Migrating Autonomous Objects in a WAN Environment*

Natalija Krivokapić *Markus Islinger* *Alfons Kemper*

Universität Passau
Lehrstuhl für Informatik
94030 Passau, Germany
Tel. (++49-851) 509-3060
FAX (++49-851) 509-3062
⟨lastname⟩@*db.fmi.uni-passau.de*

Abstract

Along with the fast progress the interconnection of computer systems makes the need for enterprise-wide distributed solutions grows. These systems have to support hundreds or even thousands of sites located all over the world. The distances between the sites will impose high communication costs on distributed activities thus significantly increasing their response times. This problem can be alleviated through migrating objects to the sites where they are needed. However, migration in such systems cannot be managed globally. Therefore, we present a migration protocol for (autonomous) objects which enables them autonomously to decide whether or not and which site to migrate to, thus making the system self-tuning. We also present two migration strategies objects can use to derive their migration decisions. The first one finds the optimal placement for an object while the second strategy in some cases returns a sub-optimal location but induces lower computation costs. The two strategies are evaluated through benchmarks in a distributed system of autonomous objects. The experiments show a significant decrease in communication costs when migration is employed.

1 Introduction

The world-wide interconnection of computer systems leads to a growing need for large-scale enterprise-wide distributed solutions. These systems have to support hundreds or even thousands of sites running on many different platforms. The coordination of objects involved in a common task in these systems cannot be done in a centralized manner, but can be achieved only through decentralized cooperation between the objects themselves or between the sites they are located at. Cooperation concepts demand a high

*This work was partially supported by the German Research Council DFG under contract Ke 401/7-1 and by the German Israel Foundation (GIF)

degree of object or site autonomy. Objects augmented by an autonomous behavior, i.e., autonomous objects, are therefore predestined for the implementation of these systems, inherently providing the needed autonomy and being easily extendible by cooperation concepts [LM96].

The response time of (widely) distributed activities in wide-area network environments is dominated by communication costs. In these environments the location of data has a much greater impact than this is the case in a system spanning a small area, e.g., a LAN. A "wrong" placement of an object can induce much higher communication costs and thus increase the response time of transactions. Furthermore, "wrong" placements of (closely interacting) objects also burdens the network, on a large scale leading to bandwidth problems. However, statically placing objects once and forever cannot solve the problem since in a wide-area configuration shifts in the system's load are to be expected. A reason for such a load shift could for example be a shift of activity between different time zones in the course of a day within an enterprise-wide distributed system. Another example would be the different phases an object goes through during its life cycle, e.g., a product being designed in one place and being built at the other end of the world. Therefore, our objective is to reduce communication costs through migration.

In truly distributed systems migration cannot be managed globally. Mariposa [SAL+96, SDK+94]—which is a recent prototype of a WAN-distributed system—addresses the demand for moving data in a decentralized manner by employing an economic paradigm. In their approach data fragments move on behalf of the sites. Sites can autonomously buy and sell data fragments through bidding processes, trying to maximize their profit.

In a system of autonomous objects the natural unit of migration is an object. Our goal is a self-tuning system which can automatically adjust to changes occurring in the system, e.g., a load shift. To achieve this autonomous objects have to monitor where they are accessed from and decide—based on the observed access patterns—whether to migrate or not.

In this paper we present a migration concept for autonomous objects in a WAN environment, e.g., within an enterprise-wide distributed system. The migration decision as well as the migration process itself are performed autonomously by the objects. We describe the implementation of the migration component and two migration policies objects can employ when considering migration.

Both strategies assume a network of LANs—that could be placed all over the world—for which the average distances between the LANs are known, or can be obtained. The first strategy, called Best-LAN strategy, considers all possible placements for an object and migrates to the best one—in case the savings would be high enough. This strategy finds the optimal location for an object but induces high computation costs.

For the second strategy, called Good-LAN strategy, clusters of LANs located close to each other are identified. These clusters can again be composed to clusters of clusters, so a hierarchy of clusters is modeled—basically reflecting the hierarchy in today's networks. When an object considers migration it starts at the highest level and considers only migrating to sub-clusters of the best cluster of the current level. This policy finds a good placement for an object but not necessarily the best one. However, its computation costs are lower than those of the first strategy.

The presented policies have been evaluated through performance measurements which were done in AutO—a distributed system of autonomously operating objects designed for a WAN environment. The system is based on our underlying model of autonomous objects.

The rest of the paper is organized as follows. Section 2 describes the underlying model of computation. The migration of objects, its implementation and the two policies we propose are presented in Section 3. The clustering algorithm used to build the hierarchical structure for the Good-LAN strategy is also presented in Section 3. A performance evaluation based on measurements performed in AutO is given in Section 4. An overview of related work is given in Section 5. Finally, Section 6 concludes this paper.

2 Model of Computation

Passive objects encapsulate a state, i.e., the value of its data, and provide a set of operations, called methods, that can be invoked to query and manipulate the data. Autonomous objects are augmented by an autonomous behavior, i.e., they are not controlled in a master/slave relation by invocations from outside, but have the ability to perceive stimuli and autonomously react to them. Semantically, autonomous objects are amended by an *activity* (thread), that enables them to react to events that have occurred, and a *behavior* describing these reactions.

In our model, introduced in [KLMW94], the autonomy of objects relies on asynchronous message passing. An object can receive messages—which trigger events in the object—and autonomously react to them. Apart from events triggered by messages (sent by other cooperating entities, e.g., by a transaction), an object can react to time events and changes of its state. It also has the ability to send messages to other objects, thus triggering events within these objects.

In most cases autonomous objects will be large objects having a set of methods operating on its data. When multiple objects are involved in a common task they have to cooperate. Different tasks may require different kinds of cooperation, even in case it is the same set of objects involved in these tasks. The coordination of the cooperation between objects is done by transactions. As said above objects can directly communicate with each other, but the usual way of coordination will be through transactions, i.e., a transaction will send messages to objects and based on the results obtained from the objects decide what to do next, e.g., which objects to send messages to.

In our model, transactions are realized as first class objects, i.e., as autonomous objects. A transaction consists of a sequence $O_1.op_{i_1}(\ldots);\ O_2.op_{i_2}(\ldots);\ O_3.op_{i_3}(\ldots);\ \ldots$ of operation invocations op_{i_j} on one or several objects O_j—the "$(\ldots)$" denotes additional parameters of the invocation. The autonomous behavior of objects makes it possible to move the concurrency control to the object's level, i.e., the objects themselves are responsible for the lock management of their data. Transactions follow a strict two-phase locking protocol: In the first phase a transaction requests operation executions on (possibly) different objects. In the second phase transactions initiate the commit processing, realized through message passing, after which the locks are released. The two-phase commit protocol is employed to ensure transaction atomicity.

Each autonomous object, i.e., a data object as well as a transaction, has a system-wide

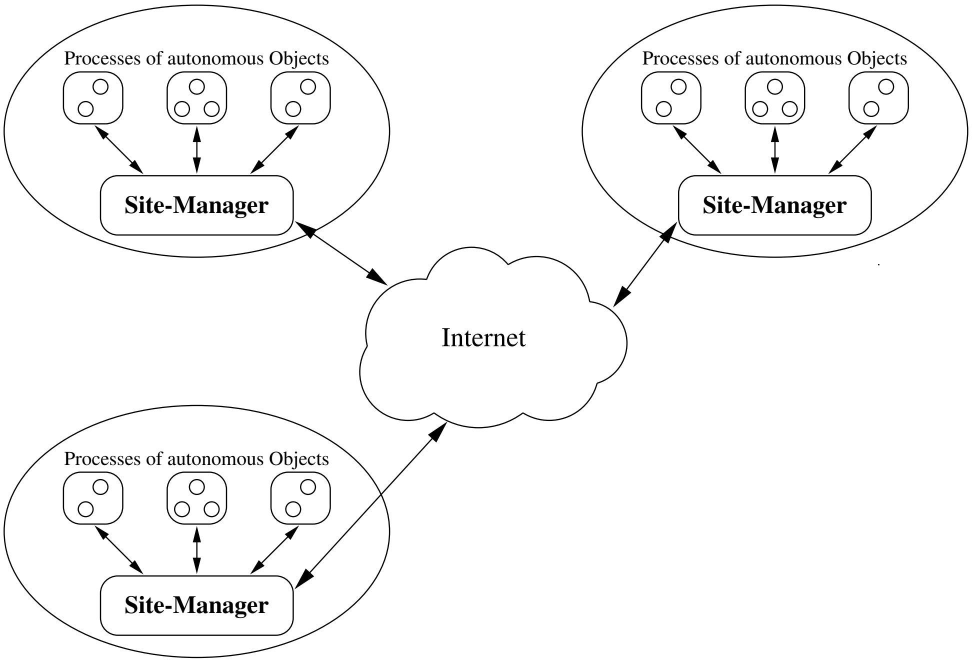

Figure 1: Architecture of a system of autonomous objects

unique identifier. The identifiers are location independent so a name service has to be integrated to locate an object in the system. Based on this model we have implemented a distributed system of autonomous objects, called AutO [KGI+97]. AutO is implemented in Java [AG96] since platform independence is a requirement for a world-wide distributed system. The architecture is given in Figure 1. The system consists of many sites, each of which has one or more processes autonomous objects reside in. Additionally, each site has a *site manager* providing for communication between the sites and coordinating the processes on its site.

3 Object Migration

In a system of autonomous objects the natural unit of migration is an object. An autonomous object encapsulates its internal state and can therefore move easily. In our system objects autonomously decide to migrate and perform the migration on their own, as will be described later.

Many distributed systems provide support for object migration. Surveys of such systems are given in [Bor92, Nut94]. These systems provide migration primitives that can be employed, e.g., by a programmer, to achieve different goals. For example, migration can be used for load-sharing—meaning the utilization of sites whose load is low—for reducing communication costs, reconfiguration, etc., see also [JLHB88, Nut94].

In a wide area distributed system communication costs for a local and a remote access are typically very different. While a local message can reach its destination in less than 1ms, a message sent over the Internet to a remote site may take 500ms or more—even at times when the network traffic is low. Communication costs dominate the response time of transactions accessing remotely located objects inducing much longer response times than those of transactions accessing local objects. Our goal is to reduce communication costs through migration and thereby decrease the response time of transactions.

