Resource Management and Checkpointing for PVM

Georg Stellner
Institut für Informatik der Technischen Universität München
Lehrstuhl für Rechnertechnik und Rechnerorganisation
D-80290 München
stellner@informatik.tu-muenchen.de

Jim Pruyne
Department of Computer Sciences
University of Wisconsin – Madison
pruyne@cs.wisc.edu

Abstract

Checkpoints cannot only be used to increase fault tolerance, but also to migrate processes. The migration is particularly useful in workstation environments where machines become dynamically available and unavailable. We introduce the CoCheck environment which not only allows the creation of checkpoints, but also provides process migration. The creation of checkpoints of PVM applications is explained and we show how this service can be used in a resource manager.

1 Introduction

Researchers from many different areas have needs for computational power to solve their specific problems. Today, many are successfully using PVM[8] on Networks of Workstations (NOW) [1] to satisfy their requirements. A PVM application uses the aggregate computational power of several workstations to speed-up the computation of a single problem. PVM has been used to parallelize a great variety of applications like genetic sequence analysis, semiconductor simulation or air quality modeling.

Usually, in a NOW a single workstation is assigned to a single *primary user (owner)*: the workstation can typically be found on this user's desk. Other users (*secondary users (guests)*) may access this machine via the network. Psychologically, the owner of the workstation expects that his workstation is at his disposal all the time. Whenever he wants to work interactively, he expects quick and immediate responses of his machine.

Running a PVM application on a NOW requires executing processes on many machines as a secondary user. Hence, a PVM application borrows resources such as computational power or main memory from other primary users. A single process of a PVM application is typically bound to the machine where it has been started for its lifetime, primary users may find their system in a state where interactive work is slowed by secondary users. As a consequence, they refrain from making the resources of their machines available to other users. For users, it becomes more and more difficult to find workstations on which to run their PVM applications. The situation becomes worse with an increasing number of users parallelizing their application using PVM. Hence, a facility is needed that maintains the ownership of the machines: as soon as interactive work begins on a machine, all processes which are borrowing resources must vacate. The mechanism for doing this are consistent checkpoints and process migration.

The remainder of this paper is organized as follows. The next section provides a brief overview on related work. In section 3 the CoCheck system to migrate processes is introduced. This is followed by a brief discussion of an interface to a resource manager (RM). Finally, the paper concludes with an outlook on possible future work.

2 Related Work

As described above, the situation in a NOW which is also used for interactive work is characterized by the fact that machines become available for executing a process of a parallel PVM application or are reclaimed by the primary user for interactive work. For single process applications environments exist that allow migration of a process from a machine, where interactive work starts, to another available host [4, 9]. Unfortunately, these systems do not support migrating parallel PVM applications.

Implementations like DynamicPvm [3] or MPVM [2] incorporate the process migration into the PVM system. In both cases the PVM library and daemons have to be modified. Whereas DynamicPVM only allows the standard message routing via the PVM daemons, MPVM also supports the direct communication of PVM tasks via TCP connections. DynamicPVM and MPVM introduce a message-forwarding technique to send messages to processes that have migrated to another host. The drawback of these approaches is the need to introduce message-forwarding and sequencing to insure the correct behavior of a PVM application and the enormous effort which is necessary to incorporate these changes in the current version of PVM.

In contrast to that, the system presented in [7] is implemented completely outside PVM. This has been achieved by providing wrappers for all PVM function calls. Therefore, no changes are required to the system with upcoming new versions of PVM. The migration of a single process is based on a consistent checkpoint of the whole application. This has the disadvantage that in case of the migration of a single process all processes need to write a checkpoint file which decreases the performance.

3 CoCheck: Checkpoints and Process Migration

Our protocol to create consistent checkpoints described in [7] proved to be viable and stable but was lacking good performance. This was due to several reasons. First, the checkpoint files which were produced by the single process checkpointer were too large. Then the migration of a single process required that all processes had to write a checkpoint file. Finally, all checkpoints were written to a network file system producing burst network traffic.

