A ROUTING STRATEGY FOR OBJECT-ORIENTED APPLICATIONS IN MASSIVELY PARALLEL ARCHITECTURES

ǂMaurelio Boari *, ǂAntonio Corradi *, Letizia Leonardi **, Cesare Stefanelli *

* Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna
2, Viale Risorgimento - 40136 Bologna - ITALY
Ph.: +39-51-6443001 - Fax: +39-51-6443073

** Dipartimento di Scienze dell'Ingegneria
Università di Modena
213, Via Campi - 41100 Modena - ITALY
Ph.: +39-59-367890 - Fax: +39-59-370040

E-mail: {mboari, acorradi, lleonardi, cstefanelli}@deis.unibo.it

Keywords: massively parallel systems, parallel objects, dynamicity, routing.

ABSTRACT

This paper describes an adaptive routing algorithm, called Virtual Path, designed to provide efficient communications for object-oriented applications running on massively parallel architectures. The main property of the algorithm is to route messages without any knowledge of the allocation of communicating entities (the objects) to the processors. Allocation transparency becomes a basic requirement when dealing with dynamic object-oriented applications where the decision of whether to create/destroy objects is taken at run-time and, in addition, objects can migrate from one node to another to achieve load balance. The presented routing algorithm is topology independent and can be employed with any topology of interconnection.

The Virtual Path routing algorithm alternates two different phases. In the first phase, the algorithm is completely adaptive and chooses one path to connect the nodes where the communicating objects reside. In the second phase, the algorithm uses the found path to route all messages exchanged by the same couple of objects. To obtain adaptivity, the routing system updates a virtual path whenever the situation changes significantly: for instance, in case of object migration.

This work has been partially supported by the "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" of the Italian National Research Council (grant No. 89.00012.69) and by the "MURST 40%" project.

ǂ IEEE Member.

1. INTRODUCTION

Interprocessor communication is potentially the component that most influences the performance of multiprocessors systems, especially in massively parallel systems typically not completely connected [1]. Interprocessor communication in direct networks where several links must be traversed to connect two different nodes involves several issues, such as data transmission mechanisms and routing requirements [2]. Most common interconnection topologies provide multiple physical paths for routing messages between two nodes: messages can either follow a path deterministically chosen or be routed adaptively [3]. Adaptive routing, in general, leads to improved performances, especially in the case of non uniform traffic conditions.

The goal of the routing algorithm presented in the paper, called Virtual Path (VP), is to provide efficient communications for applications composed of long lasting entities characterised by stable and repeated communication patterns: this means that there are entities that communicate preferably with a limited number of partners. Examples of applications where the use of the VP algorithm is advantageous are parallel object programming environments [4]. Parallel objects are allocated to the nodes of a parallel architecture and the routing system is in charge of the delivery of messages. Objects are long-lasting entities that require services by invoking each other's methods. In object-oriented applications, the cooperation between groups of objects lasts their whole life and the same couples of objects exchange messages in sequence.

Moreover, the allocation of communicating objects may change frequently. Objects may be either created or terminated dynamically and this decision is taken only at run-time; in addition, for the sake of load balancing, migration must be provided [5]. To face the dynamic behaviour of a parallel object environment, the chosen routing strategy has not to require the knowledge of the allocation of communicating entities to nodes (allocation transparency).

As a second requirement, the routing policy has not to be based on a predefined architecture but must be capable of working whichever network topology is used (transparency of network topology). This requirement excludes routing decisions based on topological assumptions, such as for instance, the ones used in hypercubes or in k-ary n-cube [1].

The above requirements of allocation and network topology transparencies are quite expensive to be obtained. The absence of global knowledge produces less informed and costly policies, that are nevertheless justified, or even imposed, in case of dynamic applications. The proposed solution is to use the VP routing that alternates two different phases. The first time a client object wants to communicate with a server object, the algorithm enters a first phase of routing and finds a path that connects the two objects. This path is obtained by sending a probe that is routed by using a completely adaptive strategy. In the second phase, the algorithm uses the found virtual path to route all messages exchanged by the same couple of objects. The routing system in its whole behaviour is still adaptive because the virtual path followed by messages is updated whenever the situation significantly changes: in particular, this occurs in the case of object migration.

The paper is organised as follows. Section 2 briefly describes the features of object-oriented environments, focusing on the requirements that a solution to the routing problem should satisfy. Section 3 presents the VP strategy. The performances of this routing algorithm compared with those of other adaptive strategies are reported in Section 4: the results refer to an implementation of the VP algorithm on a Transputer-based machine, a Meiko Computing Surface [6].

2. PARALLEL OBJECT ENVIRONMENTS

Several different models and implementations stem from the same abstract object paradigm. In particular, we consider parallel object environments in distributed and parallel architectures. Parallel objects are coarse-grained entities whose lifetime can be compared with the application one. Parallel objects can also be created (or destroyed) at run-time to satisfy the dynamic needs of applications.

