Fault Tolerant Aspects of the PVM Based Virtual Time Machine PV^{2}M

Vittoria Gianuzzi

Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova, Viale Benedetto XV, 3
16132 Genova Italy
Email gianuzzi@matgen.dima.unige.it

1 Introduction

The main goal in distributed simulation is speedup over traditional sequential simulation. In order to obtain significant results, a simulation must generate a sufficient number of typical system evolutions, thus requiring CPU time which may easily extend into days.

Several papers describing the distributed simulation of various problems have been published in the last years. Some of them refer to specific programming environments, developed for given architectures, mainly parallel systems with shared memory, the best known of them being the Time Warp Operating System (TWOS) [Fuji89], a special purpose virtual time-based operating system working on Mark III Hypercube and BBN Butterfly Plus^{TM}. All these environments try to meet the same objective, that is to free the user from the burden of dealing with the complexity of the parallel system. These environments are highly effective but not easily portable.

Recently, there has been much interest in using inexpensive, powerful workstation networks, possibly with high speed interconnection, instead of expensive parallel computers. Then, some general-purpose programming environments which provide the users with a rich toolset, easily portable on different architectures, such as PVM, Express or Linda, have been implemented. Distributed simulation can benefit from the possibility of using such network-based computing environments, in order to achieve portability.

With respect to special-purpose parallel computers, distributed systems exhibit different features, such as asynchronous computation, heterogeneity, and increased probability of communication and processor failures. These latter are due not only to hardware faults, but also to the fact that these systems fall under different administrative domains, with different policies of intervention. Failures are usually only partial, and failure of a node does not affect the whole computation, if correctly handled. Parallel computer systems lack of supports for both failure detection and recovery, because failures rarely occur, on the other hand, fault tolerance provisions are absolutely needed in distributed systems, for long-running applications, such as distributed simulation.

Taking into account these considerations, a distributed environment for the implementation of processes based on the Virtual Time concept [Jeff85] is under development on top of PVM [Geis93]. Due to the two virtualities the system has been called PV^2M [Gian95].

2 The Virtual Time synchronization mechanism

Virtual Time is a general method to simulate synchronous systems on asynchronous ones. Each process maintains a local (virtual) clock, and sends time-stamped messages. Virtual Time processes operate optimistically, i.e. they assume that messages always arrive in the correct temporal order. If causality is violated (i.e. a message has been processed before another message with smaller time stamp), a roll-back operation is performed: the local clock is adjusted to a previous value, the memory is restored as it was at that virtual time, and antimessages are send, if needed, to annihilate messages sent out during the deleted time interval. Input messages are then reconsidered, in their correct order.

Logs of input messages, output traces and memory states are needed in order to allow the roll-back operation. Uncontrolled diffusion of rollbak and unlimited extension of logs are avoided because the whole execution can be periodically committed, until a so called Global Virtual Time (GVT). GVT is defined, at any given real time, as the minimum value among all local clocks and the time stamps of the messages in transit. Execution progress is guaranteed since GVT can be proven to be an increasing function.

Virtual time synchronization can be used in various fields of distributed computation. In particular, the principles of synchronization via virtual time gives rise to the Time Warp simulation mechanism [Fuji89].

Time Warp also differs from another widely applied distributed simulation mechanism, that is the pessimistic (or conservative) approach, in that it allows the execution of a dynamically changing number of logical processes, with arbitrary communication relationships. From this point of view, PVM perfecly matches the need to face the dynamic creation and destruction of processes.

3 The PV^{2}M system

The PV^2M system has been designed in order to support optimistic distributed simulation on top of PVM programming environment, however, since it simply implements the virtual time mechanism, it can be used for any other distributed computation compatible with such a mechanism.

The system provides the user with the code of an *agent* process, that resides on each host making up the PV^2M, and a library of functions implementing the virtual time synchronization mechanism, which provides the C interface between a user defined process and PVM. The PV^2M functions are modeled as a subset of the PVM set: few of them have some different parameters, and others, exceedingly bound to the low level programming, have been eliminated in order to maintain a higher abstraction. The user defined process, together with the PV^2M function, constitutes the so-called virtual time process (*VTproc*). PV^2M has been designed in close resemblance to PVM, to allow the user to adopt a suitable and friendly programming style which is gaining even increasing favour.

The only concern of the user is to provide:
⋆ the maintenance of the local clock, an integer variable which must be updated in order to be always greater than the time stamps of every input and output messages. It represent the simulated time of the process.
⋆ the two procedures $save()$ and $restore()$ in order to allow PV^2M to save and restore the memory state.

User calls requiring receipt and sending of messages, as well as spawning and destruction of processes, are filtered by PV^2M wich provides the dispatch of them, according to the virtual time synchronization rules. In addition to these kinds of messages, called regular, other messages, the control ones, circulate among processes and agents, following the scheme given in Figure 1.

