Efficient Monitoring to Detect Wireless Channel Failures for MPI Programs*

Elsa M. Macías and Alvaro Suárez
Grupo de Arquitectura y Concurrencia
Departamento de Ingeniería Telemática
University of Las Palmas de Gran Canaria, Spain
{emacias,asuarez}@dit.ulpgc.es

Vaidy Sunderam
Dept. of Math and Computer Science
Emory University, Atlanta, USA
vss@mathcs.emory.edu

Abstract

In the last few years the use of wireless technology has increased by leaps and bounds and as a result powerful portable computers with wireless cards are viable nodes in parallel distributed computing. In this scenario it is natural to consider the possibility of frequent failures in the wireless channel. In MPI programs, such wireless network behavior is reflected as communication failure. Although the MPI standard does not handle failures, there are some projects that address this issue. To the best of our knowledge there is no previous work that presents a practical solution for fault-handling in MPI programs that run on wireless environments. In this paper we present a mechanism at the application level, that combined with wireless network monitoring software detects these failures and warns MPI applications to enable them to take appropriate action.

1. Introduction

Traditionally, parallel computing using a network of computers has been used to speed up the execution time of sequential programs. For doing that, a network of heterogenous computers and middleware is used to facilitate the programming of message-passing applications. This middleware is typically built on top of UDP/TCP/IP and operates on clusters, local networks, campus networks, and even wide area networks. Two of the most important middlewares are PVM [9] and MPI [12]. The interest in the wireless communications (primarily based on IEEE 802.11b) to do parallel and distributed computing, and the usage of mobile devices as real-time data acquisition systems or interfaces to access metacomputing systems has been increasing during recent years [7]. In a previous work [6] we have shown the feasibility of wireless communications for doing parallel computing combining networks that follow the IEEE 802.3 and IEEE 802.11 standards. We employ an infrastructure network connected to a local area network and the Internet via an access point. We consider Master-Slave parallel applications in which there is a main loop and for every iteration the processes must synchronize to exchange data (the input data subsets are derived from a previous computation), and applications with trivial parallelism in which there are no data dependencies among iterations.

An important issue when working with this type of networking is letting that parallel and distributed applications can continue working gracefully in presence of spurious disconnections and communication failures. In these situations, TCP does not behave well and besides there also is no an accepted way for a link level driver to inform TCP of packet loss and frequent disconnections [2]. Although TCP could inform MPI about congestion and spurious disconnections, this could flood the wireless channel with a lot of irrelevant information because not all the computers in the coverage area may be executing MPI program parts. To avoid this situation, our LAMGAC middleware [3] maintains a registry of wireless computers that collaborate with

* Research partially supported by Spanish CICYT under Contract TIC01-0956-C04-03 and Las Palmas de G.C. University UNI2002/17; and by U.S. DoE grant DE-FG02-02ER25537 and NSF grant ACI-0220183.

Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (EUROMICRO-PDP'04)
1066-6192/04 $20.00 © 2004 **IEEE**

the MPI program and uses a simple mechanism to manage some failures. Although there are works that deal with faults in MPI programs, for example the one of reference [11], they have not been implemented in wireless environments to the best of our knowledge. A work that deals with failures in wireless computing systems is presented in [8] but a practical solution for MPI programs has not been implemented.

In this paper we present an improved mechanism to detect failures in the wireless channel and to warn MPI parallel applications about network degradation so they can take corrective action. The mechanism uses the collected data by our monitoring software [10] that detects and predicts impending degraded network quality.

The rest of the paper is organized as follows. In section 2 the proposed mechanism is reviewed in comparison to previous work and its performance is analyzed. In section 3 the modifications made to the wireless tool to integrate it with MPI parallel applications are presented. Section 4 is devoted to discussing some experimental results. Finally, we sum up the conclusions and present our ongoing work.