In object-oriented systems, computation stems from inter-object communications. Any parallel object, when in need of an external service, must request it to the object that offers the corresponding operation. This establishes a client/server relationship between the objects. Different modes can be used for object communication. The synchronous mode engages the client object until the server object has yielded the reply. The asynchronous mode does not force the client object to wait: it simply sends the request to the server object and then it continues its execution. The future communication mode makes the client object to wait for a reply only when in need of it [7].

Independently of the mode of communication used, inter-object communication determines a dynamic scenario, where the client/server relationship is established at run-time and can involve any possible couple of objects. Parallel object applications, in general, follow stable and repeated patterns of communication: the same client/server couple of objects can be engaged in a stage of communication composed of a significant number of messages.

Any parallel object produces a load in the system with its memory and communication: a general principle of resource usage balance should be considered when implementing a parallel object support.
To balance the processing load, the mapping phase of an application on the target architecture requires the support to allocate only an average number of initial objects on each node. The intrinsic dynamicity of object applications introduces also the need of a run-time balance of resources: load balancing strategies decide the allocation of newly created objects and the reallocation of existing ones. The reallocation of objects is achieved by using migration mechanisms [8].
The need of balancing the communication resources requires routing to be adaptive in the sense of uniformly distributing the traffic of messages on the entire network. In addition, routing solutions based on the knowledge of object allocation can be too expensive in the case of a dynamic scenario.
Object migration interferes with the outstanding client/server connections. Two solutions are possible in the case of server object migration: the static one leaves a forward information in the node from which the object moves, the dynamic solution requires a requalification of all object references.
From the routing point of view, to leave a forward in a previous residence node implies that messages reaching that node are forwarded to the current location of the server object. In the dynamic solution, instead, the requalification can be made when the first message is exchanged after the migration.
In general, parallel object environments provide primitives for object creation, communication and migration, implemented by the run-time support where adaptive routing algorithms can be integrated. The VP routing has been already employed in the support layer of one of such parallel object environment, the **P**arallel **O**bjects programming **E**nvironment (**POE**) [9].

3. THE VIRTUAL PATH ROUTING SYSTEM

Object-oriented applications - composed of long lasting entities that cooperate and exchange data - may take advantage from the definition of **virtual paths**: a virtual path is one that can be followed by all messages travelling from the same source (client object) to the same destination (server object). However, there is no static decision involved: the virtual path is established when the first communication takes place and it is followed by all messages between the same couple of objects. The cost of defining the virtual path is shared among all the exchanged messages: the higher this number, the greater the advantage.

The Virtual Path routing algorithm conflicts neither with the independence from object allocation and from network topology nor with the adaptivity of the algorithm. In fact, virtual paths are created dynamically without any global knowledge of allocation or topology, and can be updated during execution to adapt to system state changes such as object migrations, link faults and varied traffic conditions. To react to one of these situations the router decides to find a new virtual path, thus initiating a different stage of communication between two objects (figure 1). A new stage begins with a first phase of routing, that ascertains the current system situation and defines a new path by using a new probe. The newly defined path become the virtual path and it is followed by all messages in the second phase of routing.

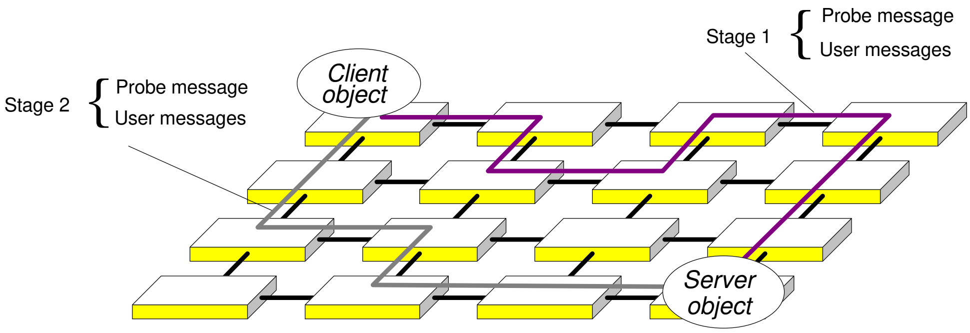

Figure 1. Stages in the communication between two objects. Each stage begins with the injection of a new probe.

