nodes. Constructing routing tables in effect creates *routing trees*, one for each node. A routing tree for a node is a tree in which a child node forwards a message to its parent node to reach the node. In effect, a routing tree for a given node has the node as the root. An example of such a tree is shown in **Fig. 2**.

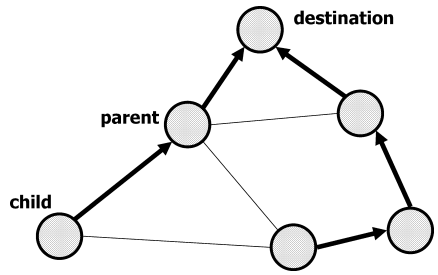


Fig. 2 All messages to the destination at the root are forwarded up the routing tree.

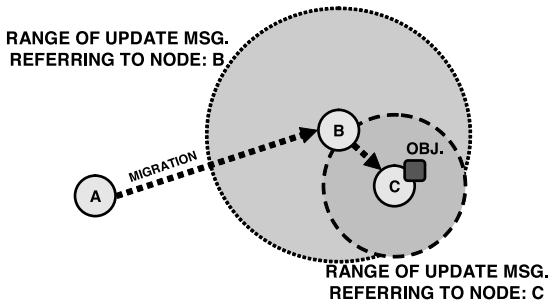


Fig. 3 The larger the migration distance, the larger update propagation range.

When an object migrates, the new object host broadcasts the updated object location message to all nodes in a given radius. The radius of dissemination is fixed to a value proportional to the distance the object traveled from the old host to the new host. We call this proportional value, the *propagation ratio*. The manner in which the update is propagated is illustrated in **Fig. 3**. Meanwhile, the old host node corrects its local reference to point to the new host node.

When a given node accesses a remote object, it does so by emitting an object access message. An object access message contains the node information of where the object is thought to be. When the message is emitted, the re-

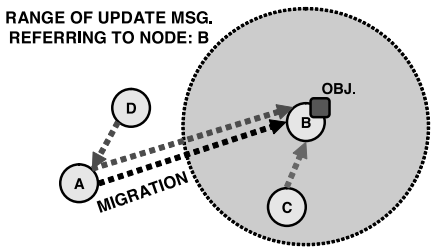


Fig. 4 While node D visits node A first, its overall stretch is not affected mostly.

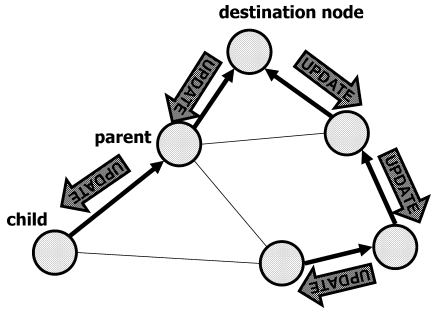


Fig. 5 The update message can be disseminated by sending the message from the parent to its children.

quest will be forwarded by intermediate nodes towards the node hosting the object. To determine where to forward a request, a forwarding node will use the newer information between the location the node has, and the location that is included in the access message. A node will overwrite the old information and will forward the request towards the location in the newer information. Quite possibly, both pieces of information will be out-dated. Thus, it is possible that the object access will reach an old object host. Since the old object host has information as to where the object migrated, the old host is guaranteed to have newer information than the object access message. Thus in this case, the message will be forwarded to the newer location. Hence, nodes that do not receive the latest object location information are still able to access the object.

The dissemination of update messages is designed to prevent senarios like that described in Fig. 1. The effect of our method can be illustrated in **Fig. 4**. The object in question migrates from Node A to Node B. In the senario, node C is close enough to node B that node C receives the true location of the object. Thus node C will access node B directly. On the other hand, node D is unaware and its access message mistakenly goes to node A when it attempts to access. However, node D is far enough from node B anyway and the overall access stretch, the ratio of the actual access path and the optimal access path, does not grow too large.

3.2.1 Optimizated Progapation Method

Disseminating a message to all nodes within a given distance can be done without having each node broadcast the message on all of its links. The same results can be achieved by using a tree. For this purpose, we will utilize the routing tree rooted at the new object host. This is shown in **Fig. 5**. In order to send a message to all nodes within a given distance from the root, the root node can propagate the message down the routing tree until the desired distance is reached.

3.3 The Algorithm

3.3.1 Node ID

Each Node that is participating in the system is given a unique ID. We will call this the node id.

3.3.2 Inter-node Routing

At start-up, the participating nodes will construct a shortest-path routing table to each node. This can be done by each node initially flooding routing information to each other. When the routing tables are complete, each node will have record the distance to the destination node on the overlay, and the *next hop* to that node. In addition, each node will record the adjacent nodes that are dependent on itself to route messages to a given node. In other words, all parent nodes record their children nodes for all routing trees.