2. Reviewing the fault detection mechanism

In [5], a mechanism was proposed to prevent the abrupt ending of MPI parallel applications when a a wireless connection used for MPI communications stays down too long during a blocking receive or a synchronous blocking send. To prevent this situation, it is necessary to use non blocking operations or asynchronous blocking sends. This sole technique is not sufficient because it does not inform the application if a communication channel exists to a slave process and therefore the master does not know the number of live slave processes to distribute data or to receive results from. For that reason, in [5] the `LAMGAC_Fault_detection` function was presented that checks the wireless channel from the master to each slave process. Briefly, the programmer specifies the ranks of the slave processes to be tested (running on wireless computers), and internally the function executes a concurrent version of the standard ICMP

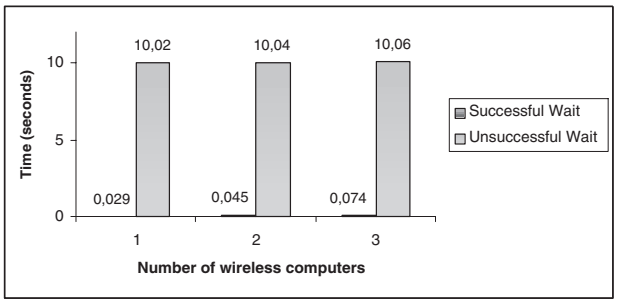


Figure 1. Response times in absence and presence of failures

(Internet Control Message Protocol) [1] echo request and reply packets to/from the wireless computers corresponding to those ranks. The function returns an array of integers with the ranks of faulty processes (those without communication from the computer where the master is running- we refer it as the *master node*).

Because of the programming model we use [4], it is more efficient and easier to implement the wireless channel detection mechanism from the master. Although we are aware of a fault in the master will crash the whole system, notice that this is also true for any MPI parallel program due to the communicators become invalid.

2.1. Studying the performance of the mechanism

We stated in [5] that the proposed mechanism is simple, portable and we truly thought it would not introduce a high overhead. For showing that, we tested the mechanism sending ICMP frames in parallel from the master node to each one of the portable computers and waiting for the echo replies. The experimental results show an overhead of ten seconds on average when the wireless channel is broken (with a low standard deviation in all the experiments) and a duration less than one hundred milliseconds when the wireless channel is good (again with a low deviation). These results are presented in Fig. 1 (values labelled as `Unsuccessful Wait` and `Successful Wait` re-

spectively). The tests were made varying the number of wireless computers to be tested from 1 to 3 with the network interface cards disabled to test the mechanism in presence of failures, and from 1 to 2 laptops with 11 Mbps wireless cards switched on and associated to an access point at 11 Mbps and one personal computer with a wireless interface at 2 Mbps and connected wirelessly to the master node in ad hoc configuration. All the wireless cards complies with IEEE 802.11 standard. It is obvious that this implementation is not suitable to be used for parallel applications because in case of failures, the overhead is considerable.

An initial improvement could be such that `LAMGAC_Fault_detection` may specify the maximum number of seconds to wait for an ICMP echo reply. The new implementation we proposed is based on combining the concurrent application with a timeout, that is to say, the thread that sends an ICMP echo request returns either when the echo reply is received or when the timeout expires. A design parameter is to determine the value for this timeout. Ideally, it should be equal or higher than the average time to send and receive one ICMP request and reply frame, that is to say, the round trip time (hereafter `rtt`). The `rtt` between two computers is variable because of the overhead imposed by the external network load. This should be taken into account to determine a more suitable value for the timeout. To determine this value for each master and slave pair, there should be information about the most recent round trip times. These values should be tested periodically by wireless network monitoring software. The collected information should be stored in a secure place and accessible to the master process.

3. Integration of LAMGAC with the network monitoring software

In [10] we present a lightweight monitoring software to detect when a wireless node is entering or departing a region with a degraded network quality (termed as `trouble spot`). Briefly, the software measures round trip times, signal and noise strength and signal quality. With these measurements, the applications or protocols can take preemptive or corrective action during trouble spots. For example, for our purposes, the master may defer the reception of results computed by slave processes if the nature of the application allows it, or on the contrary it would detach the faulty slave from the computational duties.