3.1. Constructing the Virtual Path

The routing system defines a virtual path of interconnection between two objects by sending a *probe* message from the source to the destination node when the objects start to communicate. The probe records the traversed path from the sender to the receiver nodes. The probe is routed by using the Hot-Spot Avoiding (HSA) algorithm that answers to the needs of transparency of allocation and topology [10]. HSA has been inspired by the hot-potato routing, an algorithm that aims to keep balanced the length of the local output message queues [11]. While hot-potato is isolated because takes its routing decisions by considering only the situation local to each node, the HSA policy enlarges the scope of interest to the node neighbourhood. In the HSA routing, any node decides its routing strategy on the basis of knowledge of the load of its neighbour nodes. In this way, it succeeds in obviating overloaded situations. Even if the HSA routing can be considered livelock free, there is no guarantee that the path followed by the probe does not contain loops: when the receiving site gets the probe, the router eliminates loops from the recorded path (figure 2) to produce the virtual path used in the following phase. Let us note that the resulting path established between the communicating objects is not necessarily the minimal one.

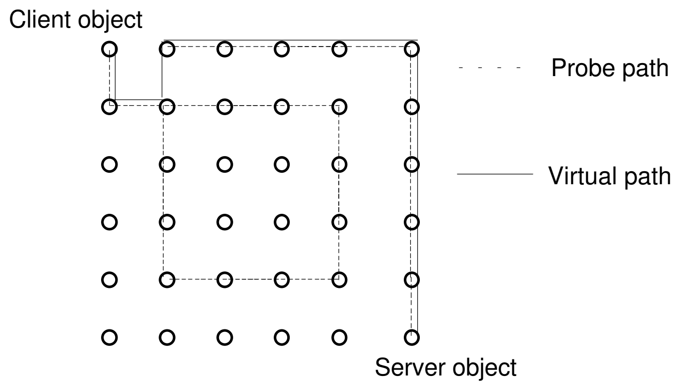

Figure 2. The definition of a virtual path in a 6x6 mesh.

The probe with the found virtual path is transmitted back to the sender router following the resulting path backwards (figure 3.a).
After sending the probe, the router at the sender site waits for a response from the receiving site within a time-out limit: if the probe does not come back within a predefined interval, a new probe is injected into the network. Once the virtual path is defined, successive probes returning to the sender node will be discarded. *Multiple probe* can be sent to reduce the probability of a time-out occurrence. The receiver router sets up one virtual path for each received probe, and transmits them back to the sender router. It is responsibility of the sender router to choose among the received paths. A multiple probe solution has the advantage of furnishing a set of paths from which to choose the virtual one, but presents the disadvantage of increasing the routing system overhead.

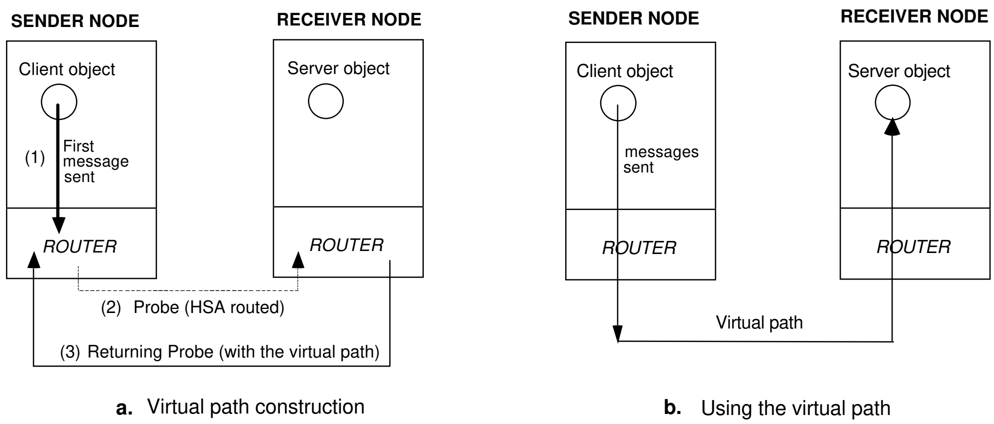

Figure 3. The two phases of the VP algorithm.

3.2. Using the Virtual Path

After the virtual path construction phase, messages between the same couple of application objects follow the established virtual path (figure 3.b): loops are no longer possible and livelock is ruled out.

In this phase, the sender router inserts in every message the indication of the path to follow: the routing algorithm is static and has no computation overhead. In fact, there is no routing policy to compute: in any intermediate node, the router only identifies the output link of the virtual path carried by the message (figure 4).

The message sent by the client object leaves the sender node with the entire virtual path and a message hop counter. The sender (and any other intermediate) router forwards the message to the router found in the virtual path with the current message hop.

In the case of synchronous and future communication modes, the reply sent by the receiver (server) object follows the same virtual path of the message but in the reverse order.

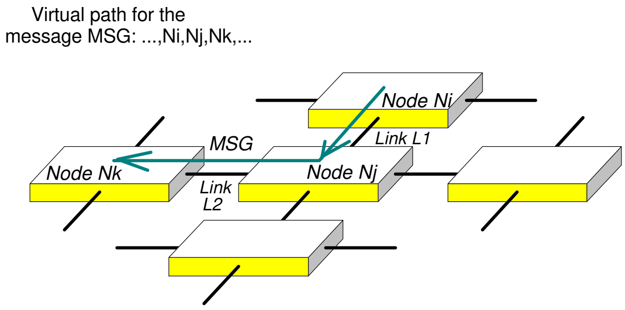

Figure 4. The N_j router directly puts the message MSG into the L2 output queue.