3.3.3 The Object Reference

Each mobile object is labeled with a tag that uniquely identifies the object, the *UID*. A reference to an object is in effect, this identifier. This identifier is assigned at birth and contains the node id of where the object was first created.

3.3.4 The Sequence Number

Each node maintains a table containing the object location information that it is aware of. Each entry in this table contains the following

uid: the object's unique identifier

location: the location information of the object

seq: the sequence number that tells the freshness of the information

In particular, the sequence number is a monotonically increasing number in which a larger value signifies a newer piece of information.

3.3.5 Object Migration

When an object migrates, it is packaged in a *MOVE* message and is forwarded to the new object host. The *MOVE* message contains the following.

object: the object itself
seq: the object's sequence number
dist: the distance the object has traveled

The *dist* element is initially set to 0. When a node receives a *MOVE* message, the *dist* element is increased by the weight of the link from which the message was received. If the node is not the destination node, the message is forwarded to the next hop to the destination. When the *MOVE* message reaches the destination node, the message in processed and the object is unpackaged. After the object has been unpackaged, the new object host returns a *MOVE ACK* message back to the source node.

3.3.6 Update Message Propagation

3.3.6.1 Emitting the Update Message

Whenever an object migration completes, the following steps are followed.

(1) the source and destination nodes update the object's entry in its object routing tables to reflect the migration. When they do so, they increment the sequence number for the object
(2) the destination node propagates the *UPDATE* message

The *UPDATE* message includes the following.

uid: the object's *UID*
seq: the object's new sequence number
old_host: the old host node's id
new_host: the new host node's id
TTL: a Time-To-Live (TTL)

3.3.6.2 The Propagation Ratio

In particular, the TTL is initially set to

$$dist \cdot k \quad (1)$$

where *dist* is the distance the object traveled and k is the propagation ratio. If $k = 1$, it would mean that the *UPDATE* message is propagated in the radius of the migration distance. By choosing a smaller value of k, we can limit the amount of message and sacrifice the access stretch performance.

3.3.6.3 Propagating the Message

When a node first emits or forwards an *UPDATE* message, it constructs a set of adjacent nodes to which the message will be sent. Let $dep(pid)$ represent the set of adjacent nodes that are dependent on the node to send a message to *pid*, and $link\_weight(pid)$ represent the weight of the edge to *pid*. The set of nodes to which the message will be sent are

$$\{x : x \in dep(new\_host) \;\&\&\; link\_weight(x) < TTL\}$$

3.3.6.4 Forwarding the Update Message

When a node receives an *UPDATE* message, it compares the update message's sequence number to that of the entry in its object routing table. If the message's sequence number is greater, it signifies a new piece of information and its object routing table is updated. If the table is updated, the *TTL* is decreased by the weight of the link from which the message was received and the node propagates the *UPDATE* message.

3.3.7 Object Access

3.3.7.1 Constructing an Object Access Message

Accessing an object requires a reference to that object. When a node tries to access a non-local object, it creates an *ACCESS_REQ* message. The message includes the following.

uid: the *UID* of the object
host: the location where the object is *thought* to be.
seq: the sequence number of the location information

The *UID* can simply be derived from the reference. The location information and the sequence number can be drawn from the node's object routing table. There may be a case where the object information is not avaiable in the table. In such cases, the object's birth location is derived from the *UID* (cf., Section 3.3.3) and the sequence number is set to 0.

The request message is sent to the *next hop* to the location specified in the message.

3.3.7.2 Forwarding an Object Access Message

When a node receives an *ACCESS_REQ* message, it processes the access if the object resides on the node. If this is not the case, the object will be forwarded to the next node. In order to estimate where the object can be, the node will use it's object routing table *and* the information in the request message. Of the two pieces of information, the information with the *larger* sequence number is respected. The piece of information with the *smaller* sequence number is overwritten with the other. After this check is complete, the request message is forwarded to the *next hop* towards the location.

3.3.7.3 Path Compression

Like the forwarding address scheme, we utilize a path compression protocol. When an object access is processed by the target object, the object returns the result of the access to the accessor. The location of the object at that time, along with its current sequence number is piggy-backed on this message. Henceforth, the accessor is able to access the object at this new location.

3.4 Access Stretch

With our algorithm, we give strict bounds on the access stretch to mobile objects. For any propagation ratio $k > 0$, an access from any node to any mobile object can be bounded by a stretch of $1 + \frac{2}{k}$. A formal analysis is given in the Appendix. This bound holds only for when an object migrates at most once between accesses from any node. If an object migrates m times in between accesses from a node, the access stretch from that node can grow with the power of m. Even in such cases however, the worse case stretch is unlikely and the access stretch stays low on average. We shall show this by evaluation.