Of course, no migration policy can achieve that all transactions perform local accesses only. The goal is to significantly reduce communication costs by migrating objects to the neighborhood where most accesses/requests originate. Neighborhood is defined in terms of the speed of network connections—not based on geographic locations.

In a WAN environment an object will participate in the execution of many different tasks some of which might even not be known at the time of the object's implementation. Moreover, usually there will be no knowledge about the overall system's behavior but only about small parts of it, e.g., inside one LAN. Therefore, our goal is a self-tuning system which does not rely on user or programmer hints.

Autonomous objects are enabled to monitor where they are accessed from and calculate, based on these past accesses, whether to migrate or not, i.e., whether migration would lead to (substantially) lower communication costs or not. In a WAN environment the access profile will change over time. This will be observed by the objects and they will automatically adjust to these changes and gradually migrate to where they are needed.

Migrating objects imposes the need to provide a mechanism to efficiently find them in the system. As mentioned before in our system objects have location independent identifiers and a name service is integrated to locate them. Discussions on this problem are given in [EKK97, DLMM93]. The name service developed in [EKK97] is designed to efficiently find objects "in the neighborhood". This guarantees that migrating objects (near) to their respective "centers of attention" will instantaneously improve the system's performance.

There are certain requirements object migration has to meet:

- Transparency and availability
  The migration process should be transparent to the user. This implies that the object is available during migration, i.e., no messages sent to the object are lost.

- Fault tolerance
  In this context fault tolerance means that migration always leads to a "correct" state, i.e., no object is lost or duplicated, even in case failures during migration occur, e.g., on the network.

- Flexible migration policy
  The migration policy of an object should be exchangeable. A system in a WAN environment may change over time so new policies may be more favorable. Also it should be possible to explicitly tell an object to move to a certain site, e.g., because its current site may disconnect from the system.

  In such a system some objects may not be allowed to move, e.g., for security reasons so a "no migration" policy should be provided.

- Efficiency
  Migration, including making the migration decision and the process of migration itself, has to be efficient. However, migration does induce costs so we assume, as

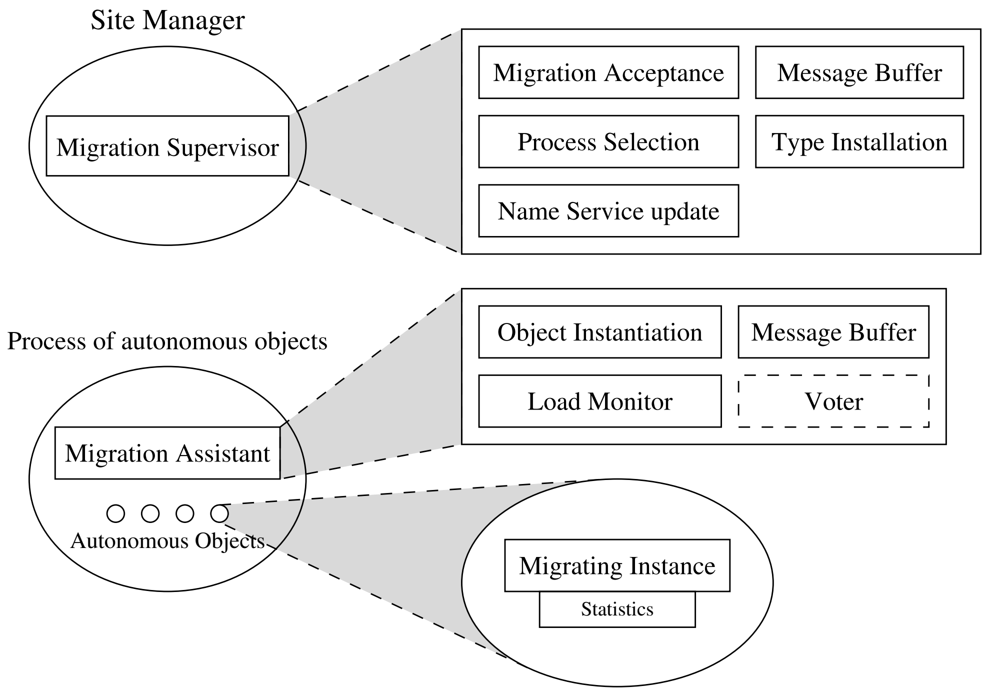

Figure 2: Architecture of the migration component

was done for example in Thor [DLMM93], that a single object does not migrate very often, i.e., that after it has migrated it will stay at its new location for some time.

3.1 Implementation of Migration

As already said the unit of migration in our system is an autonomous object. The object itself decides about migration. The migration is also executed by the object, of course, in cooperation with the involved sites—the current one and the desired location. Objects migrate in an inactive state, i.e., no operation is executed during migration and therefore no thread information is needed. Only the objects' data and type information (in case it is not already available at the new site) have to be sent.

We will describe the implementation of migration for a system having the architecture shown in Figure 1. However, the implementation can easily be adapted for any other architecture. The migration component can basically be divided into the object's part and the part needed by a site in order to coordinate the migration.

The architecture of the migration component is shown in Figure 2. Since objects migrate autonomously each object has a *migrating instance* which collects statistics about accesses transactions made to the object. These statistics are evaluated by the object when it considers migration. Migration consideration is triggered from time to time, i.e., after the object has received at least a certain (fixed, but configurable) number of new messages. New messages are those received after the last time the object has considered to migrate. When migration is triggered the migration policy is evaluated and in case migration would be advantageous migration is initiated. The migration policy is integrated in a way that it can easily be replaced by another policy. The policy can be exchanged dynamically.

Each process of autonomous objects has a *migration assistant*. Its main task is to instantiate an object that has migrated to the process. It is also responsible for buffering messages sent to the object while it is migrating.

The site's load should be taken into account when an object wants to migrate to it. Therefore, each process has a *load monitor* which observes its load and periodically reports it to the site manager. The load is given in messages per unit of time which roughly expresses how much work autonomous objects in this process do.

Interprocess communication is usually more expensive than communication within the same process.[1] Therefore, an object should migrate to the process where most accesses came from. This cannot be decided based on the processes' load so the *migration assistant* can optionally have a *voter*. The *voter* records accesses made by the objects in its process. This information can be used instead of the load to decide to which process an object should migrate.

The coordination of the migration of an object is done by a *migration supervisor* which is part of the *site manager*. The *migration supervisor* decides whether to accept an object's request to migrate to the site or not. The decision is based on the load of processes of autonomous objects on this site. If more than one process of autonomous objects is running on the site the *migration supervisor* chooses to which one an object should migrate. This can be done based on the load the processes report or by asking them to vote, depending on their *voter's* information.

The *migration supervisor* also informs the name service that an object is migrating to its site and buffers messages arriving for the object before it was actually instantiated in the chosen process.

It is possible that not all types needed for a migrating object already exist at the object's destination site. The *migration supervisor* is responsible for obtaining and installing these types (through a class loader).

Now we will present the migration protocol. The message flow diagram in Figure 3 shows a successful migration of object O_m from the old site S_o to the new site S_n.

An autonomous object, here O_m, monitors where it is accessed from and evaluates this information from time to time. In case migration to some site S_n would be advantageous the object O_m sends a migration request to S_n and waits for an answer. Of course, the object is not blocked while it waits but continues to process transaction requests.

Site S_n can accept or refuse the migration request. A refusal ends the migration request. A reason for a refusal could be high load on S_n.[2] However, S_n will usually accept the request. In this case it will start buffering messages arriving for O_m and inform the name service about the object's migration. If there is more than one process on site S_n to which O_m could migrate the *migration supervisor* chooses one of them, either through a vote process or based on the processes' load, as described above. In our example there is only one process P_1.

[1] This is especially true in Java—which we used to implement our prototype—since it does not support shared memory.

[2] What is to be considered as "high load" depends on the number of messages received by the objects at the site and the computational power of the site.