The VP algorithm is adaptive and it is able to react to system evolution: in fact, it takes into account exceptional conditions, such as:

- *Fault*. It is related to either a hardware failure (a node failure or a faulty link) or a software failure (for instance, a request to an object no longer present in the system).
- *Migration*. In dynamic load-balanced systems, a receiver object can be no longer found because it has moved to another node during execution.
- *Path congestion*. There are cases when areas of the system become congested and messages can experience delay in delivery time.

To react to one of the above situations, the router establishes a new virtual path, thus initiating a different stage of communication between two objects. The communication mode between objects determines when to start new stages.

In the case of synchronous communications an event-driven approach can be followed. A new stage is started in response to the events that can be detected by a time-out. The time-

out triggers the change of the virtual path: a new probe is sent and all new messages follow the new virtual path. The identification of the correct time-out is derived by the probe timing. A good estimate is proportional to the delivery time of the related probe.
When objects use the asynchronous communication mode, the event-driven approach is not possible. In this case only a time-driven approach is feasible. On the basis of the application dynamicity, the router initiates a different stage of communication by finding new virtual paths at predefined fixed intervals.
The case of object migration is a particular event that can be faced by different approaches. In *client migration*, the client object has to modify all the virtual paths it uses by resending new probes. The case of *server migration* is more complex because the server does not know the objects in need of its services and there is no way of notifying its migration. Section 2 briefly reports two possible solutions to this problem, a static and a dynamic one. The static solution leaves a forward in the node from which the server object moves. Messages reaching that node are directly forwarded to the current location of the server object (figure 5.a). In the dynamic case, the router at the client site looks for a new virtual path when the client object tries to communicate with the migrated server (figure 5.b): that avoids an excess of forwarding entities of the static solution. That approach is followed in the next section.
Let us note that this discussion has been simplified by assuming that an object is only a client (or a server). This, of course, is not the real case: an object can be a client of some objects and a server for others.

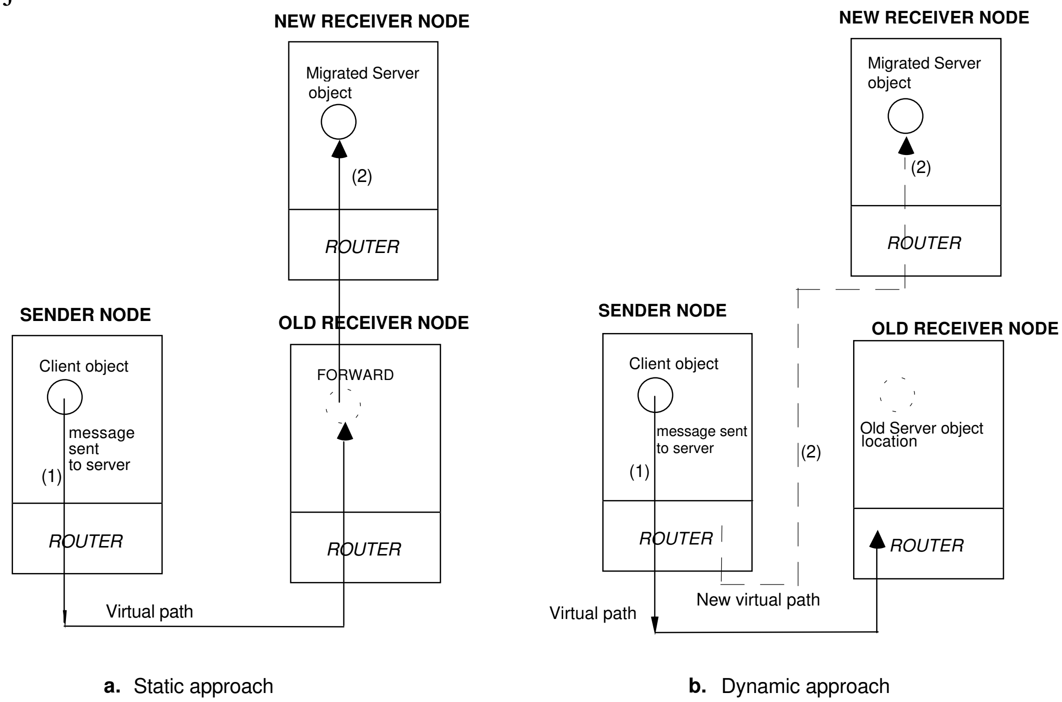

Figure 5. Migration approaches.