When the tool is started on a computer, it spawns three threads:

- `Injector`- constructs ICMP echo request packets and sends them to the access point with which it is currently associated.
- `Capturer`- monitors incoming traffic, captures and processes the ICMP echo reply packets and computes the round trip time.
- `Signal data processor`- retrieves signal and noise strength by querying the device driver.

We have made some modifications to the monitoring software to integrate it into a computing environment. Briefly, data collected individually for each wireless device is stored in a simple database with fast access from the master process. The monitoring software is running on each wireless device to reduce the computational load on the master node.

Fig.2 presents the threads and processes that take part in the monitoring and storage phases, and also the database structure:

- `Injector` (Fig.2.a)- sends N (configurable) ICMP echo requests to the master node via the access point. Then it notifies `Filter` and it is suspended.
- `Signal data processor` (Fig.2.a)- tests N samples (also configurable) about the signal and noise strength (ss and ns values in Fig.2.c) and signal/noise relationship (sq). After that, it notifies the `Filter` thread and it is suspended.
- `Capturer` (Fig.2.a)- the same behavior as the original `Capturer`.
- `Filter` (Fig.2.a)- this new thread averages out rtt, ss, ns and sq values. It sends these values to the

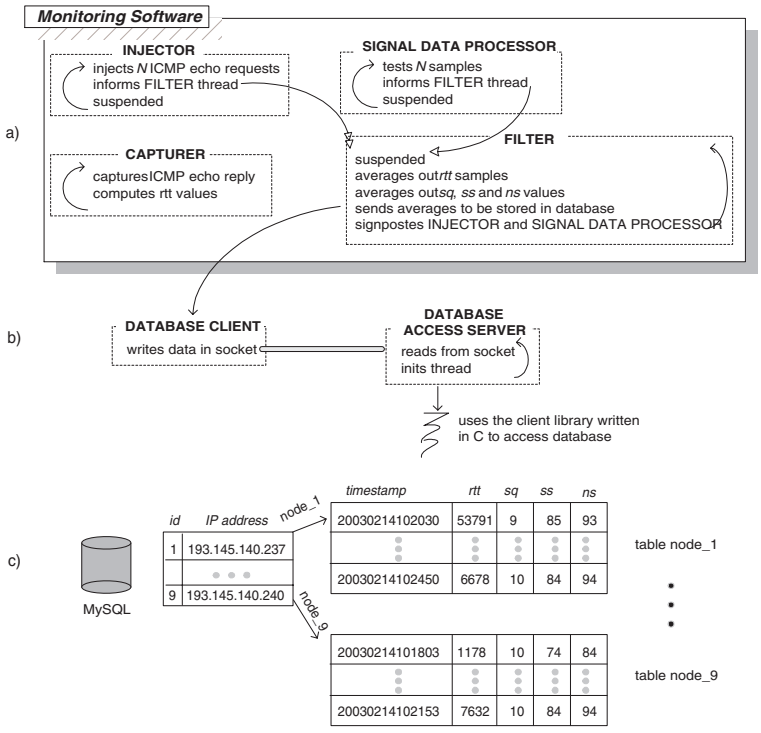


Figure 2. Software parts. Monitoring software (a), database client and database access server processes (b), and database structure (c)

`db_access_server` process (invokes `db_client` process) to update the database. Then, it awakes `Injector` and `Signal data processor` and it is suspended.

- `Database client` (Fig.2.b)- sends the values calculated by the `Filter` to the `db_access_server` process and the portable node's IP address via sockets.
- `Database access server` (Fig.2.b)- listens the values sent by the `db_client`, it throws a thread that stores them in the table corresponding to the portable computer. There is one instance of this process running in the master node.
- `Database` (Fig.2.c)- there is one table per each portable computer which stores (per columns) a *timestamp*: date and time of the updating, and *rtt*, *sq*, *ss* and *ns* values (integers). There is one table that relates the IP address to its corresponding table through the id field.

Figure 3 shows the main ideas about the behavior of the modified monitoring software: after computing the averages, the `Filter` sends values to the `db_access_server` invoking `db_client` process (steps 1 and 2 in Fig.3). The `db_access_server` inserts these values in the table that stores monitored values for the portable computer (step 3).