4. Evaluation

In this section, we will show that,

- our scheme gives low access stretch to mobile objects in wide-area environments, even with small propagation ratios
- our scheme allows update dissemination to be limited to localized areas in the network while still maintaining low access stretch

We use three clusters to simulate a wide-area environment for our experiment:

HONGO (66 nodes): CPU: Intel (R) Pentium (R) M processor 1.86 GHz, RAM: 1 GB

CHIBA (64 nodes): CPU: Intel (R) Pentium (R) M processor 1.86 GHz, RAM: 1 GB

KASHIWA (66 nodes): CPU: Intel (R) Xeon (TM) CPU 2.40 GHz, RAM: 2 GB

We show the round trip time between and within each cluster in **Fig. 6**.

For all of the experiments we present below, we constructed a random graph where each node connects to 10% of all participating nodes.

In Section 4.1, we compare and evaluate the access stretch of our scheme against the forwarding address scheme in a multi-cluster environment. In Section 4.2, we discuss the message overhead and how our scheme limits the propagation area.

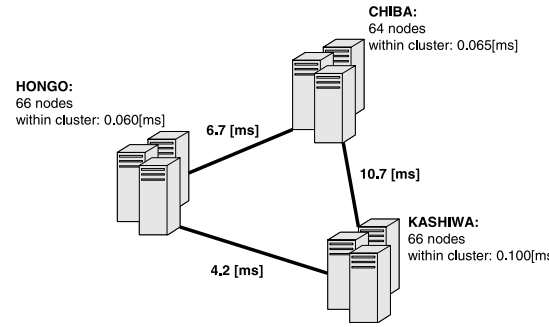


Fig. 6 The inter and intra cluster RTT for the three clusters.

4.1 Access Stretch

In the first set of experiments, we analyze the object access stretch from a single accessor when an object migrates within the three clusters. The evaluation is done with different access/migration frequencies.

4.1.1 One Migration in between Accesses

We evaluate a case where 1,000 objects migrate as a group to different random nodes. A fixed node accesses the objects sequentially. Where one access iteration is an access to each object, the objects migrate after every 5 iterations. The accesses was resumed once the migration has been complete. We compute the *average access stretch* of each iteration by dividing the total lapse time of an iteration by the total lapse time of an iteration had the accesses taken the optimal path. We let the optimal access time be the shortest completion time among every 5 iterations. We compare the average access stretch for the forwarding address scheme and our scheme. For our scheme, we varied the propagation ratio. We attained the results shown in **Fig. 7** (a)–(d).

In both Figs. 7 (a) and (b), we see that the optimal stretch is attained for the most part. A non-optimal access stretch is attained for the first of every 5 iterations. This is because a non-optimal stretch happens when the accessor goes to access the object at its old location. For the forwarding address scheme, many of the non-optimal stretches are scattered high above the optimal value of 1. We believe that the very large stretches came from senarios described in Fig. 1 because they arose when the object moved nearby the accessing node. Thus, we hypothesize that the effect will be more visible in environments where the distribution of