3.3. The Routing System Support

This paragraph describes the framework properties of the routing algorithm. In order to occupy the system resources in the path toward the destination, the implemented routing system adopts a **packet-switching** communication mode, that establishes no direct connection between any source/destination couple [12]. The allocation of buffering areas to messages in the nodes is controlled with a **store-and-forward** flow control technique: the entire message is buffered at each intermediate node and forwarded to the next node in its path when the desired outgoing link becomes available. The buffering area for a message is dynamically allocated from the node heap only by need. No memory is statically reserved for buffering. The store-and-forward is the natural mechanism adopted when working on a transputer architecture, because of its link implementation.

The VP routing support adopts a method of deadlock prevention that satisfies the transparency assumptions our routing policy is based on. In store-and-forward routing, the critical resources for deadlock are the buffers used to store messages in intermediate nodes. When a node exhausts its buffers, it enters a saturation phase. A deadlock situation occurs when two or more saturated nodes are awaiting each other circularly incapable to deliver messages. In order to prevent this situation, we have adopted a strategy similar to the one reported in [13], to permit delivery of messages in the case of saturation.

The deadlock prevention strategy requires each node to reserve a limited buffering capacity: these special buffers offer the possibility of exchanging messages between any two saturated neighbor nodes. In this way, nodes in saturation still exchange messages that ultimately reach their destinations. In saturation, message throughput lowers but can increase as soon as the situation becomes less congested.

The most important feature of this deadlock prevention method is low intrusion: the prevention technique has no cost during normal execution and impacts only in saturation, thus introducing a negligible overhead. In other words, the above prevention technique executes only when the system can no longer guarantee any acceptable response time.

Another important property for routing is absence of livelock. In our case, the allocation transparency forces the routing system to misroute frequently the probe when looking for the receiver: in the first phase of VP, misrouting can be performed at each hop. Since the sender router waits for the probe to come back with the virtual path, if a probe has incurred in a livelock situation it does not cause any problem, apart from the performance penalty of resending a new probe. In the second phase of routing, messages follow the virtual path and no livelock can occur.

4. EXPERIMENTAL RESULTS

In this section, we discuss the performance of the VP router integrated in the parallel object environment POE implemented on a Meiko Computing Surface, a MIMD parallel architecture based on the transputer technology [14]. The target Meiko Computing Surface is composed by a set of interconnected transputers: our current configuration is composed of 16 transputers and one SUN-Sparc Node. The Meiko programming environment (CStools) gives two different communication tools: transports with embedded static routing strategy and channels with a semantic similar to that of occam channels (and then without embedded routing policy). Since the transport routing policy is static, it does not fit the dynamic requirements of parallel objects applications. To overcome this limitation, we have designed and implemented the VP routing strategy by using the channel mechanism.

All tests are reported for mesh topologies, the typical transputer interconnection: in any case, our router does not use any network topology information.

The testbed application used to evaluate the VP router is composed of several objects that exchange messages of lengths in the range 100-500 bytes:

- one *Client object* synchronously sends *test* messages on the network to one *Server object*;
- *Traffic generators* inject in the network messages representing the communication load of applications.

All tests reported in the following allocate the *Client object* and the *Server object* in nodes at the maximal distance in the network (this is the worst possible case, in a general nxn mesh it is 2*(n-1)), and one *Traffic generator* in any other node (figure 6).

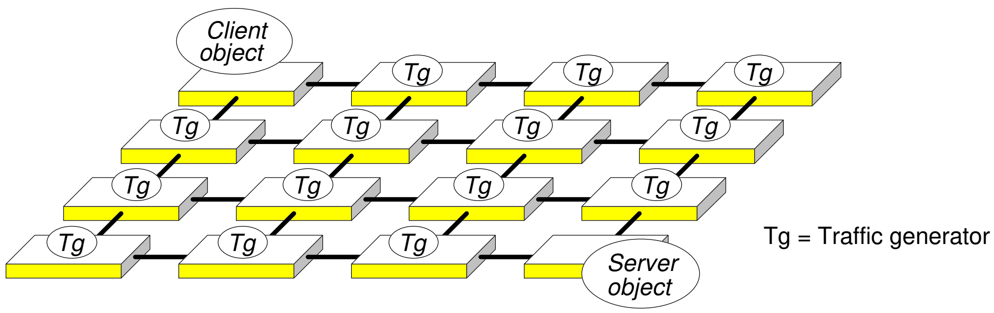

Figure 6. The testbed application in a 4x4 transputer mesh.

Each time the *Client* and the *Server objects* enter a new stage of communication, the VP router determines the virtual path connecting them: it defines the virtual path by sending several probes and choosing the path with the minimal number of hops among the obtained ones. All others messages exchanged during the current stage of communication between the *Client* and the *Server objects* follow that virtual path. The choice of the number of sent probes is a trade-off between the cost of the multiple probe solution and the number of messages exchanged during each stage: the higher the number of messages using the same virtual path, the higher the number of probes one can advantageously send. Let us note that there is no need to wait for a response to all sent probes: one can choose the virtual path only within a subset of them. Figure 7 shows the average communication latency for different number of messages exchanged during a stage of communication, with different number of sent and waited for probes in the testbed application. The reported average communication latency considers the cost of establishing the virtual path (by sending a number of probes) and the cost of all communications performed in the stage. In fact, the former cost is shared among all messages exchanged.

