Fault Tolerant PVM Application? — A Case Study with D-CICADA(FT)*

(Extended abstract)

Luděk Matyska and Karel Taft
Department of Information Technology
Faculty of Informatics, Masaryk University
Burešova 20, 602 00 Brno, Czech Republic
e-mail: {ludek,ktaft}@informatics.muni.cz

September 14, 1994

Motivation

The ever increasing number of computers connected into network leads to the development of new programming techniques to allow full use of the potential of these *virtual computers*. Despite problems of programming difficulty, associated with the distributed computing environment, the problem of *fault tolerance* is also emerging. The distributed computing environment is often used as a low-cost substitute of fast (super)computers. However, the very nature of these virtual computers, created from dozens of ordinary workstations in local or metropolitan area networks, is not really suitable for long time tasks. Congestion in the network, software and hardware crashes, broken connections etc are all the reasons while the efficient use of the virtual computer is further complicated. To ease its use, modified and/or new tools and techniques must be provided, both at the level of distributed computing environment and at the level of programming methodology.

In this article, D-CICADA(FT), a robust implementation of the D-CICADA system [1] is presented. It is shown how careful design may lead to robust (and almost fault tolerant) system.

D-CICADA: master — slave programming model

D-CICADA is the distributed implementation of the method for analysis of potential energy hypersurfaces (PES) of molecules. It searches for the low energy conformations and paths between them. It uses driver approach, rotating all predefined torsions in all directions. D-CICADA uses a notion of *driving path* as a sequence of driving steps leading from a minimum w of PES to another minimum w', $w \neq w'$. Minimum on a driving path is defined as a nuclear configuration x such that both the nuclear configurations x_{-1} (i.e., the configuration on driving step *before* the x) and x_{+1} (i.e., the configuration one step *after* the x) have higher energies then x.

*This work has been supported by the Grant Agency of the Czech Republic, grants no 201/94/1493 and 203/94/0522.

The main purpose of the CICADA approach is to find all the low energy conformations and all the saddle points between them (ie, all minima and maxima on driving paths).

D-CICADA [1] is a PVM based implementation of the CICADA approach. It uses the inherent parallelism of the method, where each driving path is served by a slave process, while the master being responsible for the task allocation (choosing the paths to be driven) and result collection — construction of the tree which describes the topology of the PES analyzed. The vertices of this tree are individual conformations, either minima on PES or saddle points (maxima) between them. The result of the D-CICADA run is the whole tree, containing all the minima on PES with energy lower than some predefined value. The implementation has very good efficiency (the computation/communication ratio is very high for individual slaves) and scalability, but is almost unusable in unstable local area network environment. A failure of any individual PVM node leads to the final failure of the whole computation and, more importantly, to the inefficient use of resources — there was no possibility to restart the `pvmd` daemon and many nodes remained idle for long period of time (especially during weekends) before they were restarted manually.

This experience lead to the development of more robust version, D-CICADA(FT), which is described in the following section.

Fault tolerancy enhancement

There are three general weak points in the master-slave implementation model:

1. The PVM system itself, where a crash of the main `pvmd` daemon inevitably leads to the crash of the whole virtual machine.

2. The master looses connection with some of its slaves. This includes the crash of the slave program or the whole machine where it is running as well as the break in the communication line.

3. The slave looses connection with the master. This, again, may be caused by the crash of the master program or its node, or by the communication breakdown (this is, however, not exactly the other side of the master's loose of the connection to the slave).

There is no simple solution for the first case. A substantial redesign of the whole PVM system is needed to overcome this limitation, but this is completely out of the scope of this article. There are, however, some actions that can enhance the robustness of the virtual machine, namely:

- running the `pvmd` daemon on the most robust node of the whole virtual machine, with the most robust operating system;

- using the `pvmd` node just for simple tasks, where the possibility of break down is negligible;

- running a simple "super" master program on the same machine. This supermaster is responsible for the integrity of the application program in the same sense as the `pvmd` is responsible for the integrity of the PVM itself.

This approach was used in the implementation of the D-CICADA(FT) program.

The application program must be robust with respect to the other two weak points, and this robustness must be a result of a careful program design. PVM keeps track of the individual nodes and automatically deletes any failing node, ie any node with lost connection. The slave `pvmd` with lost connection shuts itself down. This means that whenever the master `pvmd` marks any node as lost, we can safely assume that all local PVM tasks aborted, too.

In the D-CICADA(FT) implementation, the master keeps individual timers for all slaves with assigned driving path. Whenever the time limit exceeds a predefined value (default is 5 times the longest already encountered interval), D-CICADA(FT) master sends a request for information about the particular node. This request is served by the D-CICADA(FT) supermaster, running on the master PVM machine. There are two possible answers for this request:

1. The node is already marked down. In this case the master deletes the slave from its internal tables and marks the driving path as not yet tried. It also resets all the information connected with this driving path, in particular, it removes the information of reached maximum on the path (if the driving path was already in the descending phase).

