DynamicPVM

Dynamic Load Balancing on Parallel Systems*

Leen Dikken[1], Frank van der Linden[2], Joep Vesseur[2] and Peter Sloot[2]**

[1] Shell Nederland Informatieverwerking
Rijswijk
[2] Parallel Scientific Computing and Simulation Group,
Faculty of Mathematics and Computer Science,
University of Amsterdam
Kruislaan 403, 1098 SJ, Amsterdam

Abstract. This paper describes DynamicPVM, an extension to PVM (Parallel Virtual Machine) [1] . PVM enables users to write parallel applications using message passing primitives and statically places the parallel tasks on a collection of nodes. System schedulers schedule atomic jobs over a predefined number of nodes. DynamicPVM addresses the problem of scheduling parallel tasks over a set of nodes. It therefore has to integrate a process checkpointing, migration and restart restart mechanism with the PVM runtime support system. DynamicPVM facilitates an efficient use of existing computational resources for computational jobs consisting of parallel subtasks. Typical target HPC platforms for DynamicPVM are multi user, multi tasking, loosely coupled processors.

Introduction

The number of workstations in industrial and academical institutions has grown tremendously over the past years. A migration from centralized mainframes to collections of these high-performance workstations connected by LANs is taking place. These new loosely coupled parallel systems require new programming environments that provide the user with the tools to explore the full potential of the distributed resources.

Although these parallel programming systems provide the user with large amounts of processing power, the behaviour of these systems under changes of the environment dramatically influences the usability and efficiency of the system.

Environmental changes (e.g. increase of demand for processing power or a decrease of the number of available processors) impose the necessity to migrate tasks between nodes in order to improve the resource utilization. DynamicPVM

* Appeared in “High-Performance Computing and Networking”, proceedings 1994, volume 797, pp. 273–277.

** All correspondence with regard to this work can be addressed to this author at peterslo@fwi.uva.nl

aims to add system scheduling mechanisms on top of tasks running in parallel on this new type of parallel system.

Our primary objective is to study models describing systems like DynamicPVM that dynamically adapt to changes in the environment. To validate these models, experiments with actual implementations of such dynamic systems are required. The work described here presents a pilot implementation of DynamicPVM.

The choice for PVM is motivated by the fact that at the time of writing it is in use by more than 10,000 users and thus is considered as the *de facto* standard environment for parallel programming. The process migration primitives used in DynamicPVM are based on the checkpoint-restart mechanisms found in a well established global scheduling system, Condor [2].

In Table 1, several different aspects of load managing for the three systems discussed in this paper are shown.

	Condor	PVM	DynamicPVM
intended usage	longer running background jobs	parallelized distributed application programs	
unit of execution	job	task	
load managing objective	load distribution	load decomposition	both
schedule policy	dynamic load sharing	cyclic allocation	dynamic load balancing
schedule objective	resource utilization	application response time	both
performance objective	efficiency	effectiveness	both

Table 1: Aspects of load managing in Condor, PVM and DynamicPVM.

Parallel Virtual Machine: Runtime Support System for Parallel Programs
PVM offers primitives for remote task creation and inter process communication (IPC) to the user. Both point to point communication and broadcasts are supported. Newly created tasks are assigned to the available nodes using a cyclic assignment policy; PVM does not integrate any load information of the nodes into the assignment policy. Once a job is started, it runs until completion.

PVM as such does not support operations on files but PVM tasks can share files through the Network File System (NFS). Terminal I/O is transparently redirected to the terminal where the job was initiated, the *PVM console*.

At startup time each processor in the PVM pool runs a PVM-daemon representing that node. Requests to create a new PVM task onto a host are sent to the host's daemon which executes the requested binary. The daemon also takes care of IPC generated by or addressed to tasks on the host: tasks direct their communication to their local daemon which takes care of delivery to the daemon of the destination task. The remote daemon delivers the message to the destination task.

PVM uses XDR [4] as eXternal Data Representation, therefore messages can easily be sent between different architectures, thereby facilitating the use of heterogeneous processor pools. Apart from IPC using the PVM daemons, users are also able to initiate their own communication channels (TCP connections) which require no intervention of the PVM daemons.

Condor: Runtime Support for Job Scheduling
Condor stems from the observation that many of the—constantly increasing number of—workstations in academic institutions are lightly loaded on the average. Most workstations are intended for personal usage, which has a typical activity pattern where machines are only used for a small part of the day. As a consequence many cycles are idle or unused during the day. Typical figures of large pools of workstations have a mean idle time of 80% [2].

To address this problem, Condor implements a global scheduling based on dynamic load balancing by job migration. Condor monitors the nodes in its pool keeping track of their load. New jobs are spawned on lightly loaded nodes and tasks from heavy loaded machines can be migrated to lighter loaded ones. When Condor detects interactive usage of a workstation all jobs are evacuated in order to retain the sympathy of the workstation's owner. To implement this job migration Condor creates checkpoints on a regular basis. These checkpoints can be restarted on another machine.