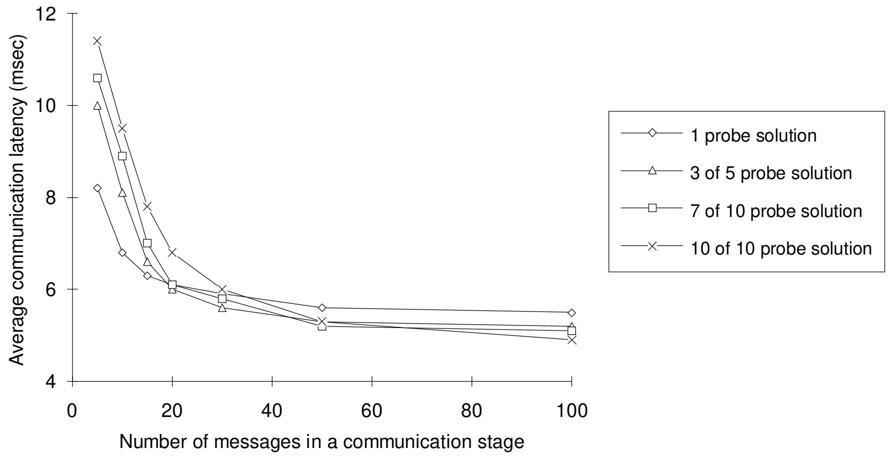

Figure 7. The performances of different multiple pr0obe solutions in a 4x4 mesh.

The average communication latency of messages sent to the *Server object* in figure 7 greatly varies with system dynamicity. In particular, the two main variables are the communication frequency between objects and the occurrences of exceptional situations.

The influence of the communication frequency between objects is reported in figure 8, with the average communication latency obtained by varying the frequency of messages synchronously sent by the *Client object* to the *Server object*. In this test, *traffic generators* inject different number of messages per second, to generate different traffic volumes in the network. Figure 8 shows that, for each traffic volume, a value of the *Client/Server* communication frequency exists that represents a performance threshold: above it the virtual path becomes congested.

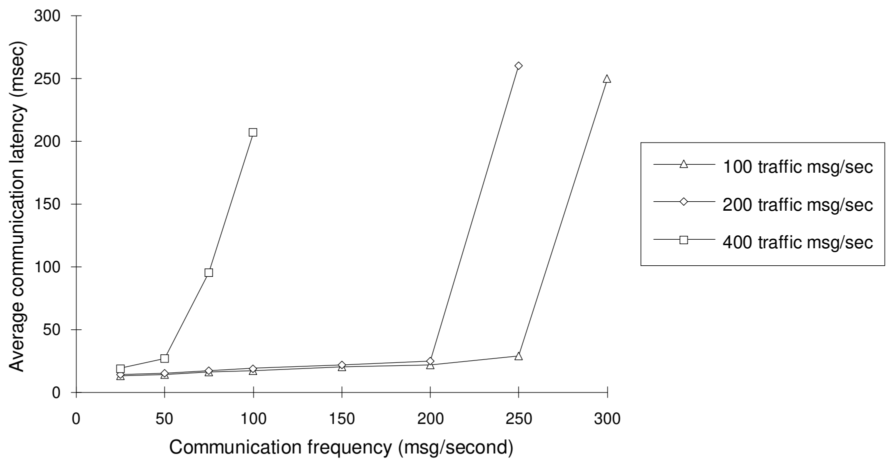

Figure 8. The performance of the VP algorithm with the increase of the communication frequency. The curves are obtained for different traffic volumes in a 8x8 mesh.

Up to now, test objects communicate by using the same path; the possibility that exceptional situations such as fault, migration or path congestion could force the objects to enter a new stage of communication is not considered. Since the testbed application uses synchronous communications, all exceptional situations can be revealed by a time-out event, that can trigger a new probe phase to find a new virtual path, thus initiating a new stage in the communication between objects.

This happens, for instance, in the case of object migration, the most interesting case to be considered in parallel objects environments where load balancing policies are employed. Figure 9 considers the case of the migration of the *Server object*. The test represents the average communication latency for messages sent by the *Client object* when the *Server object* migrates after the receipt of a certain number of messages. Two algorithms are compared, the hot-potato routing and the VP one. With the fully adaptive hot-potato routing, the average communication latency does not vary because of the *Server object* migration: each message searches the destination node independently of the allocation/reallocation of the *Server object*. In the VP case, when the *Client object* gets to know of a *Server* migration, it initiates a new stage of communication by finding a new virtual path. In other words, the first message arriving at the destination node after migration does not find the *Server* and reports the migration to the *Client* router that is forced to construct a new virtual path for the couple *Client-Server*. In the figure, the cost of the virtual path, i.e., the initial cost for establishing it and the cost to redefine it, is divided among all the messages. The VP approach outperforms the hot-potato as soon as each stage of communication contains even a limited number of messages.