When the master calls `LAMGAC_Fault_detection`, for example to test the slave with rank equal to x (step 1 in Fig.4) running on portable computer with IP $193.145.140.237$, it is selected from database the table for this node (step 2) and from this table is selected the maximum value of `rtt` (step 3). This integer will be the timeout.

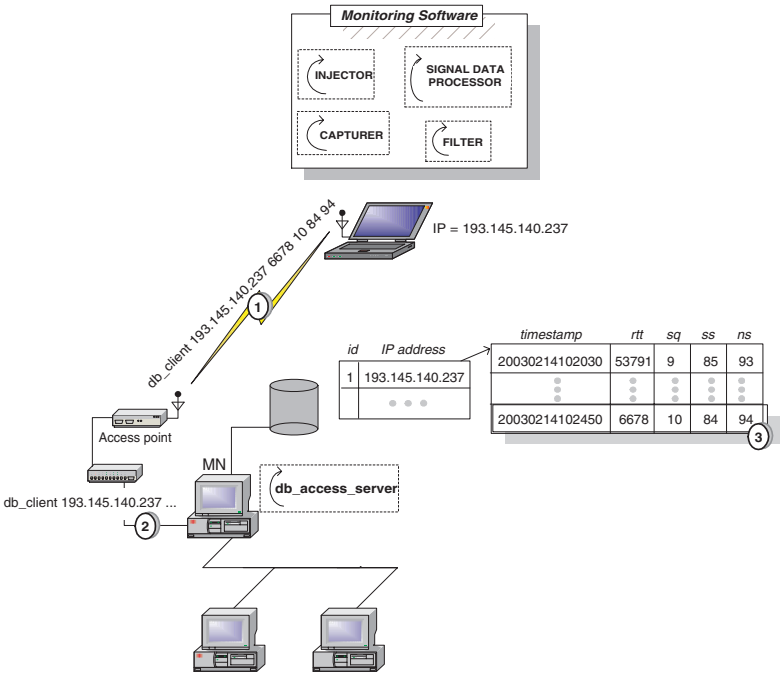


Figure 3. Monitoring software operational overview

Within `LAMGAC_Fault_detection` an ICMP echo request is sent (steps 4 and 5) and the timeout or the reply packet will determine if there is or not communication between the master and slave process.

4. Experimental results

We first evaluate the `rtt` for each wireless computer presented in section 2.1 to the master node and collected by the monitoring software. The measurements were made in three different periods (each one of 10 minutes of duration) and the graphic in Fig.5 shows the average measurements for each minute (the tool injects 10 packets per second). Table 1 presents the `rtt` values in microseconds.

The execution times of our fault detection mechanism compared with the two implementations (with and without timeout) for situations in which the wireless channel is broken are presented in Fig.6.a. Similar experimental results are shown in Fig.6.b when the wireless channel is good. In both cases, the timeout-based implementation sets the timeout to the sum of the maximum `rtt` values for the computers involved in the tests, and for both implementations we include in the measurements the time to create the threads that perform the injection and capture of standard ICMP packets. As you can see, in Fig.6.a the implementation without timeout introduces a high overhead compared with the solution with timeout (in the latter, the timeout expires before a possible reply reaches).

However, the execution time for the implementation with timeout in absence of failures is higher, although negligible, than the implementation without timeout due to the overhead of creating the thread that monitors the timeout and the extra synchronization between this thread and the ones in charge of doing the injection and capture, and the parent process.

An important issue for our fault-detection mechanism is the rate of false alarms, that is, the percentage of time the mechanism believes that there is no connection to a wire-