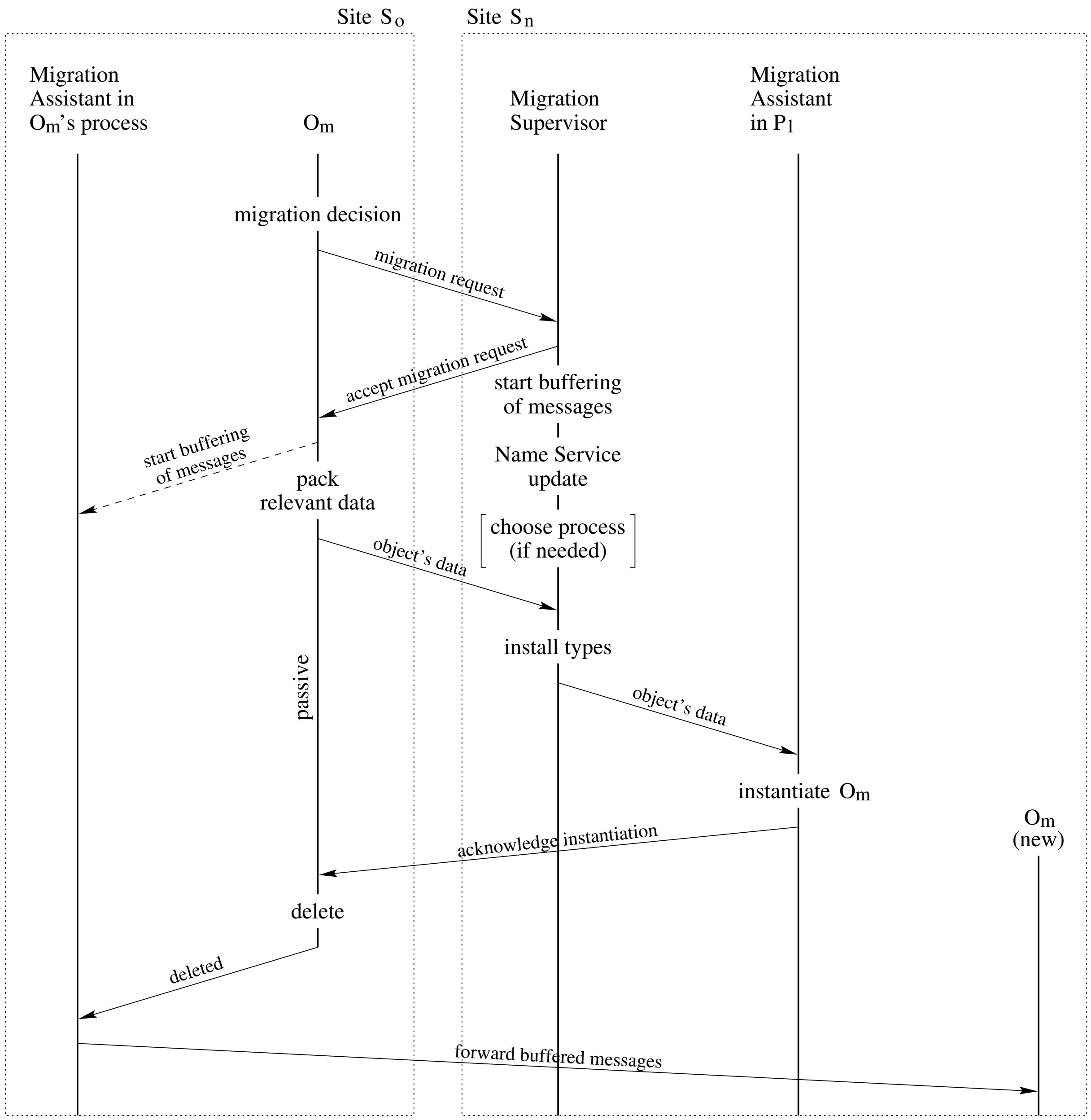

Figure 3: Migration Protocol

Upon receiving the accept message O_m will pack[3] a copy of its data and additional relevant information—such as log and type information—and send it to S_n. O_m informs the *migration assistant*—synchronously (denoted through a dashed line in Figure 3)—that it should buffer all messages addressed to O_m. From now on the object is passive on site S_o.

When site S_n receives the message containing the data it installs the objects types needed for O_m if it does not already have them. The message is then forwarded to the process chosen to get the object. The process "restarts" O_m and acknowledges its successful instantiation. When the acknowledgment is received at S_o the old copy of O_m deletes itself and possibly buffered messages are forwarded to the new location of O_m.

Each of the messages sent can get lost. In order to guarantee fault tolerance for each case precautions have to be taken. In general, sending a message has to be associated with a timeout specifying the time the sender waits for an answer. In case the timeout expires before the answer is received the message is considered lost. Actions to be taken in this case depend on the state of the migration process in which the failure occurs. Basically, in case O_m's data have not been sent yet, O_m aborts the migration. If a failure occurs later in the protocol the "missing" step is repeated until it succeeds. This implies that except in case the step in which O_m's data are sent fails, O_m can process messages at one of the sites S_o or S_n. A failure during the sending of O_m's data postpones the instantiation of O_m at site S_n while it is already passive at site S_o. During this time O_m cannot process messages, but no messages are lost, since they are buffered at both sites.

There is an extension to the described protocol that allows a site manager to forward a migration request to another site within the same LAN. The other site can again accept the request or forward it to another site. If none of the sites accepts the request the site in the LAN that gets it last will send a refuse message to the object. In order to get forwarded the migration request has to be marked by the requesting object. If a request is not marked it will be refused by the first site. This extension may be useful in case a site has to refuse a migration request coming from a distant site. In this case it will probably be beneficial to the object to move into the neighborhood of its desired location, i.e., to a site within the same LAN.

In a system supporting decentralized migration there is no guarantee for load-balancing. In smaller configurations, e.g., comprising a single LAN, this may lead to an imbalance since many objects might try to migrate to the same site, e.g., the fastest machine, since most accesses are sent from there. In a WAN environment this will not be the case since the requests will come from different LANs and therefore the load will automatically be distributed over these LANs. Therefore, we provide only the described load-balancing mechanism which considers only sites within one LAN.

3.2 Migration Policies

The crucial part of migration is the policy based on which objects decide whether to migrate or not. Our goal is a self-tuning system so objects have to monitor where they

[3] In Java packing means serializing, i.e., creating a stream of bytes which can be sent by socket communication facilities.

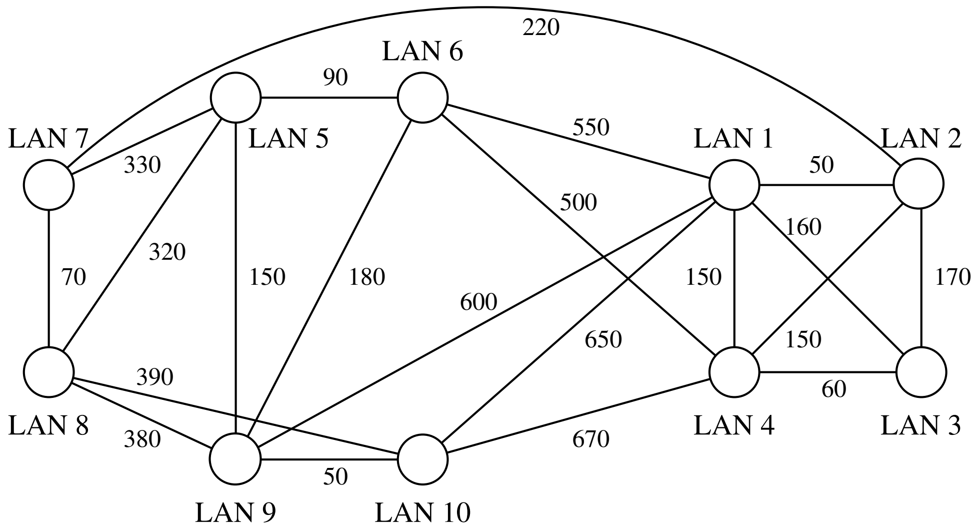

Figure 4: Example of a network structure

are accessed from and choose the best location assuming that accesses in the near future will be similar to those in the recent past.

In a LAN environment the most profitable site to move to—regarding communication costs—would be the one from where most of the accesses come from. This is not true in a WAN environment since the costs for accesses from different sites may vary largely and therefore have to be considered.

Before presenting the migration strategies we will take a look at the network configuration that is to be expected in a WAN-distributed system.

The network structure of an enterprise being represented at different continents will consist of many LANs spread over these continents. The "network distances" between these LANs will be different, a small (and incomplete, since not all connections are shown) example is given in Figure 4. Distances between LANs are given in milliseconds. Of course, the communication costs are not constant but an average value can be determined. In order to obtain these costs the average of many measurements performed at different times has to be taken—so different network loads are covered. Costs for accesses within a LAN are not given. They are assumed to be constant and equal for all LANs.[4] Each process of autonomous objects has a copy of the system's structure. Based on this structure an object can easily calculate the communication costs of the accesses made to it. The configuration of such a system is not static and will change over time but it can be assumed that such changes will not occur too often. A new site being added to the system has to be registered, either as a member of a LAN already existing in the system or as a member of a new LAN. This information is given to a *structure agent*. In case of a new LAN the network distances to other LANs have to be determined. It can either be provided by the site upon registering—if it has the knowledge—or determined by the *structure agent* by measuring the average message transfer times between the new and the other LANs. The knowledge is then propagated to the site managers. A site can also leave the system and

[4]This could be made more accurate but the costs do not vary much and would not make a significant difference considering the much higher costs for communication between different LANs.

again inform the *structure agent* about it (in case this is not only a temporary change).

Both strategies we present consider only the problem of determining the LAN an object should migrate to. Within a LAN the object chooses the site most accesses came from. In case this site cannot accept the request it is forwarded according to the protocol described above.

Of course, both strategies take into account the possibility that it could be beneficial for an object to migrate within the same LAN.

3.2.1 Finding-The-Best-LAN Strategy

In a WAN environment a migration decision cannot be made purely based on counting accesses from different LANs but the network distances also have to be considered. The best placement for an object can be in a LAN whose transactions rarely access the object, e.g., if it is "in the middle" between two LANs many accesses come from.

When an object checks whether it should migrate or not it considers only a certain number of the latest messages it has received.[5] In order to find the best LAN the object calculates the communication costs these messages would have caused in case the object had been located in the other LANs in the system. Accesses from within the same LAN are not taken into account here, the costs they induce are assumed to be much lower than for accesses from outside the LAN. The costs that would have been encountered in the best LAN, i.e., the one with the minimum costs, are denoted as C_{best_lan}.

The object decides to migrate in case the profit is higher than a certain threshold t, explained below:

$$C_{cur\_lan} - C_{best\_lan} \geq t.$$

In case migration to a different LAN would not be profitable enough, the object additionally calculates the communication costs in case it would have been located at the site within its own LAN where most accesses came from, denoted as C_{best_site}. In many cases this will be the object's current site, so no computation will be needed. Again it decides to migrate if this would reduce the communication costs by a given threshold.

The value for t represents the profit that should be achieved in order for the migration to be beneficial. We request that the communication costs are reduced per message on average by some time c. Since communication costs within a LAN will be much lower than between different LANs, two values for c should be used. When considering migration to another LAN, c_{global} is used to determine the threshold. In case the object considers migrating within its current LAN, c_{local} will be used. The reason for this differentiation is that migration should be initiated only in case communication costs can be reduced significantly. Therefore, the threshold has to be high which would imply that an object would never migrate within its own LAN if only one c is used. However, c_{local} should be set in a way that migration to a site within the same LAN will happen only in case many more accesses from within the LAN originate from this site than from the current site. The threshold t is then calculated as:

$$t = c \times \text{number\_of\_considered\_msgs}.$$

[5]The number of messages considered is not to be confused with the number of new messages needed to trigger migration evaluation. They could be the same but they do not have to.

Note that costs for migration are not taken into account since we assume that after migration an object stays at its new location for some time, so the (relatively low) migration costs are negligible.

The pseudo code for the described strategy is given below.