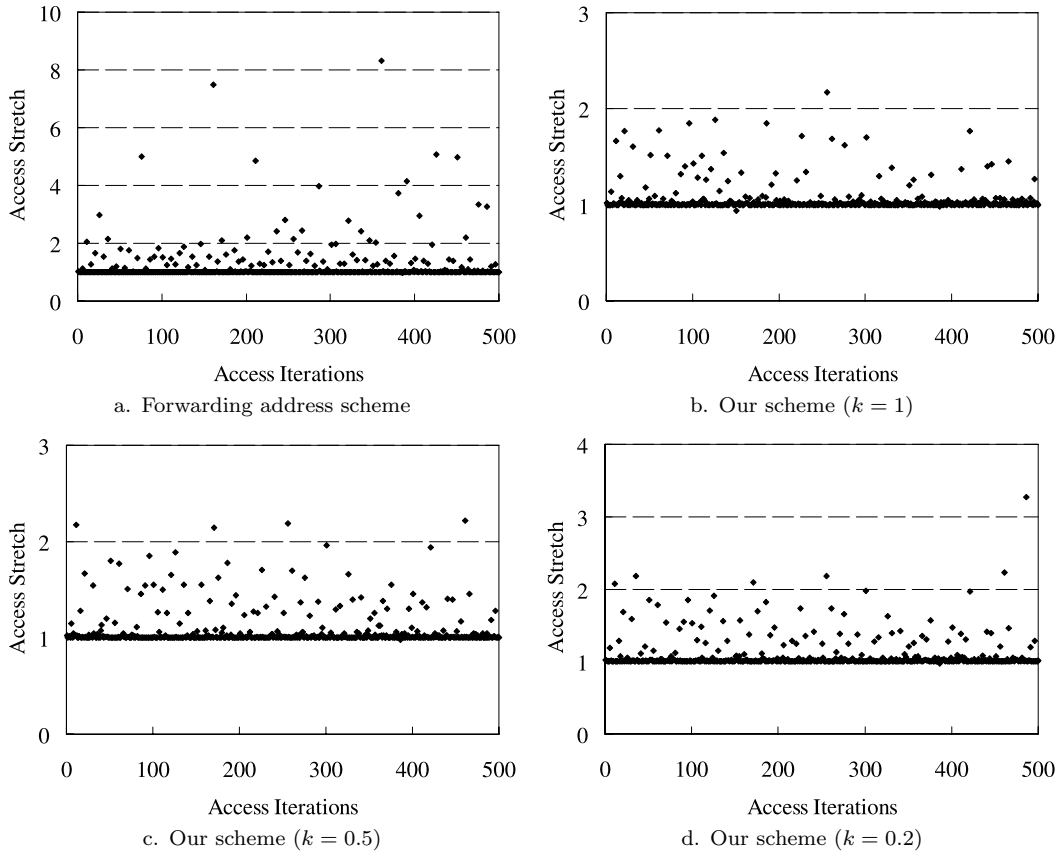


Fig. 7 Average access stretch when 1,000 objects migrate after every 5 iterations with 195 nodes in 3 clusters.

the inter-node latency is much greater. On the other hand for our scheme, we see that the non-optimal stretches are bounded by a stretch of 3. This is a confirmation of our assertion in Section 3.4.

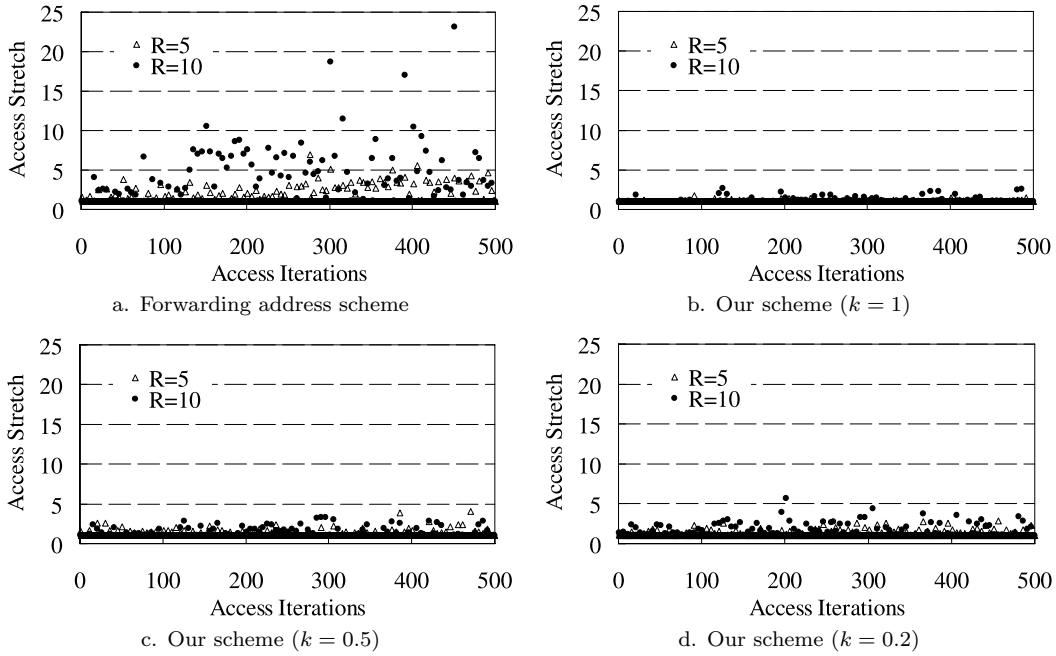
In Figs. 7 (c) and (d), we show how our scheme performs with smaller propagation ratios. Though their non-optimal access stretches are larger at some points, they are roughly equivalent to our results when $k = 1$. In these case, the maximum stretch bounds guaranteed in our analysis become larger, but in reality, we see that their maximum stretch is roughly uneffected by a smaller k.

4.1.2 Multiple Migrations in between Accesses

Generally, an object can move independently of how it is being accessed. Below we generalize the previous experiment to reflect this case. Now, the object group is allowed to migrate to random nodes R times after every 5 iterations. Likewise, we compare the average access stretch between the forwarding address scheme and our scheme. We show the results in **Fig. 8** (a)–(d).

We can see clearly in Fig. 8 (a) the forwarding address scheme's vulnerability when the target object moves multiple time before it is accessed. This can be explained by the fact that, when the object continues to migrate, the forwarding chain becomes longer. Moreover in a wide-area environment, a single forwarding may entail a number of hops on the overlay network, and a single hop may be a wide-area connection that is very costly.