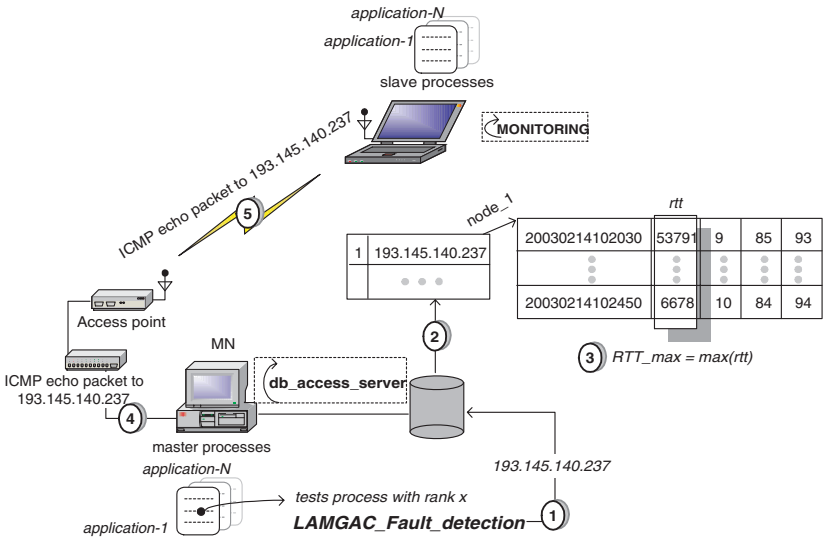


Figure 4. LAMGAC_Fault_detection operational overview

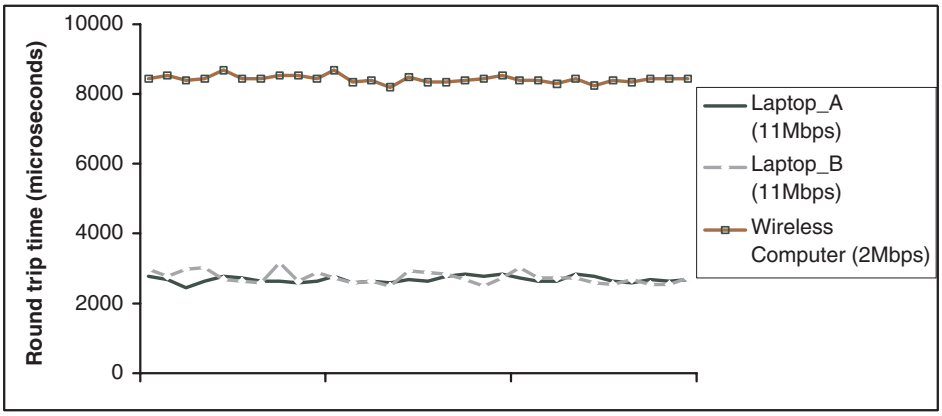


Figure 5. Round trip times from each wireless computer to the master node

less computer and in fact there is (this percentage should be near 0%). Although the percentage of time that the mechanism determined correctly that there was or not connection to a wireless computer was equal to 100%, we keep in mind studying the probability that a fault be signalled in absence of failures.

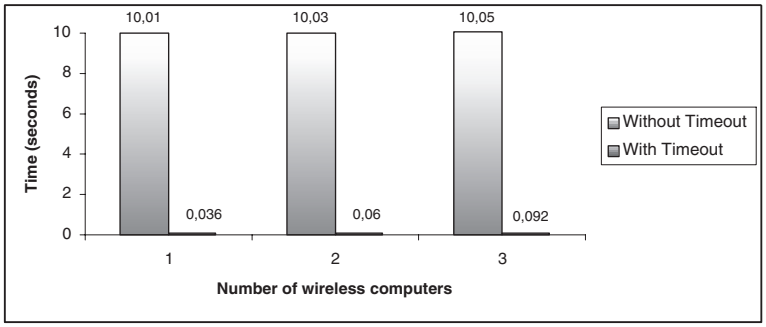
5. Conclusions and future work

In this paper we have presented an improved mechanism that detects the state of the wireless channel and conveys it to the MPI program to take corrective action. To reduce the overhead that our initial proposal can impose to the parallel program, we combine the testing function with the col-

	LAPTOP A	LAPTOP B	WIRELESS COMPUTER
Minimum	2456	2507	8198
Maximum	2825	3191	8670
Average	2688,5	2742,6	8427,2

Table 1. Round trip time values in microseconds

a)



b)