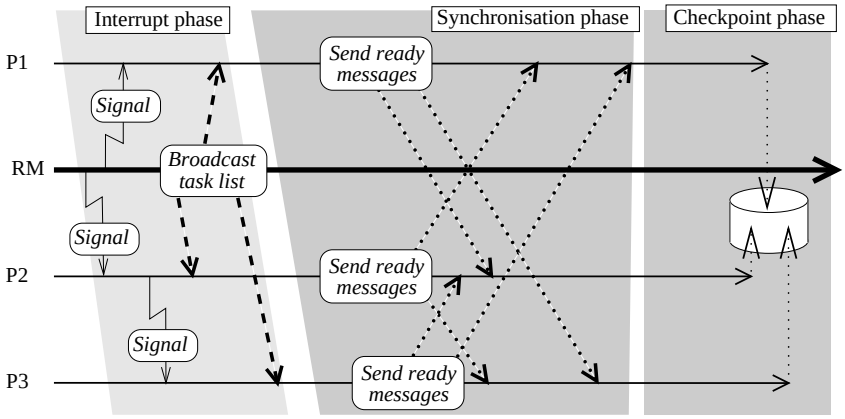

Figure 1: Steps to create a consistent checkpoint in CoCheck

Hence, in the latest version of the CoCheck checkpointer for PVM applications we have addressed those problems to improve the performance. First, we have incorporated the latest version of the Condor single process checkpointer. It reduces the size of the checkpoint file of each process [9]. Then, the protocol has been enhanced to allow that checkpoint files can also be written to local discs and that only migrating processes write a checkpoint file.

Both the creation of a consistent checkpoint of an application and the migration of a set of processes is initiated by the RM of the virtual machine by delivering a combination of a signal and a message to each process of the PVM application (cf. Figure 1). As the wrappers for the PVM functions guarantee that the PVM calls cannot be interrupted by a signal, it is possible to use PVM functions within the signal handler. PVM calls such as `pvm_recv` present a problem because they cannot safely be interrupted by a signal. Here, the message which is sent together with the signal can be used to force the interception of the signal. The wrapper functions for the blocking PVM calls have been implemented in that way. In addition this message contains information about all tasks of the application, which tasks need to create a checkpoint file, the location of the checkpoint file and if the process should exit. The location can either be a file on a local or network file system or a network address. The network address can be used in case of process migration, where the checkpoint can be directly transfered to the new process on another host. Exiting is only necessary if a process must vacate a machine.

After the signal and the task list have been intercepted all the tasks send specially tagged “ready” messages to each other. As PVM guarantees the delivery of the messages in sending sequence each task can draw the following conclusion: after all ready messages from all other tasks have been received there are no more outstanding messages on the network for this task. Hence, it can be seen as a normal stand alone process without any network state. While each process waits for the reception of the ready messages, it collects all other incoming messages in its address space, so that they become part of the checkpoint file when it is produced and can be retrieved after restart from the wrapper functions.

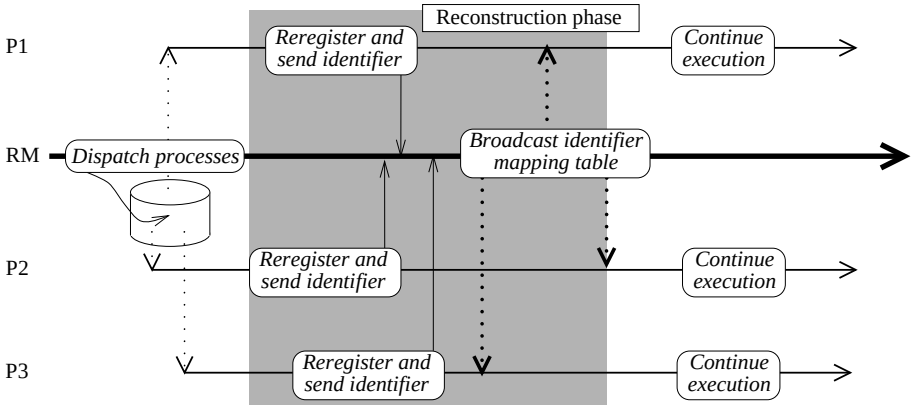

Figure 2: Restarting from a checkpoint.