On the other hand, our scheme with $k = 1$ is hardly affected on average by the extra migration. Figures 8 (b), (c), and (d) also show that decreasing k does not dramatically change the access stretch performance either. More importantly, though we were not able to provide any bound on the access stretch in this general case, the Figs. 8 (b), (c), and (d) show that the access stretch in reality stays low even if an object moves multiple times in between accesses.



a. Forwarding address scheme
b. Our scheme ($k = 1$)
c. Our scheme ($k = 0.5$)
d. Our scheme ($k = 0.2$)

Fig. 8 Average access stretch when 1,000 objects migrate R times after every 5 accesses with 195 nodes in 3 clusters.

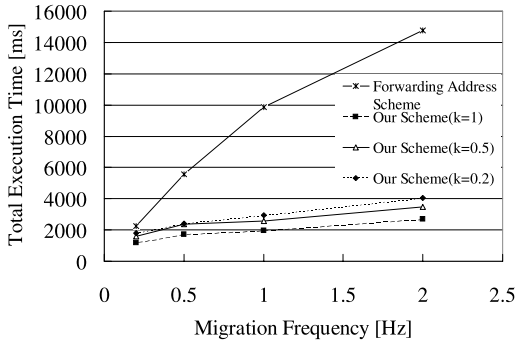


Fig. 9 Completion time of 100 iterations with varying migration frequencies with 193 nodes in 3 clusters.

4.1.3 Access with Aggressive Object Migration

For the final experiment in this round of experiments, we compare our scheme's performance when objects migrate aggressively. We perform the evaluation by moving 1,000 objects as a group every P seconds while a fixed node incessantly accesses them. We measured the time to complete 100 access iterations. The result is shown in **Fig. 9**.

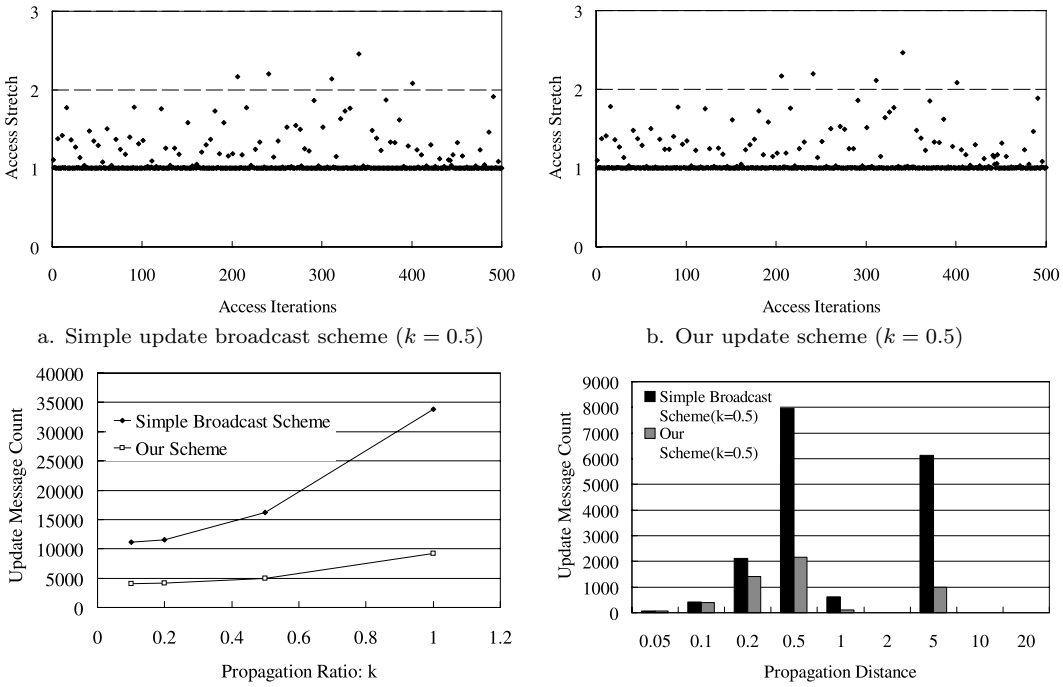
It is clear from Fig. 9 that the completion time deteriorates quickly for the forwarding address scheme. This can again be attributed to the fact that as the objects' consecutive migration increases, the fowarding chain becomes unbearably long. Our scheme performs well for all values of k with which we tested. Moreover, the execution time's growth rate with respect to the migration frequency is considerably smaller compared that of the forwarding address approach. Thus, we can say that our scheme performs well even in settings where a target object continually migrates from node to node.

4.2 Message Overhead

So far, we have demonstrated that the access stretch stays low for our scheme in wide-area settings. We now evaluated its update message overhead. First, we see how update messages in our scheme can be limited to localized areas in the network while still providing low stretch. Next, we compare our scheme to the simple localized update broadcast scheme.