- Object O_i, located at site S_i, in LAN L_i considers migration

```
SiteId considerMigrationBestLan()
  C_cur_lan := calculateCommCosts(L_i);
  best_lan := L_i;
  C_best_lan := C_cur_lan;
  for each LAN L_j ≠ L_i
     costs := calculateCommCosts(L_j);
     if (costs < C_best_lan)
     then C_best_lan := costs;
        best_lan := L_j;
     endif
  endfor
  if (C_cur_lan − C_best_lan ≥ c_global × nr_of_cons_msgs)
  then return id of site in best_lan with most accesses;
  endif
  if (L_i = best_lan)
  then best_site := site within L_i where most accesses came from;
     C_best_site := calculateCommCosts(best_site);
     C_S_i := calculateCommCosts(S_i);
     if (C_S_i − C_best_site ≥ c_local × nr_of_cons_msgs)
     then return id of best_site;
     endif
  endif
  return id of S_i;
```

The method *calculateCommCosts* computes the communication costs that the considered messages would have caused in case O_i had been placed in a certain LAN or site.

We assume that for sites within one LAN communication costs caused by messages sent from other sites outside this LAN are the same. Therefore, when O_i checks whether it should migrate within its own LAN L_i only messages local to L_i have to be considered. Actually, when O_i considers migrating to S_j (=best_site) only messages local to S_i and S_j have to be taken into account. The savings in communication costs are expressed as:

$$\mathrm{C}_{S_i} - \mathrm{C}_{S_j} = (\mathrm{nr\_msgs\_S}_i - \mathrm{nr\_msgs\_S}_j) \times (\mathrm{lan\_comm\_costs} - \mathrm{local\_comm\_costs}).$$

Here, local_comm_costs and lan_comm_costs are communication costs for a message within one site and for a message within the LAN, respectively.

3.2.2 Finding-A-Good-LAN Strategy

The first strategy we have described finds the location inducing the least communication costs for the object. However, no matter how efficient the implementation of the strategy

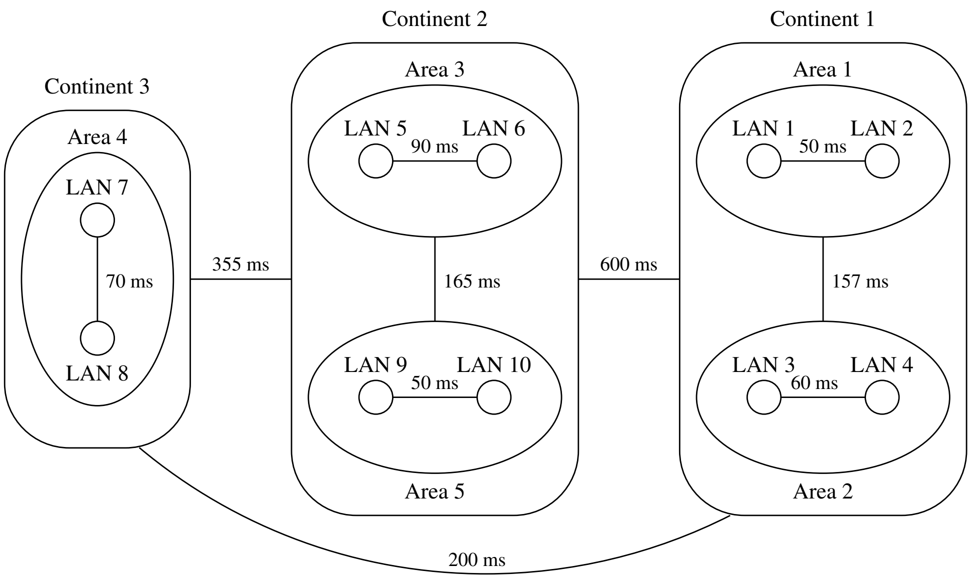

Figure 5: Example for a hierarchical structure

is, in a system comprising many LANs it will be too expensive since it is invoked often. Therefore, we will present a simpler strategy that in most cases finds the best location, but may sometimes return a "sub-optimal" placement.

In the previous strategy we looked at a "world" consisting of many LANs connected through a network. However, in the real world these LANs will build a hierarchy of LAN clusters based on the distances between them. LANs located close to each other, regarding the network distance, could be seen as clusters. These clusters could be joined to build bigger clusters, based on the average transfer time of messages between them. We will use geographical metaphors in our discussion of network clusters; even though geographical distances may not always properly reflect the network distances. Therefore, a cluster built out of LANs will be called an *area*, and a cluster of areas will be referred to as a *continent*. The network distance between two clusters is given by average message transfer times between them which is determined as the average of the distances between each of the sub-clusters they comprise.[6]

The hierarchy of clusters corresponding to the structure in Figure 4 is given in Figure 5. The algorithm used to create the cluster hierarchy is described later. In this example areas contain LANs between which the distance is less than 100ms. Areas belong to the same continent in case the distance between them is on the average not more than 200ms. In our example we have only a three level hierarchy, but this in not a restriction given by the model. In case connections between continents differ largely *planets* could be built—representing clusters of continents.

Given such a hierarchical structure, the migration policy can be simplified. When an object checks whether it should migrate or not, in the first step it considers only continents. It again takes only a certain number of messages into account and calculates the communication costs these messages would have caused if it had been located at the

[6]It would also be possible to "define" a center of a given cluster and determine the distance to another cluster as the average message transmission time between their centers.

different continents. Only inter-continental messages are considered in this step. Note that the calculation is based on the average communication costs between the continents given in the hierarchy. For example, if in Figure 5 an object O located at LAN1 had received ten messages from LAN10 they would have contributed 6000ms to the communication costs. The winner is the continent having the lowest inter-continental communication costs.

Within the best continent the object chooses the best area, i.e., the one that would have caused the least costs in case the object had been located in it. For all areas within one continent the inter-continental costs are equal and correspond to those calculated in the previous step. This implies that only messages from within the continent are considered in this step. The costs they would cause are then added to those previously calculated. For the winning area the object determines the LAN at which the communication costs would be the lowest. As in the previous strategy at the LAN level the object assumes that it would be located at the site of the LAN where most accesses come from.

The decision whether to migrate or not is made the same way as described for the Best-LAN strategy. The object compares the communication costs at its current location with those that would have been encountered at the LAN chosen by the strategy. In case migrating to this LAN would not be profitable enough the object considers migrating to the "best site" in its current LAN. The threshold is determined as described for the Best-LAN strategy. The pseudo code for the Good-LAN strategy is given below.

- Object O_i, located at site S_i, in LAN L_i, in area A_i, considers migration

```
SiteId considerMigrationGoodLan()
  C_cur_lan := calculateCommCosts(L_i);
  best_continent := findBestContinent();
  best_area := findBestArea(best_continent);
  best_lan := findBestLan(best_area);
  if (C_cur_lan − C_best_lan ≥ c_global × nr_of_cons_msgs)
    then return id of site in best_lan with most accesses;
  endif
  if (L_i = best_lan)
  then best_site := site within L_i where most accesses came from;
      C_best_site := calculateCommCosts(best_site);
      C_S_i := calculateCommCosts(S_i);
      if (C_S_i - C_best_site ≥ c_local × nr_of_cons_msgs)
      then return id of best_site;
      endif
  endif
  return id of S_i;
```

The methods *findBestContinent, findBestArea* and *findBestLan* return the continent, area within that continent or LAN within that area, where O's messages would have had induced the least communication costs, respectively. Of course, they also calculate the corresponding communication costs.

Building Clusters

The Good-LAN policy works on a hierarchy of clusters. These clusters have to be built based on the network distances between the LANs in the system, an example of which was given in 4. These distances are usually presented in a graph. To build the clusters the graph has to be analyzed and LANs that are close together are united to one cluster. This can be done by using clustering algorithms. For our system we have adapted the *greedy graph partitioning* algorithm [GKKM93] since it achieves good results in reasonable time.

In the first phase the algorithm sorts the edges of the graph in decreasing order according to their weights, i.e., the distances between the LANs the edges connect. In the beginning of the second phase the algorithm considers each LAN as a cluster. Clusters which are "close enough" are merged. Close enough means that the maximum distance between LANs in these two clusters is less than a given diameter. The algorithm searches through the sorted list of edges to find the clusters. When no more clusters can be merged the algorithm stops. In order to get a hierarchy of clusters the algorithm is iterated. In each iteration it builds clusters of clusters it has created in the previous iteration. Of course, the diameter is increased each time.

How many hierarchy levels are to be built and the size of the diameter for each invocation is given to the *structure agent* which is responsible for the clustering. The number of levels is a trade-off: many levels lead to more accurate migration decisions but also induce higher computation costs.

For the example in Figure 4 a three level hierarchy was built, using diameters of 100 and 200ms, respectively. In the first iteration areas 1–5 were clustered, out of which three continents were created in the second iteration. The resulting hierarchy is shown in Figure 5.

Changing the Hierarchy

The algorithm described above will create a hierarchy in which all clusters of one level are disjoint. When a new LAN is added to the system it can join an existing area or a new one is created. If the new area is too far away from the existing continents also a new continent has to be created. The system administrator adding the site to the system can also give hints on where the LAN should be placed. These changes are also managed and propagated by the *structure agent*. The *structure agent* has to monitor the changes to the structure and in case major changes occur, which is not likely to happen often, it should rebuild the hierarchy. Note that the structure does not have to be completely accurate but should roughly reflect how the LANs in the system are located.

Adding a new site to an existing area does not affect the hierarchy but inserting a new cluster may recursively alter the distances of its "super-clusters" to other clusters of their level. For example, a new LAN in an already existing area may change the network distances of this area to other areas and thereby also the distances of the area's continent to other continents. However, the *structure agent* can check whether these changes are significant or not, and depending on this decide whether to propagate them or not. For this purpose the *structure agent* looks at the network distances of the new LAN to some other LANs distributed over the "world". The chosen LANs should belong to a couple of different areas. The *structure agent* then compares the distances between the new LAN and the chosen LANs with the current distances between the corresponding areas. If the deviation is not too high the *structure agent* decides not to propagate the changes but only the existence of the new LAN and its distances to other LANs within its area. This way only major changes are propagated. Smaller changes are kept by the *structure agent* in a separate "working copy" of the hierarchy and are propagated only when they accumulate to a significant one. Of course, the *structure agent* uses this copy for all further actions. Note that this way a LAN can be added to the hierarchy before all the distances to other LANs are known.