After each process has collected all the ready messages it disconnects itself from the PVM system and uses the Condor single process checkpoint library to create a checkpoint. Depending on what was specified in the task list message the file is either written to disk (local or via a remote file system) or its state is transferred directly to a remote process which will become the migrated process. If the checkpoint has to vacate the machine it exits after the creation of the checkpoint is complete. Otherwise it directly begins the restart protocol (cf. Figure 2).

At the beginning of the restart phase all processes are stand alone processes. As the first step to continue the parallel PVM application, each process rejoins PVM and gets a new task identifier (tid). This tid is then sent to the RM which can set up a tid mapping table from original tids to current tids. As the application should not be aware of the fact that a process has migrated or a checkpoint was taken and the application has been restarted from that checkpoint, the tids which each task got, when it enrolled for the first time, are used throughout the lifetime of the whole application. To achieve this transparency of the tids the wrappers use a mapping table each time a tid is referenced in a PVM call. This mapping table is set up and distributed to all tasks by the RM. After each task has received its copy of the mapping table it can continue normal execution. The wrapper functions take care of retrieving messages that have been stored in the restored user address space before other incoming messages.

As already mentioned one requirement is to have a signal safe extension of PVM. CoCheck achieves this by providing wrapper functions for all PVM calls which block signals. This introduces an additional overhead of about 2 percent during normal operation for sends and receives.

4 The Resource Manager Interface

As described above, one of the key components of the CoCheck protocol is the Resource Manager (RM) process. The use of a RM process was introduced to PVM in version 3.3 [6]. The goal of creating the RM interface was to allow users to customize the handling of resource management decisions, and to provide an easy way of interfacing to existing scheduling systems. A RM process assumes responsibility for

making resource allocation and process placement decisions. In order to make these decisions, the RM must have information on the global state of the virtual machine. Because it already maintains this state, it is natural for it also to control the CoCheck protocol.

The Resource Manager acts in two ways during an execution of the CoCheck protocol. The low level responsibility is to provide the list of running tasks to the application processes, and to establish a new task identifier mapping at restart. Here, the RM takes advantage of its global state information. Since the RM needs to maintain a list of all active processes in order to make load balancing decisions, it can easily provide this list in the CoCheck protocol. At restart time, the RM collects information as tasks reconnect to PVM, and provides the new mappings to each task.

At a higher level, the Resource Manager must determine when a checkpoint or migration is required. This can be done either automatically or as the result of a user request. An automatic checkpoint would take place when the RM determines that one or more hosts being used by an application must be released to their owners or to other secondary users. When sufficient machines are available to replace those which must be released, a migration rather than a checkpoint can be performed. It may also be desirable for a user to trigger a checkpoint or migration for fault tolerance or load balancing purposes. Therefore, we expect to develop a user interface to the CoCheck mechanisms.

One of the goals of introducing the notion of the RM process to PVM was to interface to existing workstation scheduling systems such as Condor, Codine, LoadLeveler, etc. All of these systems are able to monitor resources and determine when they have been reclaimed by their owners, and allocate individual machines to user applications. Using these existing facilities, a resource manager can determine when a host must be vacated, and determine whether a migration is possible or whether a full checkpoint is needed. The ability to use the services already available in these scheduling systems allows us to bring CoCheck into large, existing clusters, and gives these systems much more flexibility in running PVM programs than ever before.