4.2.1 Localized Updated Dissemination

First, we demonstrate that by changing the propagation ratio, we can effectively contain update messages within localized sections of a wide-area network. We performed an experiment where an object was migrated randomly 100 times within the overlay network. While varying the propagation ratio, we measured the total distance each update message traversed in the network. We then constructed a histogram



a. Simple update broadcast scheme ($k = 0.5$)

b. Our update scheme ($k = 0.5$)

c. Message ratio between the two schemes for 100 migrations

d. Histogram of the distance the update messages traversed for the two update schemes

Fig. 11 Comparison of the three update schemes with 195 nodes in 3 clusters.

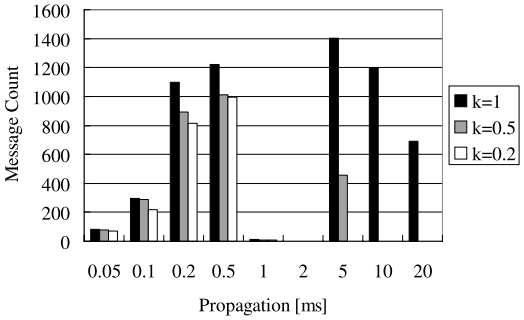


Fig. 10 A histogram of the distance the update messages traversed for 100 migrations with 193 nodes in 3 clusters.

based on the distance they covered. The result is shown in **Fig. 10**.

In the histogram in Fig. 10, we see that there are hardly any messages that traversed more than 0.5 [ms] but less than 2.0 [ms]. This arrises from the geographical position of the clusters as expressed in Fig. 6. From this, we can safely say that the messages travelling less than 0.5 [ms] are messages that stayed within a cluster, while those above 2 [ms] are messages that crossed cluster boundaries. It can be observed that messages disseminated within the cluster of the migrated object do not differ among different values of k. Particularly long stretches arise when nodes within the cluster in which the object reside are not notified of the migration. For different values of k, the extent to which the nodes of the object's local cluster are informed does not change. This explains why the stretch performance did not significantly decay in the previous experiments. Therefore, long stretches can still be avoided with smaller values of k. On the other hand, where the extent of dissemination *does* change according to k are the notifications that cross clusters. However, we have already established that nodes far away from the object are mostly unaffected in terms of stretch. Thus, inter-cluster notifications are for the most part, unnecessary. Hence, we can say that by setting a small value of k, update messages can be propagated economically while still maintaining low access stretch.

4.2.2 Effects of the Update Message Optimization

Next, we evaluate the effect of our optimized broadcast scheme. To gain a comparison, we performed the experiment in Section 4.1.1 using two different methods. For the first, we chose a simple localized broadcast method where update messages are simply flooded until they

reach a certain distance. For the second, we only applied our optimization presented in Section 3.2.1. During the experiment, we also counted the total number of update messages that was sent. The results are presented in **Fig. 11** (a)–(d).

By comparing the Figs. 11 (a) and (b), we can discern that their average stretch performance are identical. Thus, we confirmed that the access stretch is not affected by this scheme. Figure 11 (c) shows the benefits of our optimization schemes. Our optimized broadcast scheme is clearly better than the simple broadcast scheme as the messages sent can be reduced to a third without degrading stretch. A much more noticable difference is visible when comparing the distance the update messages travel. This is shown in Fig. 11 (d). In the histogram, there is a noticable difference between our scheme and the simple broadcast scheme beyond the 2 [ms] mark. This implies that our optimizations significantly reduces unnecessary updates that cross cluster boundaries. Thus, we can say that not only does our scheme reduces message count, it also reduces the area in which they are propagated.

5. Conclusion and Future Work

In this paper, we presented a new object migration scheme for wide-area environments. Our solution for the problem setting was to propagate update messages when objects migrate and to provide low access stretch.

We confirmed by experiment our analysis on access stretch bounds for when an object only migrates once in between accesses. We were not able to provide any theoretical strech bounds for the general case. However by evaluation, even if the object migrates multiple times in betweeen accesses, we showed that the access stretch stays very low on average.

We compared our results to that of the conventional forwarding address approach and showed our scheme's significance. While a forwarding address method is threathened by a much longer stretch by every migration, ours constantly gives low stretch. We also showed that our scheme makes a large difference when object migration frequency is high.

Finally, we demonstrated that by utilizing our scheme, we can confine the range of update propagation to small sections of the whole network, like a cluster. We also showed that this is enough to maintain low access stretch. Furthermore, we demonstrated that our optimized broadcast significantly reduce message overhead in terms of the number of messages disseminated and the area in which it is disseminated. Overall, we showed that it is possible to utilize location update messages to maintain constantly low access stretch to mobile objects.