When a new LAN causes the creation of a new area then its distances to other areas of its continent have to be determined—by measuring these distances—but the distances of the continent to other continents may remain unchanged. This can be applied recursively. Assume a new LAN leads to the creation of a cluster C at a hierarchy level l. Then also at all levels below l new clusters will have to be created. Let SC be the direct "super-cluster" of C, i.e., the cluster at the next hierarchy level C belongs to. Then the network distances of all new clusters to the clusters within SC of the same level have to be determined and added to the hierarchy. The changes of network distances at levels beyond l will be determined by the *structure agent* but in case the changes are not significant this may happen after propagating the existence of the clusters to the system's sites.

Even though changes to the hierarchy do not happen often when the system runs for a long time they might accumulate. In this case the hierarchy should be rebuilt, using the clustering algorithm given above. This could, for example, be done after a certain number of propagation changes.

3.2.3 Comparing the Strategies

As said before the Best-LAN strategy always finds the best placement for an object while the Good-LAN policy may fail to find the optimal location and return a sub-optimal placement decision. The reason is that the Best-LAN strategy bases its calculations on the average communication costs between LANs while the Good-LAN strategy operates with the average communication costs between the clusters. The underlying assumption is that the origins of messages sent to an object from one cluster are uniformly distributed over the sub-clusters of the cluster. In this case the average communication costs the messages induce would correspond to the costs the Good-LAN strategy uses for its calculations. The strategy may derive a sub-optimal decision in case the majority of messages sent from a cluster comes from sub-clusters with communication costs deviating from the average costs given in the hierarchy.

Even in case the communication costs do not deviate from the average costs given in the hierarchy the Good-LAN strategy may return a sub-optimal placement, since it cannot detect hot-spots at the sub-clusters' level. The algorithm simply assumes that the messages' origins are uniformly distributed within the cluster. Messages coming from a cluster are simply counted without taking into account where in the cluster they come from. If there is a hot-spot at the sub-clusters' level in one cluster this may significantly reduce communication costs within this cluster, so a previously "more expensive" cluster may become the "cheapest" one.

An example in which the first strategy finds a better placement for an object than

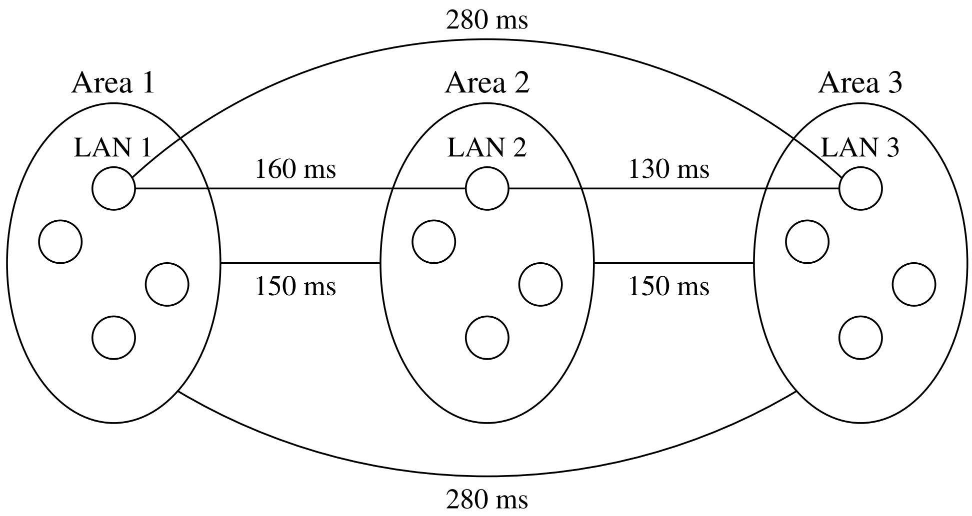

Figure 6: Example for Best-LAN and Good-LAN determining different placements

the second one is given in Figure 6. For simplicity we assume a world consisting of only three areas. An object O residing in LAN2 evaluates whether it should migrate or not. Distances between the areas and between the relevant LANs are given in the figure. Sites located in Area1 have sent 200 messages to object O, sites at Area2 and Area3 have sent 100 and 190 messages, respectively. Assume that all messages from Area1 have been sent from sites in LAN1, while messages originating at Area3 have been sent from sites in LAN3.

The Good-LAN strategy would calculate the following communication costs:

$$\begin{array}{lclcl} C_{Area1} & = & 100 \times 150ms + 190 \times 280ms & = & 68200 \\ C_{Area3} & = & 100 \times 150ms + 200 \times 280ms & = & 71000. \end{array}$$

C_{Area1} and C_{Area3} are inter-area communication costs that would have occurred in case O had been located at Area1 or Area3, respectively. O would choose Area1 as the best area and determine that LAN1 is the best LAN to migrate to. Depending on the threshold it would either decide not to migrate or to migrate to LAN1.

The Best-LAN strategy would consider the distances between the LANs and calculate the following communication costs:

$$\begin{array}{lclcl} C_{LAN1} & = & 100 \times 160ms + 190 \times 280ms & = & 69200 \\ C_{LAN3} & = & 100 \times 130ms + 200 \times 280ms & = & 69000. \end{array}$$

Based on this calculation O would consider migrating to LAN3, again depending on the threshold.

The placement the Best-LAN strategy has found is better than the one found by the Good-LAN strategy, since the latter considered only inter-area communication costs. However, the computational costs for the Good-LAN strategy are much lower, as will be shown in Section 4, since it calculates only the communication costs for each area and afterwards considers only LANs within the best area, while the Best-LAN strategy has to calculate the costs for each LAN in the system. Moreover, in most cases the decision whether to migrate or not will not be so tight so the Good-LAN strategy will mostly also lead to the best solution. This is also confirmed by the experiments we present in Section 4, which will show that the Good-LAN strategy performs almost as good as Best-LAN.

4 Performance Evaluation

In this section we present a quantitative evaluation of the Good-LAN and the Best-LAN migration policies. First we will compare the strategies in a large synthetic configuration. This setting is used to compare the computational costs the strategies induce and the quality of the decisions they derive. Secondly the migration policies are analyzed in a smaller (semi-)real configuration in order to evaluate them in an operational system.

Generally, the results we report constitute averages of several runs. The number of measurements per experiment depends on the variation of the results. The standard deviations of the recorded results are shown (as error-bars) in the graphs.

4.1 Separate Implementation of the Strategies

4.1.1 Computational Costs for the Migration Decision

In Section 3 we have shown that the Good-LAN strategy may fail to find the best placement for an object but we also stated that it induces less computational costs than the Best-LAN strategy. Here we will show the quantitative difference between these two strategies.

In order to achieve accurate results we have separately implemented the decision making procedure of the strategies. We have measured the CPU time the strategies need depending on the number of sites in the system. Each strategy had to determine the LAN where the communication costs for the given set of messages would have been as low as possible. The implementation was done in C++ and the experiment was conducted on a Sun Sparcstation Ultra I running Solaris 2.6.

Parameters for the experiment are given in Figure 7. As described in Section 3.2, both strategies make their migration decision on the LAN-level and only in case the object considering migration should migrate one of the sites within the chosen LAN is selected. Additionally, in case migration to another LAN would not be profitable enough, the site from the current LAN that has made most accesses to the object is considered. In this experiment sets of (five) sites were grouped to a LAN. For the Good-LAN strategy a hierarchy of clusters is needed. In the presented experiment the number of continents was fix. The number of areas was chosen randomly within the range $[2 \times \text{no\_of\_conts}, \text{no\_of\_lans} \div 2]$. On each continent at least two areas were placed. The distances between the LANs have been generated randomly but in a way that they are compatible with the hierarchy built for the Good-LAN strategy. Note that the distances between the LANs have no influence on the results measured in this experiment. The number of considered messages was 1000. The origins of the messages were distributed uniformly over the sites in the system.

The results of the experiment are presented in Figure 8. The graph shows that for small numbers of sites both strategies need about the same amount of time to decide

No. of sites	100, ..., 5000
Sites per LAN	5
Hierarchy levels	3 (LANs, areas, conts.)
No. of areas	[10, no_of_lans ÷ 2]
No. of continents	5
No. of considered msgs	1000

Figure 7: Parameters for Measuring Computational Costs

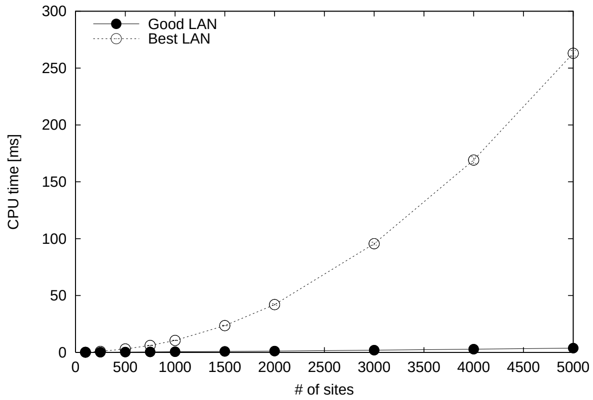

Figure 8: Computation Costs for Migration Decision

which the LAN with the lowest communication costs is. When the number of sites is increased the differences between the run times become significant. For 5000 sites, i.e., 1000 LANs, the Good-LAN strategy is almost 90 times faster than the Best-LAN strategy.

One could argue that for both strategies the calculation lies in the range of milliseconds and will therefore not make a significant difference on the overall system's performance. However, the migration strategy is invoked every time an object considers migration which can happen quite often in case many objects reside in the system. Moreover, an object considers migration after it has received a certain number of messages, which usually implies that the object has work to do. Most likely the home site of an object will not be idle when migration is considered since in most cases not only one object at a site will be active at a time. Therefore, it is even more important that the migration decision does not take too long.

4.1.2 Quality of the Migration Decision

The separate implementation of the strategies described above was also used to analyze the quality of the migration decisions of the strategies. We compared the communication costs that would have occurred in case the object whose messages were analyzed had been placed at the site chosen by the strategies.