5 Conclusion and Future Work

The latest version of CoCheck allows for creating checkpoints of PVM applications. Due to the enhancements discussed in this paper CoCheck now also provides a means to efficiently migrate single processes. This has been achieved by restricting the creation of checkpoint files to the migrating processes. Local discs can now be used to store those files or the checkpoint can be directly transfered over a TCP connection to the new host of the process.

The CoCheck environment comprises a set of libraries, a simple RM for PVM and a program to use CoCheck's functionality. Applications which should benefit from CoCheck's features simply need to be relinked against the CoCheck libraries. The user can then use the supplied command to either create a checkpoint, restart an application or migrate certain processes. The CoCheck system is publicly available under the terms of the GNU public license. To obtain the software refer to the CoCheck homepage: http://wwwbode.informatik.tu-muenchen.de/~stellner/CoCheck.html.

In addition to the supplied control command CoCheck offers an API with which the users can request CoCheck functionality within their applications. This includes the creation of checkpoints or the migration of processes. Hence, the user is capable of integrating dynamic load balancing into his applications, where the application can request to migrate a set of processes to new hosts, which are more powerful or less

loaded. Together with the PVM feature of notifications, CoCheck can be used to detect failures of machines and restart the application from a formerly created checkpoint.

CoCheck provides a preemptive global scheduler to RM processes. Hence, in the future we will focus on using this service in the RM. In addition to the use for RM, CoCheck will be incorporated in THE TOOL-SET[5]. Apart from providing basic checkpointing, CoCheck will be used within THE TOOL-SET to provide the core migration facility for system integrated dynamic load balancing. The debugger of THE TOOL-SET will use CoCheck's checkpointer to provide cyclic debugging, i.e. during a debugger session the user can create a checkpoint and restart the program from that state all over again to examine it.

References

[1] T. E. Anderson, D. E. Culler, and D. A. Patterson. A case for NOW (networks of workstations). *IEEE Micro*, 15(1):54–64, February 1995.

[2] J. Casas, D. Clark, R. Konuru, S. Otto, R. Prouty, and J. Walpole. MPVM: A Migration Transparent Version of PVM. Technical report, Dept. of Computer Science and Engineering, Oregon State Institute of Science and Technology, 20000 NW Walker Road, P.O.Box 91000, Portland, OR 97291-1000, 1994.

[3] L. Dikken, F. van der Linden, J. Vesseur, and P. Sloot. DynamicPVM. In W. Gentzsch and U. Harms, editors, *High-Performance Computing and Networking, International Conference and Exhibition Volume II: Networking and Tools*, volume 797 of *Lecture Notes in Computer Science*, pages 273–277. Springer Verlag, Berlin, April 1994.

[4] Genias Software GmbH, Erzgebirgstr. 2B, D-93073 Neutraubling, Germany. *CODINE User's Guide*, 1993.

[5] T. Ludwig, R. Wismüller, R. Borgeest, S. Lamberts, C. Röder, G. Stellner, and A. Bode. THE TOOL-SET — an integrated tool environemnt for PVM. In *Proceedings of the 2nd European Users' Group Meeting*, Lyon, September 1995. Editions Hermes.

[6] J. Pruyne and M. Livny. Providing resource management services to parallel applications. In J. Dongarra and B. Tourancheau, editors, *Proceedings of the Second Workshop on Environments and Tools for Parallel Scientific Computing*, SIAM Proceedings Series, pages 152–161. SIAM, May 1994.

[7] G. Stellner. Consistent Checkpoints of PVM Applications. In *Proceedings of the 1st European PVM Users Group Meeting*, http://www.labri.u-bordeaux.fr/~desprez/CONFS/PAPERS/abs010.ps.gz, 1994.

[8] V. S. Sunderam, G. A. Geist, J. Dongarra, and R. Manchek. The PVM concurrent computing system: Evolution, experiences and trends. *Parallel Computing*, 20(4):531–545, April 1994.

[9] T. Tannenbaum and M. Litzkow. The Condor Distributed Processing System. *Dr. Dobb's Journal*, (2):40–48, February 1995.