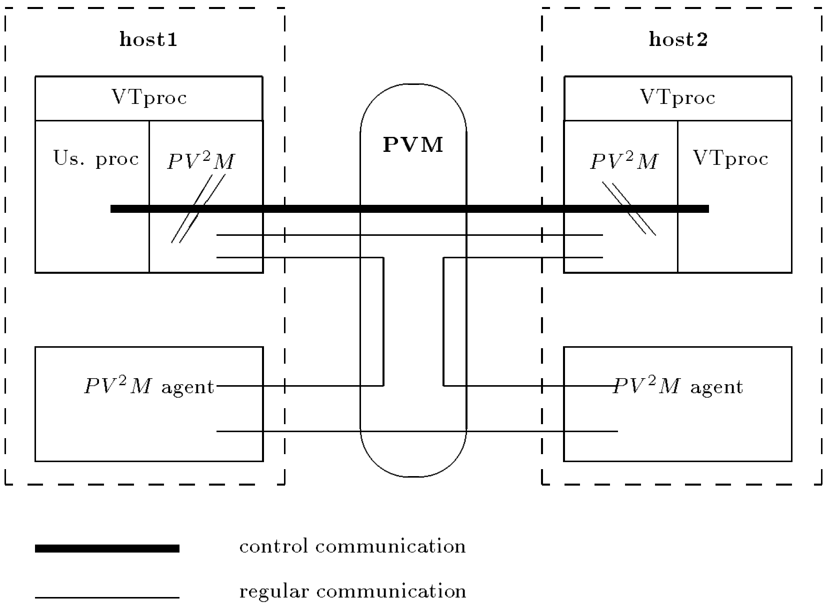

Figure 1. Interprocess communication

All the virtual time processes residing on the same host are connected to the local *agent* which participate in process migration and failure handling algorithms. As well as the master pvmd, also the *master agent* acts as coordinator of all other *agent*s, and provides the user with an interface, similar to the PVM console, to monitor the execution and to modify the run time environment. All the *agent*s are then connected with the *master agent* (see Figure 2), which is also responsible to detect host failures and to start the control distributed algorithms.

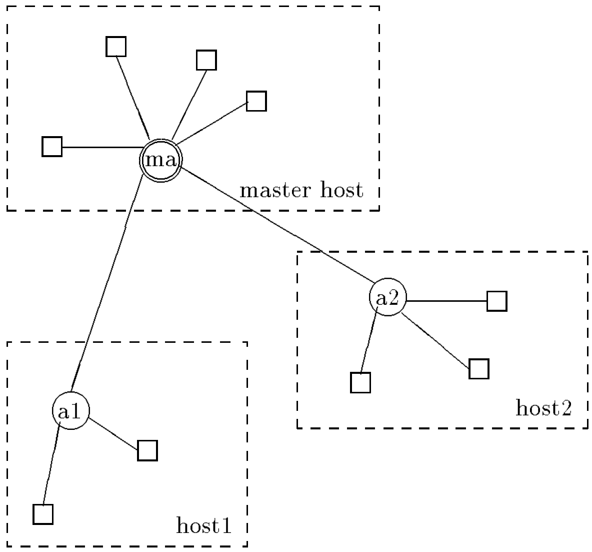

Figure 2: PV^2M architectural model

4 Influence of the architecture on host failure handling

The features of the PVM architecture that must be envisaged with respect to fault handling are mainly four.

Dynamic configuration. The PVM architecture can change run time, then, more hosts can be added as they become available, or be used as spare hosts to re-integrate failed ones.

Failure notification. Since PVM automatically detects the failure of a virtual host and deletes it from the virtual machine, if not the main host, any PVM process can ask to be notified about this event, in order to recover the killed tasks.

Centralized organization. The master pvmd cannot be recovered: it must be a member of the PVM for the complete duration of the virtual machine. Then, from this point of view, PVM has a centralized organization.

Fault tolerance can be achieved by means of redundancy. Usually, the redundancy management protocol of a fault tolerant system which removes failed nodes and re-integrates repaired ones, is a distributed protocol, to avoid a single point of failure. But, because of the central role played by the master demon in PVM, a distributed redundancy management algorithm, able to handle the failure of any host, is unfeasible, and a centralized protocol has to be considered.

PVM task identifier. The task identifier given to a process enrolled in PVM has encoded in it, for sake of communication performance, the code of the host on which it is located. Thus, process migration, which is requested to resume failed host processes on different hosts, cannot be performed in a transparent way, with respect to the user.

Dynamic group functions. PVM provides supports for dynamic process group implementation. Groups are very useful in order to implement active process replication. Since the identifier of a process belonging to a group is independent from the host on which it is located, groups can also be used for implementing process migration.