As in the experiment described in the previous section 1000 messages were generated. Both strategies computed where the object that has received these messages should migrate to or whether it should stay at its current location. The parameters for the measurements were the same as before, i.e., as given in Figure 7.

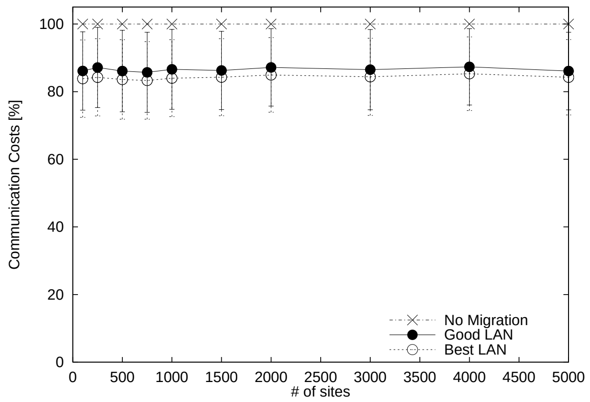

Figure 9: Communication Costs (uniform message origin distribution)

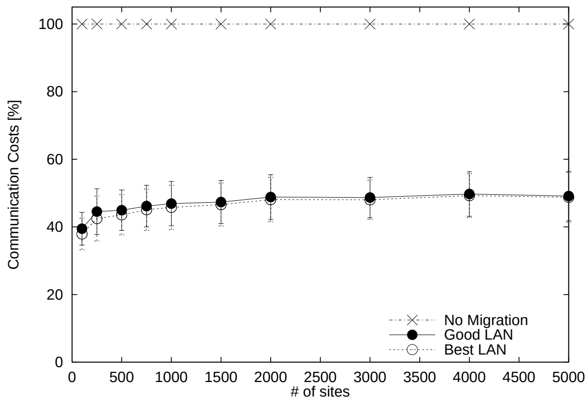

Figure 10: Communication Costs (80% from one continent)

First we have investigated the quality of the strategies when the origins of the messages were distributed uniformly over the sites in the system. The results are shown in Figure 9. At the first glance it is surprising that even though there is no locality of the message origins the migration strategies find a placement where the communication costs would be about 15% lower than at the current location. The strategies reduce the communication costs by deciding to migrate to locations that on the average have the fastest connection to other sites in the system.

The location determined by the Best-LAN strategy would induce about 2% less communication costs than the one found by the Good-LAN strategy. The reason is that the Good-LAN strategy bases its decision on average costs between the clusters of the system while Best-LAN uses the average costs on the LAN level. The price Best-LAN pays for the precision has been discussed in Section 4.1.1.

It is not to be expected that the origins of the latest accesses to an object will be uniformly distributed over the world. Therefore, we have also investigated how the strategies perform when 80% of the messages originate from one randomly chosen continent. The results are shown in Figure 10. Here the placement the strategies choose would reduce the communication costs by more than 50% due to the high locality of the origins of the messages.

For smaller configurations the savings in terms of communication costs are higher than for larger ones. The reason is that in a smaller configuration there is a smaller number of areas per continent. Assuming a uniform distribution of the origins of messages within a continent, as has been the case in our measurements, migrating to a continent with a small number of areas implies that many messages coming from this continent will now be not only local with respect to the continent but also area-local. With many areas on a continent the share of area-local messages decreases.

The results of the two strategies in this measurements were alike, so considering the computation costs Good-LAN would be the better choice.

No. of continents	4	Dist. between conts.	[200, 700]
No. of areas	5	Dist. between areas on one cont.	[100, 200]
No. of LANs	10	Dist. between LANs in one area	[20, 100]

Figure 11: Configuration for experiments in AutO

4.2 Experiments in AutO—A Distributed System of Autonomous Objects

4.2.1 The Experiment Environment

Except for the experiment presented in Section 4.1 the performance evaluation was done in AutO [KGI+97], a distributed system of autonomous objects we have implemented based on the model described in Section 2. For this presentation we will abstract from a full description of the system. AutO has a peer-to-peer architecture, basically corresponding to the one shown in Figure 1. At each of the involved sites a site manager and one (or more) AutO-processes, in which autonomous objects reside, are started.

Autonomous objects communicate via asynchronous message passing. The activity of an object is represented through a thread an object becomes associated with in case it has work to do. Transactions are also autonomous objects, as described in Section 2, realized through two threads, one performing the transaction's task, e.g., accessing objects, and the other one having the role of a transaction manager.

The system is designed for a wide-area environment so sites are connected through the Internet. The system is fully implemented in Java, so it runs on a wide variety of platforms.

The experiments were done in a (semi-)real system but due to the lack of computing facilities our configurations were relatively small, comprising up to ten sites. Also, these ten sites were not distributed around the world but were located in two LANs with a fast network connection between them. In order to evaluate the migration policies in a WAN like environment we have artificially prolonged network latency between the sites. This way we could simulate a network in which the sites were far away from each other, each of them belonging to a different LAN.

For the experiments we present here we "distributed" the sites over four ("virtual") continents, see Figure 11. On one of the continents two areas were placed while the others comprised one area only. In each area two LANs were located. The network distances are also given in Figure 11. For the Best-LAN strategy distances between all LANs are needed. We will not give the whole distance matrix here, the distances are compatible with those given in Figure 11.

For the Good-LAN strategy the hierarchy of clusters of sites has to be created. The configuration of a system in a wide-area environment does not change very often, especially not in a way that has an effect on the hierarchy. Therefore, in the experiments a fixed hierarchy has been used.

For the experiments we modeled an application that could for example be a very simplified flight booking system. In this kind of applications the response time of the transactions is clearly dominated by the communication costs. Parameters for the experiments are given in Figure 12. There are two types of objects which we will call *airlines*

No. of airlines	20
No. of flights	200
No. of trans. at full load	17 per site
LoyalTa	
Transaction size	[4–8]
Access pattern	2 preferred airlines 1 random airline
Share	60%
RandomTa	
Transaction size	[4–8]
Access pattern	3 random airlines
Share	30%
SimpleTa	
Transaction size	[2–3]
Access pattern	1 random airline
Share	10%

Figure 12: Parameters for experiments in AutO

and *flights*. Each airline "offers" ten flights. For each continent four airlines are marked as "preferred", which is needed for some transaction types. Customers are represented through different transaction types: *LoyalTa, RandomTa* and *SimpleTa*. LoyalTa and RandomTa transactions access three airlines, each airline returns between one and five of the flights it offers, i.e., their identifiers, and the transactions then access one third of the flight objects returned to it. The difference between these two transaction types is that LoyalTa transactions access one randomly chosen airline and two of the preferred airlines of the continent they are started at while RandomTa transactions choose all airlines they want to access randomly. Transactions of type SimpleTa access one randomly chosen airline only and book half of the returned flights.

In a WAN distributed system the load will most likely not be as homogeneous as described above. In such a system it is to be expected that the transaction's size will vary more. However, since objects base their migration decision only on the communication costs caused by messages received—not considering which transactions have sent these messages—the transactions' size does not have a significant impact on the results.

When running at full load a site has 17 transactions concurrently processing on it. An aborted transaction is automatically restarted while a committed transaction is replaced by a new one. The type of the newly started transaction is chosen randomly according to the share the transactions types have in the overall load, see Figure 12.

In the experiments only one name service instance was used and requests to it were not delayed. The effect of this is as if a name service is located at each LAN and all of them are updated every time a migration occurs.[7] Additionally, all object types were known at each site, so the classloader was not needed.

Our goal is a decrease of the transactions' response time through the reduction of

[7]Despite the eager updates messages to an object that has migrated might at first be sent to a wrong site. One reason is that an update does not happen instantaneously and the other is that sites may have cached the old location of the object.

communication costs. Since the experiments were done on a real system and we did not have exclusive access to the machines we used, measuring the response time would not lead to meaningful and comparable results. Therefore, the main focus in the experiments were the communication costs.

For each transaction its communication costs were calculated based on the message delays imposed on the system through the prolonged network latency. However, we additionally have to consider the costs a transaction encounters due to the migration of an object it waits for. When an object migrates it takes with it all requests (messages) it has received but has not yet processed. The costs for sending the migrating object to its new destination are added to the communication costs of all transactions waiting for this object. The costs for forwarded messages are also added to the communication costs of the transaction that has sent the message, even if multiple messages were forwarded in one message. Note that this way each message received at the object's old location is treated as a separately forwarded message, which is usually not the case and therefore disfavors the migration policies when compared to the system without migration.

We ran the experiments in AutO with both migration strategies. For both strategies the average communication costs of the transactions and the percentage of the messages sent within the same LAN or area were determined. The results for the case without migration were also calculated from the runs with migration. For the communication costs this was done by calculating the costs the requests of a transactions would have caused in case all object would still reside at the site it has been created at. The share of messages that would have been local without migration can be determined in a straight forward way, by counting messages from the LAN (area) the objects originates from.

4.2.2 Scenario 1: Randomly Placed Objects, Constant Load

In this scenario the objects were randomly placed at the sites in the system and full load was introduced to all sites during the whole experiment. In a WAN distributed system this is not a realistic scenario since the load will usually not be equally distributed over all sites in the system. Such a distribution makes the migration decision difficult so the scenario shows the behavior of the migration strategies in a "bad case".

Since we have a relatively small setting the objects considered migration after every 40 messages they have received. The decision was also based on the last 40 messages.

The average communication costs per transaction over time are shown in Figure 13. The load in this experiment is constant so the communication costs without migration are stable. When the migration strategies are employed the communication costs decrease over time. The decrease in the communication costs is slow since the locality of the accesses made by transactions is low, cf. Figure 12. Due to the load distribution in this scenario objects are constantly accessed from many different sites, so for most objects there will not be a "best location" in the system. However, even in this scenario the migration strategies can reduce the communication costs per transaction by up to 34%, see Figure 13.