2. The node is alive, just the response time is slower. This is often the case when a particular node of the virtual machine becomes overloaded by other tasks. If there are idle nodes in the PVM, the master sends a *retargeting* signal to the slow slave and assigns a new slave to take its responsibility. The slow slave opens a connection with the marked node and transfers all the information needed to continue in the driving path. Simultaneously, the master modifies its internal tables and associates a mark *slow* with the slow node. This is used in the scheduling part of the master to assign new task to this node only when there is no other idle processing node[1].

We use a receive with timeout call (`pvm_trecv`) in the main loop so the master is able to react even in situation where all the slaves aborted (like when there is just the last slave running).

The most difficult situation to handle is the failure of the D-CICADA(FT) master. This means that the master `pvmd` daemon loses connection with the node where the D-CICADA(FT) master runs. This situation has to be discovered by the slave who did not receive in-time acknowledgement. Being this the case, the slave notifies the supermaster (this is possible due to the slave split — while one thread[2] waits for the message from the master, the other thread performs the molecular modeling calculation and at its end is able to react on the time-out. The supermaster inspect the state of the virtual machine and either

- discovers that the master is just overloaded — in this case it did nothing in the hope that the load on the master's node will eventually diminish;

- discovers a failure of the node. In this case the supermaster selects another node and restarts the master there, sending him the information of the whole virtual machine and the D-CICADA(FT) state (ie `tids` of all slaves). The new master then notifies all the slaves about this event and asks for the complex information from each slave in order to re-establish internal tables, lost during the old master crash.

[1] This task retargeting is also used to speed up the whole D-CICADA(FT) computation in the end of the run. It was observed that the longest driving paths were assigned to the slowest machines and this random (but unfortunate) assignment often slowed down the whole computation (the 10 % slowdown was not unusual). Nodes with the *slow* mark can be periodically checked to inspect their actual load; this feature is not yet implemented.

[2] We use this notion to emphasize the mutual dependence of the two processes the slave is splitted to. These *threads* are, however, implemented as ordinary UNIX processes.

The original slave implementation had to be rewritten in order to enhance the robustness. Namely, the slave has to keep information about the whole computed driving path to be able to restore master's internal tables. Slave must be also able to react on the change in the master (via unspecified `pvm_recv` call).

The master performs some checkpointing, too. Individual vertices of the tree constructed by D-CICADA(FT) are stored as they are found and their coordinates computed. We decided to use the NFS shared directory to store the vertices, and the supermaster "knows" nodes with the access to this shared directory. Whenever new master D-CICADA(FT) restarts, it inspects this directory and rebuilds the tree from the information stored in individual files. Only the information about initiated but not yet finished driving paths is not directly available and it must be restored with the help of individual slaves. New master sends a broadcast request to all slaves and expects to receive complete description of "open" driving paths. It also checks the list of slaves sent by the supermaster to detect any lost slaves.

The supermaster is also responsible for restart of lost nodes. It periodically inspects the current state of the PVM and tries to restart individual `pvmd` daemons on disconnected nodes, using a specialized process (direct slave of the supermaster) on some selected node (this try is never performed on the master node where the master `pvmd` daemon and the D-CICADA(FT) supermaster run). This node restarting can be also used for modification of the whole PVM (there are usually more nodes available for application during nights and weekends than during work hours). This ability of supermaster to change the whole virtual machine, together with the retargeting feature fastened significantly the whole computations.

The D-CICADA(FT) is implemented in C over the PVM version 3.

Conclusion

One of the weak points of the virtual computers, whose nodes are single workstations connected via local or metropolitan area networks, is the relative high rate of breakdowns of connection and/or individual computers. The PVM system, although providing an excellent running environment for distributed applications, was not developed with fault tolerancy in mind (especially due to the original development focus [2]). However, the robustness (if not true fault tolerancy) of any distributed program with respect to the faults in a virtual machine (either in individual nodes or in the communication lines) is a crucial aspect of the usability of the concept of virtual computers as low-cost substitutes for larger (super)computers.

We believe that the decision taken in D-CICADA(FT) and presented in this article can be used in redesigning other PVM-based programs using the master-slave model to improve their robustness. We also believe that the trend towards more robust distributed applications is an important part of the whole distributed computing effort. The robust (if not truly fault tolerant) implementation increased significantly the efficient use of the virtual machine used.

References

[1] Matyska L., Koča J.: *D-CICADA — a software for conformational PES elucidation on network of workstations*, J. Comp. Chem., **15**(7), 1994.

[2] V.S. Sunderam, *Concurrency: Practice and Experience*, **2**, 315 (1990).