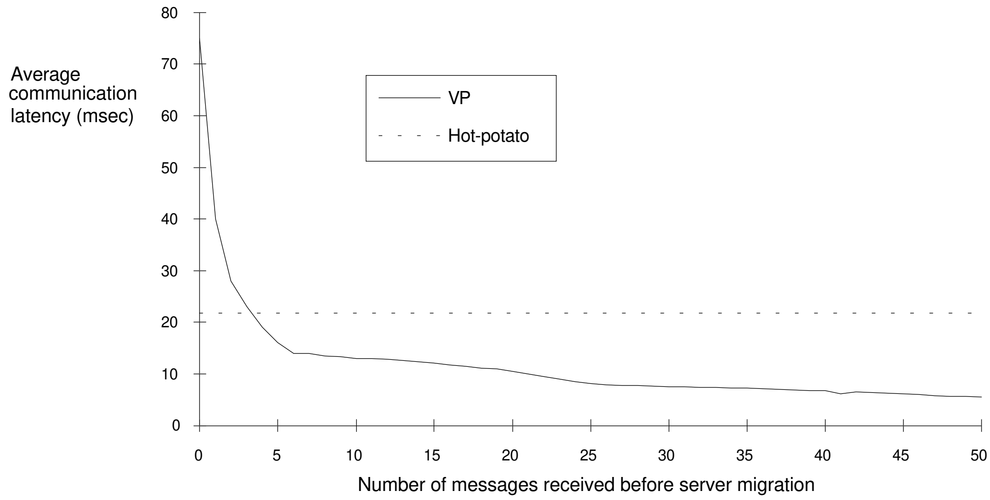

Figure 9. Comparison of hot-potato and VP algorithms for communication stages of different lengths in a 4x4 mesh.

The last set of tests deal with the system's computational load. To compare fully adaptive algorithms, such as hot-potato, and VP in the case of balanced and unbalanced computational load, several compute-bound objects are allocated to nodes. Figure 10 reports balanced conditions, figure 11 unbalanced situations. In both cases, the VP routing shows better performances than those of the hot-potato one. This can be explained with the pure locality of hot-potato routing that considers only the current node local situation. The situation in a neighbourhood of nodes is instead taken into account by the VP algorithm to define the virtual path. The resulting policy is more informed and can achieve efficiency and performance.

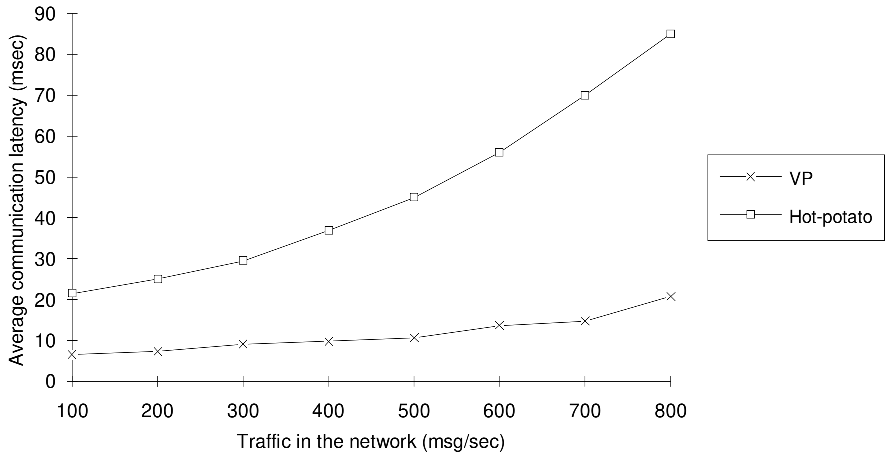

Figure 10. Comparison of hot-potato and VP routing, with balanced load conditions in a 4x4 mesh.