In this experiment the migration strategies perform alike, i.e., there is no significant difference between the reduction in communication costs they achieve. The reason is that in the given scenario, as mentioned before, there is not one best placement for an object

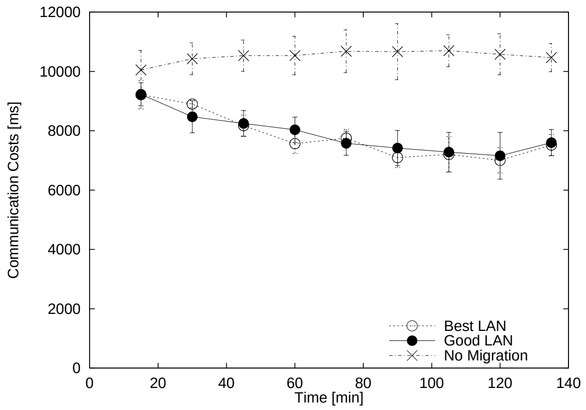

Figure 13: Communication costs over time

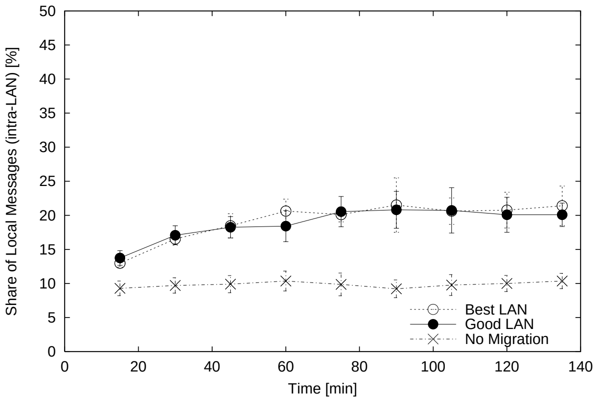

Figure 14: Share of messages in local LAN

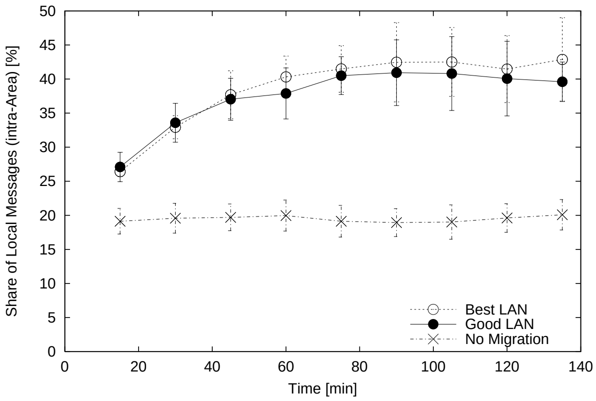

Figure 15: Share of messages in local area

but a couple of good ones.

Figures 14 and 15 show the percentage of messages sent locally within a LAN and area, respectively. Even though the locality of the accesses made by transactions is low the migration strategies achieve that up to twice as many messages were sent within a LAN compared to the case without migration. The percentage of messages sent locally within the area could also be doubled. Again, there is no significant difference between the strategies.

In this experiment the Good-LAN strategy is clearly the better choice. It performs as good as the Best-LAN strategy but, as has been shown in Figure 8, needs much less computation for the migration decision.

4.2.3 Scenario 2: Randomly Placed Objects, Load Shift

In this scenario we analyze the behavior of the migration policies when load shifts occur in the system—which is likely to happen in a WAN distributed system.

To implement load shifts we have distributed the continents over three different "time zones". The load has periodically been shifted through them—mimicking the shift of activity encountered in the real world between different time zones—as shown in Figure 16.

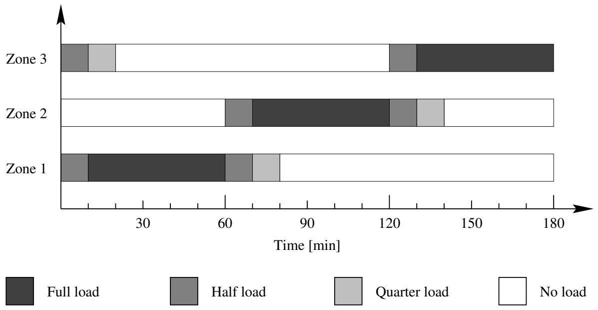

Figure 16: Load shift over time

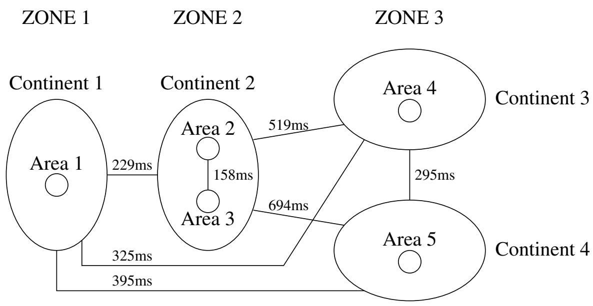

Figure 17: Experiment Configuration

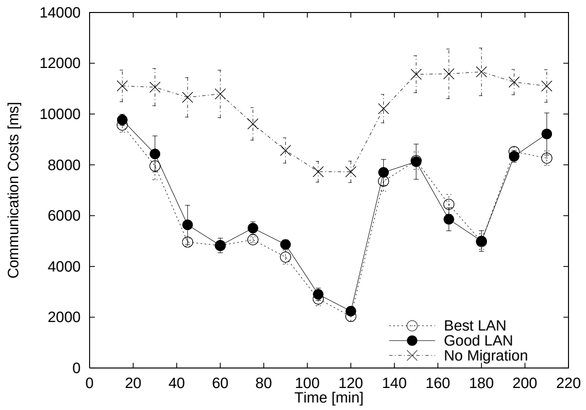

Figure 18: Communication costs over time

The activity in one zone starts with half load, increases after ten minutes to full load at which it stays for 50 minutes. After this period the load is reduced to one half and afterwards to one quarter, for ten minutes each. Finally, no load is introduced to the site for 100 minutes. A full "load cycle" takes 180 minutes. At the beginning of the measurements sites located at continents in zones 1 and 3 were running at half load.

The number of messages needed to trigger the evaluation of the migration strategy and the number of messages considered for the decision was again 40.

In order to investigate the migration under different aspects the configurations on the time zones differed, see Figure 17. In zone 1 one continent with two areas has been located. Zone 2 consists of one continent with one area while zone 3 consists of two continents comprising one area each. The network connections of the continent in zone 2 to other continents were the fastest ones. The connections of the continent in zone 1 to other continents were faster than those of the continents in zone 3.

The average communication costs per transaction are shown in Figure 18. In this scenario the costs without migration are not stable but vary depending on the load. This is due to the differences in the speed of network connections between the continents in the different zones and within the continents itself. When a continent has fast connections the average communication costs are lower than when the connections are slow.

In case the migration strategies are employed the communication costs are significantly lower. The load shifts are clearly visible in the graph: Every time the load moves to a different zone the communication costs increase—because the objects are still placed at

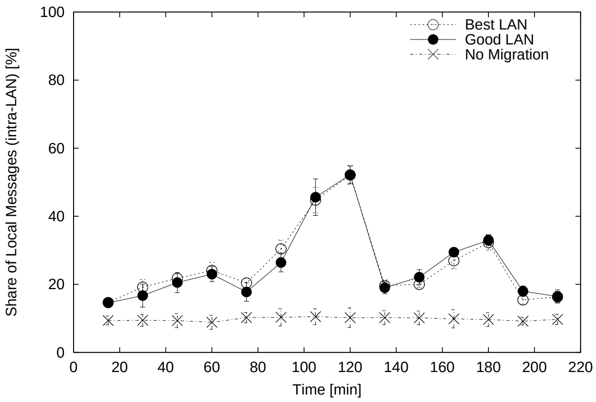

Figure 19: Share of messages in local LAN

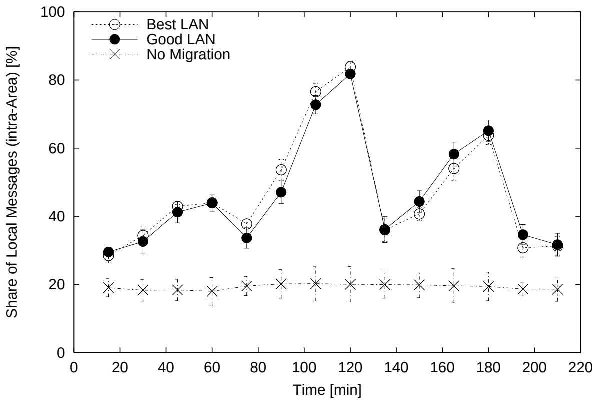

Figure 20: Share of messages in local area

the sites that had previously run at full load. For example, at dusk in Zone 2 and dawn in Zone 3 (after 120 minutes of the benchmark) a significant increase in communication costs can be seen in the graph. The costs decrease again as soon as the objects compensate the shift of activity by migration. The decrease in communication costs achieved by the strategies is up to 73%. Of course, in a real system the activity changes occur less often and, therefore, the migration effort yields more benefit than in our artificial scenario where load shifts occur every 10 minutes.

As can be seen from Figure 18 the communication costs are not equally reduced at different zones. This is again caused by the differences in the speed of network connections. It can be seen that the continent in zone 2 has much faster network connections to the other continents than the continent in zone 1 which again has faster connections than continents in zone 3. As will be explained below, how much the costs are reduced also depends on how many areas are located on the continents of one zone.

In our measurements the time in which the "center of activity" stays at one zone is quite short, so the time the system benefits from migration is also short. In real applications the system's load will not move so fast, thus making the periods in which communication costs will be low much longer.

Figures 19 and 20 show the share of messages sent locally within a LAN and area, respectively. When the center of activity is in zone 1 the share of local messages achieved by the strategies is much lower than when zone 2 is active. The reason is that in zone 1 one continent with two areas is placed while in zone 2 there is one continent with only one area. In zone 1 the objects migrating to it distribute over the two local areas, so the share of local messages within a LAN/area is lower. Of course, the distribution of objects within the active zone is desirable. Also note that the communication costs were reduced by 56% when zone 1 was active. The same effect has been explained in Section 4.1.2.