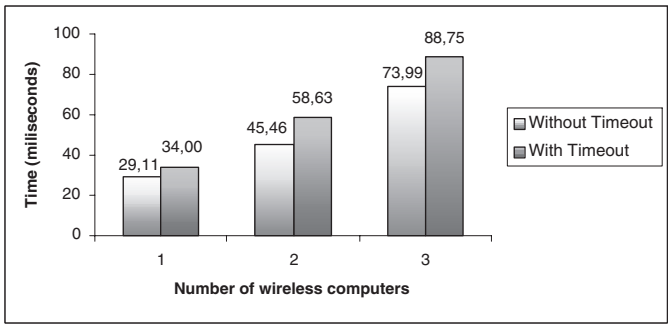


Figure 6. Average execution times of the mechanism over 20 and 50 runs in presence (a) and absence (b) of failures

lected data by our WLAN monitoring tool.

Although the monitoring software is lightweight and the degree of accuracy when approaching trouble spots is high, we have intention of testing the mechanism with a large number of nodes to study its scalability and the percentage of false predictions. An efficient mathematical characterization of these predictions in terms of probability is not possible. To do that, we plan to use a network simulator because of experimenting with a high number of computers in a WLAN is not a trivial task. It is expected that the monitoring software adapts itself to the traffic on the network and sets the appropriate timeout. In this situation the combination of all the metrics stored in the database will gain a greater degree of confidence in the wireless channel state detection.

References

[1] D. E. Comer. *Internetworking With TCP/IP: Principles Protocols, and Architecture*, volume 1. Prentice Hall, Fourth edition, 2000. Chapter 9.

[2] G. Huston. TCP in a wireless world. *IEEE Internet Computing*, 5(2):82–84, March/April 2001.

[3] E. M. Macías, A. Suárez, C. N. Ojeda-Guerra, and E. Robayna. Programming parallel applications with LAMGAC in a LAN-WLAN environment. In Y. Cotronis and J. Dongarra, editors, 8^{th} *European PVM/MPI Users Group Meeting*, volume 2131, pages 158–165, Santorini, Greece, September 2001. LNCS, Springer Verlag.

[4] E. M. Macías, A. Suárez, and C. N. Ojeda-Guerra. Solving non-smooth unconstrained optimization problem with LAMGAC in a LAN-WLAN grid domain. In 10^{th} *Euromicro Workshop on Parallel Distributed and Network-Based Processing*, pages 471–478, Las Palmas de Gran Canaria, Spain, January 2002.

[5] E. M. Macías and A. Suárez. A mechanism to detect wireless network failures for MPI programs. In P. Kacsuk, D. Kranzlmuller, Z. Németh, and J. Volkert, editors, *Distributed and Parallel Systems. Cluster and Grid Computing*, SECS 706, pages 211–218. Kluwer Academic Publishers, 2002.

[6] E. M. Macías and A. Suárez. Solving engineering applications with LAMGAC over MPI-2. In D. Kranzlmuller, P. Kacsuk, J. Dongarra, and J. Volkert, editors, 9^{th} *European PVM/MPI Users Group Meeting*, volume 2474, pages 130–137, Linz, Austria, September 2002. LNCS, Springer Verlag.

[7] M. Migliardi. Mobile interfaces to metacomputing and collaboration systems. In *Advanced Topic Workshop Middleware for Mobile Computing*, Heidelberg, 2001.

[8] Y. Morita and H. Higaki. Checkpoint-recovery for mobile computing systems. In *International Conference on Distributed Computing Systems*, pages 479–484, Phoenix, 2001.

[9] V. S. Sunderam. PVM: a framework for parallel distributed computing. *Concurrency, Practice and Experience*, 2(4):315–340, 1990.

[10] G. Tonev, V. Sunderam, R. Loader, and J. Pascoe. Location and network issues in local area wireless networks. In *International Conference on Architecture of Computing Systems: Trends in Network and Pervasive Computing*, Karlsruhe, Germany, April 2002.

[11] V. Zandy and B. Miller. Reliable network connections. In 8^{th} *Annual International Conference on Mobile Computing and Networking*, pages 95–106, Atlanta, USA, 2002.

[12] Message Passing Interface (MPI) Forum Home Page, www.mpi-forum.org.