For future work on this topic, we will like to improve our updating scheme. In particular, instead of propagating updates for a distance proportional to the migration distance, it may be better to do so for a distance proportional to the *log* migration distance. Realistically speaking, this may be able to curb the update propagation area when an object migrates a great distance, without affecting the overall access stretch. In addition, we will like to evaluate the improved scheme in a much wider scale. So far, we have evaluated our scheme in a metropolian environment. We believe that much more interesting results can be obtained in a more wide-area environment.

References

1) Boger, M.: *Java in Distributed Systems*, Wiley and Sons (2001).
2) van Nieuwpoort, R.V., Maassen, J., Hofman, R., Kielmann, T. and Bal, H.E.: Ibis: An efficient Java-based grid programming environment, *JGI '02: Proc. 2002 joint ACM-ISCOPE conference on Java Grande*, New York, NY, USA, pp.18–27, ACM Press (2002).
3) Huet, F., Caromel, D. and Bal, H.E.: A High Performance Java Middleware with a Real Application, *Proc. Supercomputing conference*, Pittsburgh, Pensylvania, USA (2004).
4) Becla, J. and Wang, D.L.: Lessons Learned from Managing a Petabyte, Technical report, SLAC.
5) Philippsen, M. and Zenger, M.: JavaParty — Transparent Remote Objects in Java, *Concurrency: Practice and Experience*, Vol.9, No.11, pp.1225–1242 (1997).
6) Roy, P.V., Haridi, S., Brand, P., Smolka, G., Mehl, M. and Scheidhauer, R.: Mobile Objects in Distributed Oz, *ACM Transactions on Programming Languages and Systems*, Vol.19, No.5, pp.804–851 (1997).
7) Jul, E., Levy, H., Hutchinson, N. and Black, A.: Fine-grained mobility in the Emerald system, *ACM Trans. Comput. Syst.*, Vol.6, No.1, pp.109–133 (1988).
8) Chase, J., Amador, F., Lazowska, E., Levy, H. and Littlefield, R.: The Amber system: Parallel programming on a network of multiprocessors, *SOSP '89: Proc. twelfth ACM symposium on*

Operating systems principles, New York, NY, USA, pp.147–158, ACM Press (1989).
9) Fowler, R.J.: The complexity of using forwarding addresses for decentralized object finding, *PODC '86: Proc. fifth annual ACM symposium on Principles of distributed computing*, New York, NY, USA, pp.108–120, ACM Press (1986).
10) Steensgaard, B. and Jul, E.: Object and native code thread mobility among heterogeneous computers (includes sources), *SOSP '95: Proc. fifteenth ACM symposium on Operating systems principles*, New York, NY, USA, pp.68–77, ACM Press (1995).
11) Baude, F., Caromel, D., Huet, F. and Vayssiere, J.: Communicating Mobile Active Objects in Java, *Proc. HPCN Europe 2000*, LNCS, Vol.1823, pp.633–643, Springer (2000).
12) Moreau, L.: Distributed directory service and message routing for mobile agents, *Science of Computer Programming*, Vol.39, No.2-3, pp.249–272 (2001).
13) Day, M., Liskov, B., Maheshwari, U. and Myers, A.C.: References to Remote Mobile Objects in Thor, *ACM Letters on Programming Languages and Systems*, Vol.2, No.1-4, pp.115–126 (1993).
14) Perkins, C. and Bhagwat, P.: Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers, *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pp.234–244 (1994).
15) Perkins, C.: Ad-hoc On-demand Distance Vector Routing (1997).
16) Ramasubramanian, V., Haas, Z.J. and Sirer, E.G.: SHARP: A hybrid adaptive routing protocol for mobile ad hoc networks, *MobiHoc '03: Proc. 4th ACM international symposium on Mobile ad hoc networking & computing*, New York, NY, USA, pp.303–314, ACM Press (2003).

Appendix

A.1 Analysis on the Access Stretch

Below, we provide some theoretical backing to our scheme given a few assumptions.

A.1.1 Formalization of Our Problem Setting

By assuming a static overlay topology, we assume a connected graph with bidirectional weighted edges, $G(V, E, \omega)$ where V is a collection of nodes, $E \subseteq V \times V$ is a set of edges, and ω is a weight function $\omega : E \to [0, \infty)$. We assume the following properties.

- there exists a node pair (u, v) such that $(u, v) \in E$ if and only if there exists a TCP connection between node u and node v
- for any $(u, v) \in E$, $\omega(u, v) \in \mathbb{R}$ and $\omega(u, v) > 0$. It may be any metric that satisfy these conditions. For example, the round trip time for the TCP connection between node u and node v

For the analysis below, we also apply a constraint concerning the object's mobility.

- $\forall u \in V$, any given object migrates at most once between u's access to the object.

We shall call this the *object mobility contraint*.