Zone 3 comprises two continents with one area each. The shares of local messages sent locally within a LAN and area are higher than when zone 1 is active since objects wanting to migrate to a certain continent are not distributed between different areas. However, the shares are lower than during the active time of zone 2 since objects have the choice of two continents to migrate to.[8]

[8]Recall that for each continent there are four preferred airlines which will definitely migrate to this

When zone 2 is active the share of messages sent within the local area reaches more than 80%. This does not mean that 80% of the objects have migrated to the continent. In our experiment most likely the preferred airlines of the currently active continent and the flights they offer will migrate. Since the load at the active sites is very high also many of the other airlines will migrate to the continent but their flights will typically not get enough requests to make migration meaningful.

As in scenario 1, the differences in the decrease of communication costs between the strategies is not significant, thus the Good-LAN strategy is to be preferred because of its much lower calculation costs. Sometimes Good-LAN is even superior to Best-LAN because of a lucky decision.

5 Related Work

Many distributed systems provide facilities for process, task and object migration. Surveys of such systems are given in [Bor92, Nut94]. Process and task migration are mainly used for dynamic load balancing in a LAN [Nut94] and will not be considered further.

Most systems supporting object migration, e.g., Emerald [JLHB88] and DOWL [Ach93], provide migration primitives that can be employed, e.g., by a programmer, to achieve different goals. For example, migration can be used for load balancing, reducing communication costs, reconfiguration, garbage collection, moving parameter objects to the remote site during a remote invocation, etc., see also [JLHB88, Nut94].

Additionally to primitives enabling migration, Emerald and DOWL provide the possibility to *attach* objects, meaning that when an object migrates all objects attached to it will also migrate to the same destination. The attach migration primitive is transitive but not symmetric [JLHB88, Ach93, CKW96]. As pointed out in [CKW96], in non-monolithic applications, i.e., when independently developed components work together, attaching objects may cause problems. The reason is that an application induces attachments of objects without knowledge about attachments created on behalf of other applications. One effect is that migrating one object could be very expensive because many objects are attached to it but the object itself may not know it and might therefore underestimate the costs [CKW96].

The problems described in [CKW96] would also arise in a wide-area distributed system since applications are independent and without knowledge of each other. However, in our model an attach mechanism is not needed since objects working on a common task, i.e., requested by the same transactions, will automatically migrate to the site these requests originate. Attachments would be useful in case the cooperative tasks would be performed by the objects themselves through direct inter-object communication—not through transactions as it is the case in our model. Therefore, the presented migration concept does not provide for attachments, not even for a "restricted" version as proposed in [CKW96]. In our scenario, we therefore do not expect to encounter the "ping-pong" effect that one object moves to the other and vice versa.

The only other system we know of that deals with migration in a WAN distributed system is Mariposa [SAL+96, SDK+94]. In contrast to our system where objects can

continent.

autonomously decide to migrate, in Mariposa data are moved according to decisions made by sites they are located at. A microeconomic paradigm is used for this purpose. For each data fragment an access history is kept based on which a site can autonomously decide to buy (or sell, if it already has it) the fragment, depending on whether it would be profitable to have it or not. Mariposa's goal is not necessarily to minimize communication costs but query optimization.

Migration is also an issue in mobile agent systems. There are many different definitions of agents [FG97]. However, most definitions of mobile agents seem to have in common that they require agents to be able to migrate purposefully within computer networks during execution. Migration of agents ranges from code migration—as it can be realized through Java [AG96] and Telescript [Gen]—to the migration of the whole object as it is the case with aglets [LC96, Lan97]. For further information on mobile agents see, e.g., [RPZ97, VT97]. An overview and classification of mobile code, including mobile agents, is given in [FPV98].

The main difference between the migration of mobile agents and our autonomous objects is that agents have a task to accomplish which determines their movement (together with the agent's state). Agents may even have a travel map based on which they migrate through the system. In contrast, autonomous objects receive messages containing requests for method executions and migrate depending on where these requests came from.

Push-caching strategies, as presented in [GS94, Bes95, BC96], try to reduce communication costs in distributed systems, e.g., the WWW, by caching "popular documents" in the neighborhood of the clients accessing these sites. The caching is initiated by a server having the document, so the server has to determine the "optimal placement" for the document, which is also what autonomous objects do when considering migration. However, the criteria here are different from those for migration, since caching increases the number of copies of an item while migration moves the whole item to a new destination. For autonomous objects caching does not seem to be a promising approach, since the behavior of objects could hardly be cached. Also, autonomous objects will on the average change their state much more often than, for example, WWW documents are modified, thus drastically increasing the update overhead.

6 Conclusion

In this paper we have presented a migration protocol on the objects' level for a distributed system of autonomous objects designed for a WAN environment. The response time of distributed activities in such systems is clearly dominated by communication costs so our goal was to reduce these costs and thereby lower the response times. In WAN-distributed systems it is hardly possible to manage migration globally so a self-tuning system is needed. We gave two migration strategies for autonomous objects enabling them to autonomously decide whether and where to migrate thus avoiding the need for global management. The object migration cooperates with the distributed name server [EKK97] that ensures that migrated objects, which, may now be located in the neighborhood of a request, are efficiently located.

The first strategy, called Best-LAN, determines the optimal location for the object, relying on accesses in the recent past. The second strategy, Good-LAN, approximates the communication costs it uses for its calculations so it sometimes returns a sub-optimal location for an object. However, it induces significantly lower computational costs than the Best-LAN strategy.

We have also evaluated the strategies through benchmarks in a distributed system of autonomous objects, AutO. Additionally, we have compared the computation costs of the strategies and the quality of the decisions they make. The experiments have shown that the communication costs significantly decrease when migration is employed. The two strategies perform alike in most cases so the Good-LAN strategy should be preferred because of its lower computation costs.

Acknowledgments

We thank the other members of the AutO-Team: Stefan Grießer, Markus Keidl, Stefan Pröls and Stefan Seltzsam, for their implementational work on AutO. We also thank Stefan Seltzsam and Stefan Pröls for their help on running the benchmarks.

References

[Ach93] B. Achauer. The DOWL distributed object-oriented language. *Communications of the ACM*, 36(9):48–55, September 1993.

[AG96] K. Arnold and J. Gosling. *The Java Programming Language.* Addison-Wesley, Reading, MA, USA, 1996.

[BC96] A. Bestavros and C. Cunha. Server-initiated document dissemination for the WWW. *IEEE Data Engeneering Bulletin*, 19(31):3–11, September 1996.

[Bes95] A. Bestavros. Demand-based document dissemination to reduce traffic and balance load in distributed information systems. In *Proc. IEEE Symp. on Parallel and Distributed Processing*, San Antonio, TX, October 1995.

[Bor92] U. M. Borghoff. *Catalogue of Distributed File/Operating Systems.* Springer-Verlag, New York, Berlin, etc., 1992.

[CKW96] O. Ciupke, D. A. Kottmann, and H.-D. Walter. Object migration in non-monolithic distributed applications. In *Proc. of the 16th Intl. Conf. on Distributed Computing System*, pages 529–536, 1996.

[DLMM93] M. Day, B. Liskov, U. Maheshwari, and A. C. Myers. References to remote mobile objects in Thor. *ACM Letters on Programming Languages and Systems*, 2(1–4):115–126, March–December 1993.

[EKK97] A. Eickler, A. Kemper, and D. Kossmann. Finding data in the neighborhood. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 336–345, Athens, Greece, August 1997.

[FG97] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proc. of Intl. Workshop on Agent Theories, Architectures, and Languages*, pages 21–35, New York, Berlin, etc., 1997. `http://www.msci.memphis.edu:80/ franklin/AgentProg.html`.

[FPV98] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding code mobility. *IEEE Trans. Software Eng.*, 24(5):342–361, May 1998.

[Gen] General Magic, Inc. Mobile agents white paper. `http://www.generalmagic.com/technology/techwhitepaper.html`.

[GKKM93] C. A. Gerlhof, A. Kemper, C. Kilger, and G. Moerkotte. Partition-based clustering in object bases: From theory to practice. In *Proc. of the Intl. Conf. on Foundations of Data Organization and Algorithms (FODO)*, volume 730 of *Lecture Notes in Computer Science (LNCS)*, pages 301–316, Chicago, IL, October 1993. Springer-Verlag.

[GS94] J. Gwertzman and M. Seltzer. The case for geographical push-caching. Technical Report HU TR-34-94, Harvard University, Cambridge, MA, 1994.

[JLHB88] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-grained mobility in the Emerald system. *ACM Trans. Comp. Syst.*, 6(1):109–133, February 1988.

[KGI+97] N. Krivokapić, S. Grießer, M. Islinger, M. Keidl, S. Pröls, S. Seltzsam, and A. Kemper. AutO—A distributed system of autonomous objects. `http://www.db.fmi.uni-passau.de/projects/auto/auto.html`, 1997.

[KLMW94] A. Kemper, P. C. Lockemann, G. Moerkotte, and H.-D. Walter. Autonomous objects: A natural model for complex applications. *Journal of Intelligent Information Systems (JIIS)*, 3(2):133–150, 1994.

[Lan97] D. B. Lange. Java aglet application programmin interface (J-AAPI) white paper, Feb. 1997. `http://www.trl.ibm.com/aglets/JAAPI-whitepaper.htm`.

[LC96] D. B. Lange and D. T. Chang. Programming mobile agents in java, Sept. 1996. `http://www.trl.ibm.com/aglets/whitepaper.htm`.

[LM96] L. Liu and R. Meersman. The building blocks for specifying communication behavior of complex objects: An activity-driven approach. *ACM Trans. on Database Systems*, 21(2):157–207, June 1996.

[Nut94] M. Nuttall. A brief survey of systems providing process or object migration facilities. *Operating Systems Review*, 28(4):64–80, October 1994.

[RPZ97] K. Rothermel and R. Popescu-Zeletin, editors. *Proc. of the Intl. Workshop on Mobile Agents*, volume 1219 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, New York, Berlin, etc., 1997.

[SAL+96] M. Stonebraker, P. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A wide-area distributed database system. *The VLDB Journal*, 5(1):48–63, January 1996.

[SDK+94] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in Mariposa. In *Proc. of the IEEE Conf. on Parallel and Distributed Information Systems*, pages 58–67, September 1994.

[VT97] J. Vitek and C. Tschudin, editors. *Proc. of the Intl. Workshop on Mobile Object Systems*, volume 1222 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, New York, Berlin, etc., 1997.