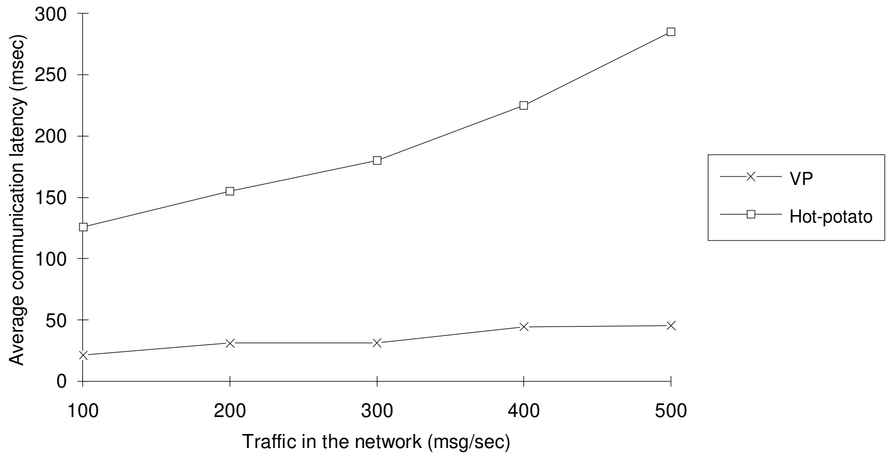

Figure 11. Comparison of hot-potato and VP algorithms, with unbalanced load conditions in a 4x4 mesh.

As a last comment, the VP routing compares well with other routing systems implemented on transputer-based architectures. The performances of those systems, in terms of mean internode latency [15], are in the range 450-1200 μs depending on the traffic volume for the Multiple Rings routing [16] and about 500 μs for Tiny [17]. VP routing system presents mean internode latency within the same range (450-900 μs).

CONCLUSION

The paper proposes an adaptive routing strategy suitable for applications composed of long lasting entities that present stable and repeated communication patterns: in these cases, in fact, the cost of determining the virtual path the first time is counterbalanced by its use in tight communications between the same entities.

The Virtual Path algorithm does not use any information about entities allocation and network topology, and has shown its best performances in dynamic parallel object-oriented applications. The Virtual Path is a succession of stages of communication between two communicating parties: the stage is the adaptivity step and it is terminated when the current resources situation degrades the message routing in a intolerable way. In this case, a new stage can be initiate, to consider the modified resources conditions.

The VP routing solution is a good trade-off between adaptivity and efficiency and can be usefully employed where the system dynamicity forces to consider not static solutions to the routing problem.

REFERENCES

1. W. J. Dally "Performance Analysis of k-ary n-cube Interconnection Networks", *IEEE Transactions on Computers*, Vol. 39, No. 6, June 1990.
2. A. Varna, C. S. Raghavendra (editors), *Interconnection Networks for Multiprocessors and Multicomputers Theory and Practice*, IEEE Computer Society Press, 1994.
3. P. Gaughan, S. Yalamanchili, "Adaptive Routing Protocols for Hypercube Interconnection Networks", *IEEE Computer*, Vol. 26, No. 5, May 1993.
4. P. Wegner, "Concepts and Paradigms of Object Oriented Programming", *ACM OOPS Messenger*, Vol. 1, No. 1, August 1990.
5. A. Hac, "A Distributed Algorithm for Performance Improvement through file replication, file migration, and Process migration", *IEEE Transactions on Software Engineering*, Vol. 15, No. 11, Nov. 1989.
6. Meiko, Computing Surface Reference Manual, Meiko Ltd. 1989.
7. A. Yonezawa, et al., "Modelling and Programming in an Object-Oriented Concurrent Language ABCL/1", in *Object-Oriented Concurrent Programming*, eds. A. Yonezawa and M. Tokoro, MIT Press, 1987.
8. G. T. Almes et alii, "The EDEN System: a Technical Review", *IEEE Transactions on Software Engineering*, v.SE-11, No. 1, January 1985.
9. A. Corradi, L. Leonardi: "Parallelism in Object-Oriented Programming Languages", IEEE Int. Conference on Computer Languages, New Orleans, March 1990.
10. M. Boari, et alii, "Hot-Spot-Avoiding: an Adaptive Strategy for Transputer Routing", Transputers'94, Besancon, Sept. 1994.
11. P. Baran, "On Distributed Communications Networks", *IEEE Transactions on Communications Systems*, Vol. 12, No. 1, March 1964.
12. A. S. Tanenbaum, *Computer Networks*, Prentice-Hall, 1988.
13. J. Blazewicz, J. Brzezinski, G. Gambosi, "Time-Stamp Approach to Store-and-Forward Deadlock Prevention", *IEEE Transactions on Communications*, Vol. 35, No. 5, May 1987.
14. INMOS, Transputer Reference Manual, Prentice Hall, 1988.
15. D. Talia, "Message-Routing Systems for Transputer-Based Multicomputer", *IEEE Micro,* Vol. 13, No. 3, June 1993.
16. M. Cannataro, G. Spezzano, D. Talia, E. Gallizzi, "Design, Implementation and Evaluation of a Deadlock-Free Routing Algorithm for Concurrent Computers", *Concurrency: Practice and Experience*, Vol. 4, No. 2, April 1992.
17. L. Clarke, G. Wilson, "Tiny: an efficient routing harness for the Inmos transputer", *Concurrency: Practice and Experience*, Vol. 3, No. 3, June 1991.