The Condor scheduler consists of both a centralized and a distributed part. Each node in the pool runs a small daemon that gathers statistics about the node and forwards this information to the central scheduler. This information is used optimize the available processing power.

System calls made by a Condor process are redirected to the host that initiated the job. In this way the user is freed from the complications of the checkpointing mechanism since it is completely transparent to the job.

Using Condor, it is not possible to migrate jobs consisting of parallel tasks that cooperate to complete a job since Condor does not provide any IPC primitives and the standard UNIX IPC primitives are host-addressed and therefore not suited for migrating tasks.

Combining PVM with an extended version of Condor's checkpoint-restart facility makes it possible to apply global scheduling to parallel tasks.

DynamicPVM: Runtime Support System for Job Scheduling of Parallel Tasks

In DynamicPVM we add checkpoint-restart primitives to the PVM environment. Most of PVM's features are compatible with the checkpoint-restart mechanism we use, with the exception of the inter process communication. We present a protocol that ensures that no messages get lost whenever a task is migrated. This protocol involves a special role for the PVM-daemon that initiated the computation, the Master-daemon. We also present an extension to the PVM IPC routing mechanism in order to redirect messages to the new host after a task has migrated.

DynamicPVM's task migration facility consists of four principal components:

1. A global scheduler that initiates job migration (not addressed in this paper)
2. Task checkpointing including a method to indicate checkpoint save moments
3. Actual Task migration
4. Task restart and updating of routing tables to indicate the task's new location.

Task Checkpointing
Checkpoints are created using the standard Condor checkpoint code. This code creates a core-dump of the task onto a shared file system (NFS). Condor appends extra information to the core in order to be able to restore the state of the process when it is restarted again.

Checkpointing cooperating tasks introduces new conditions compared with checkpointing stand-alone tasks. Checkpoints should be avoided, e.g., when a task is communicating with another task. To enable DynamicPVM to determine which situations are save to checkpoint, we have introduced the concept of critical sections. When a task is inside such a critical section, any checkpoint request is postponed until the task leaves the critical section. All PVM communication routines are contained in critical sections.

In the absence of a global scheduler, the task itself initiates any possible checkpoint requests by calling a new routine `pvm_move`.

As soon as the task initiates a checkpoint it sends a message to all tasks with which it shares a communication channel. These remote tasks will then close the channel and issue an `accept` call in order to re-establish the connection as soon as the task is restarted.

Task Migration
The main objective of the DynamicPVM task migration protocol is to guarantee migration transparency, i.e. to allow the movement of tasks within the system without affecting the operation of tasks in the system. With respect to a PVM task selected for migration this implicates transparent suspension and resumption of execution. With respect to the total of cooperating PVM tasks in an application communication may be delayed but not fail due to the migration of one of the tasks.

First we create a new task context at the new node by sending a message to the remote PVM daemon. The task's new daemon is now able to receive any packets for the migrated task.

Next, the new routing information is sent to the task's new daemon, the Master PVM daemon and finally updated in the routing tables of the task's old PVM daemon. All messages directed to the migrating task will from now on be send to the new node.

At this point the task is actually checkpointed and written to disk.

Task Restart
As soon as the checkpoint is finished, the task's new daemon is notified. It then restarts the task which restores any saved state information including the re-establishment of any communication channels that were closed at checkpoint

time. The daemon then delivers any messages that arrived during the creation of the task's checkpoint.

Any future messages that are directed to the task's old daemon will fail. Whenever this happens, the sending daemon will contact the Master daemon to obtain the task's new location to which the message will be directed next.

Current Status and Ongoing Research

DynamicPVM is currently implemented on a cluster of IBM RS/6000, AIX32 machines [3]. Due to a conflict between Condor's restart mechanism and PVM's startup code, we are currently not able to checkpoint tasks that perform file-I/O.

Tests performed with the pilot implementation strongly support the usability of our integrated approach. In the very near future we will design additional experiments for quantitative testing of the behaviour of DynamicPVM.

Our final goal is to use probabilistic models of jobs in a dynamic environment and validate them by experiments with an actual implementation. We have added several primitives to PVM that allow us to monitor PVM tasks. Using these primitives we will implement a scheduler based on our models. Simulations using our models will gain us insight in the behaviour of DynamicPVM and will be used to improve the scheduling mechanism. The scheduler is a topic of future research, and it is expected to be dealt with in the coming months.

References

1. V. S. Sunderam. *PVM: A framework for parallel distributed computing*, Concurrency: Practice and Experience, vol. 2(4), pp. 315–339, December 1990.
2. M. J. Litzkow, M. Livny. *Condor — A hunter of idle workstations*, 8th International Conference on Distributed Computing Systems, San Jose, California, June 1988.
3. L. Dikken, *DynamicPVM: Task Migration in PVM*, Technical Report, Shell Research, ICS/155.1, November 1993.
4. Sun Microsystems, *XDR: External Data Representation standard*, Sun Microsystems, Inc., 1987, RFC 1014.