5 Host failure detection and recovery in PV^{2}M

Two approaches to the fault tolerance has been considered for PV^2M, and will be presented in the subsections below. The first one has been selected for the present implementation. However, the second mechanism will be also implemented, to allow a comparison of the relative merits.

The time-warp mechanism is space consuming, but, from the fault tolerance point of view, the recording of states and messages is a valid support for the local chekpointing mechanism. Moreover, due to the virtual synchronicity of Time Warp model, no additional synchronizations or freezing of processes are needed for the global chekpoint.

5.1 Back-up copies

This is the simplest fault tolerance mechanism for PV^2M, which only require the maintenance of a global checkpoint of the VTprocs. The host failures are considered in a non-homogeneous way, because of the peculiar model of PVM.

Non Master Host failure: a logical sequence of hosts, headed by the main host, is maintained. Every host, excepts the last one, periodically records in its main memory the memory states of its local processes and the status of the processes on the host which follows in the sequence. All checkpoints are kept at the same virtual time, without freezing the underlying execution.

If a slave host fails, the master host is notified (by the PVM environment) and the distributed execution of the user program is automatically restarted from the last taken checkpoint. In such a case, the backup host of the failed one starts up new processes in place of the lost ones. A better load balancing can be achieved by applying a general migration policy. The only problem is the lack of support for migration of PVM that requires each VT process be given a proper identifier, different from the PVM tid, then, a correspondence table bounding the two values is maintained in each VT process.

Master Host failure: every agent periodically records a copy of the status of the processes residing on its host, kept at the same virtual time, on a stable storage. Then, execution can be restarted on the same hosts, after master host repairing.

Only the master host has not a backup host: if it fails, the PVM dies and the system can be restarted using the stable storage copies.

Only fail stop failures are considered, and the system is 1-resilient with respect to the non master host failure, that is, after a host failure, no other host can fail before its recovery.

5.2 Passive replication

Active and passive replication both require that each process be replicated with a given each one receiving the same messages. Messages must be sent by broadcast and be processed by the replicas in the same order. Non virtual time synchronization mechanisms allows the existence of there could exist more than one consistent linearization of the execution history. In this case, additional protocols must be added to implement causal broadcast, so that all the copies of a process can determine their stable input messages [Schn90].

Some distributed systems exist which support programming with process replications. Probably, the best known of them is PARALEX [Baba92] which has been built on top of the ISIS toolkit [Birm89], and uses its coordinator-cohort library to implement passive replication. However, PARALEX cannot be used for implementing TW distributed simulation, because its model only allows the definition of processes without persistent internal status and a statically defined communication acyclic graph. On the other side, due to the optimistic point of view of the virtual time model, CBCAST protocols are not needed in order to implement consistency among replicas.

In order to obtain passive replication in Time Warp it is sufficient that all the components of the group of replicas receive the same messages, not necessarily in the same order, providing that they all participate in the definition of the global virtual time. If the number of replicas in each group is statically defined, PVM provides with its process group functions all that is needed to implement passive replication in PV^2M. Moreover, the global virtual time evaluation algorithm can be implemented so that to allows the detection of lost messages. These features should allow the use of the UDP protocol, also for local communications, without the need of implementing the PVM code to make transmission more reliable.

This last observation can then lead to a simplification in PVM message passing, but it requires a modification of PVM routines itself. Passive replication implemented by means of the PVM group library seems to be a promising method, since the PVM group management allows an easier implementation of migration mechanisms. [1]

References

[Baba92] O. Babaoglu et al., "Paralex: an environment for parallel programming in distributed systems", *Proc. 6th ACM Int. Conf. on Supercomputing*, pp.178-187, 1992.

[Birm89] K. Birman, K. Marzullo, "ISIS and the META Project", *SUN Technology*, 2(3), pp.90-104, 1989.

[Fuji89] R.M. Fujimoto, "Time Warp on a shared memory multiprocessor", *Proc. Int. Conf. Parallel Processing*, pp.242-249, 1989.

[Geis93] A. Geist et al., "PVM 3 User's Guide and Reference Manual", Techn. Rep ORNL/TM-12187, 1994.

[Jeff85] D.R. Jefferson, "Virtual time", *ACM Trans. Programming Languages Syst.*, 7(3), pp.404-425, 1985.

[Jeff87] D.R. Jefferson et al., "Distributed simulation and the Time Warp Operating System", *Proc.* 12^{th} *SIGOPS*, pp.77-93, 1987.

[Gian95] V. Gianuzzi, F. Merani, "Using PVM to implement a distributed dependable simulation system", *Proc. Euromicro Worksh. on Parallel and Distributed Processing*, Sanremo, Jan. 1995.

[Schn90] F.B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial", *ACM Comp. Surveys*, 22(4), pp.299-320, 1990.

[1]This work has been supported in part by the *Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo* of the Italian National Research Council.