A.1.2 Distance

For any pair of nodes $(u, v) \in V \times V$ in our graph $G(V, E, w)$, we define their distance to be the sum of the weights of all the edges that are traversed to go from u to v. First, we define the order of traversal from u to v by an *access series* $R(u, v)$. We define an access series from u to v to be

$$\begin{aligned} R(u,v) \equiv \{r_i | (r_0 = u) \\ \cap (r_n = v) \cap (r_i, r_{i+1}) \in E \ (i = 0, ..., n) \\ \cap r_i = r_j \Leftrightarrow i = j\} \end{aligned} \quad (2)$$

where n is the number of hops in the traversal. The cost $\delta(R(u, v))$ of the traversal in the order of a given access series $R(u, v)$ is

$$\delta(R(u,v)) = \sum_{i=0}^{n-1} \omega(r_i, r_{i+1}) \quad (3)$$

A.1.3 Shortest Path Routing

Since we assume shortest path routing, for any pair of nodes $(u, v) \in V \times V$, their distance $d(u, v)$ satisfies the following property.

$$\forall R(u,v)\{d(u,v) \leq \delta(R(u,v))\} \quad (4)$$

We will denote the access series that yield $\delta(R(u, v)) = d(u, v)$ by $R(u, v)^*$

Lemma 1:

if shortest path routing is realized for all pair of nodes $(u, v) \in V \times V$, for any 3 nodes $u, v, w \in V$, the following inequality is satisfied

$$d(u,w) \leq d(u,v) + d(v,w) \quad (5)$$

Proof. If $d(u, w) > d(u, v) + d(v, w)$, we can concatenate two access series $R(u, v)^*$ and $R(v, w)^*$ that yield $d(u, v)$ and $d(v, w)$ to create a new access series $R(u, w)'$. $R(u, w)'$ will have a cost of $d(u, v) + d(v, w)$. This is a violation of Eq. 4 and thus, this is not possible. □

A.1.4 Stretch

Lemma 2:

Given the object mobility constraint, $\forall u \in V$, u's access to the object will forwarded at most once.

Proof. An object access is forwarded by a node

if and only if the object does not exist on the node. The access is guaranteed to be forwarded to a newer host of the object. Given the object mobility constraint, if u's access is forwarded, it is forwarded to a node where the object currently resides. In this case, the access is no longer forwarded, and so the access to the object is forwarded at most once. □

Property:

Given the object mobility constraint, $\forall u, v, w \in V$ when the object moves from node u to node v, and node v propagates the update message in the distance radius of $kd(u, v)$ where $k \in R$, node w's access stretch to the object can be bounded by $1 + \frac{2}{k}$.

Proof. If $d(v, w) \leq kd(u, v)$, node w would have been notified of the new object's location, and node w accesses node v directly. Thus, node w's access stretch $stretch(w, v) = \frac{d(w,v)}{d(w,v)} = 1$. If $d(v, w) > kd(u, v)$, in the worst case, node w will access node u. By the object mobility constraint and Lemma 2, the access will be forwarded to node v, the current host of the object. In such a case, node w's access stretch $stretch(w, v) = \frac{d(w,u)+d(u,v)}{d(w,v)}$. By applying lemma 1, we have

$$\begin{aligned} stretch(w, v) &= \frac{d(w, u) + d(u, v)}{d(w, v)} \\ &\leq \frac{d(w, v) + 2 \cdot d(u, v)}{d(w, v)} \\ &= 1 + 2\frac{d(u, v)}{d(w, v)} \qquad (6) \end{aligned}$$

Because $d(v, w) > kd(u, v)$, $stretch(w, v) < 1 + \frac{2}{k}$. □

(Received January 22, 2007)
(Accepted April 24, 2007)

Ken Hironaka is currently enrolled in the Graduate School of Information Science and Technology at the University of Tokyo. He was born in 1983, and received his B.S. degree from the University of Tokyo in March, 2007. His current research interests include grid-enabled computing and distributed object-oriented programming languages. His hobbies include Ultimate frisbee, travelling overseas, and other things that have nothing to do with programming.

Kenjiro Taura is associate professor at Department of Information and Communication Engineering, the University of Tokyo. He was born in 1969, and received his B.S., M.S., and D.Sc. degrees from the University of Tokyo in 1992, 1994, and 1997. His major research interests include parallel/distributed computing and programming languages. He is a member of ACM and IEEE.

Takashi Chikayama is a professor at the Department of Frontier Informatics, School of Frontier Sciences, the University of Tokyo. He received degrees of B.Eng. in 1977 and Dr.Eng. in 1982, both from the University of Tokyo. From 1982, he had been at the Institute for New Generation Computing Technologies engaged in the Fifth Generation Computing Project until he moved to the University of Tokyo in 1995. His research interests include parallel and distributed processing systems, programming languages, software development systems, and machine learning technologies.