The Distributed Monitoring Solution in Parallel Programs Developing

Lubica Sutovska
Ladislav Hluchy

Institute of Computer Systems, Slovak Academy of Science
Dubravska 9, 842 37 Bratislava, Slovakia

Abstract

This paper presents a distributed monitoring solution for tuning the performance of parallel programs. Our design of a monitoring/kernel tool offers advantages for optimization of parallel applications development and the aim of the monitoring is to improve parallel program development through analysis of the execution of parallel program parts. The primary intention of monitoring is directed towards timing analysis. The motivation behind our work is to offer the programmer a suitable environment which aims at simplifying such activities as to determine what data must be collected, where in the program that data can be collected, and to produce interesting data. The graphical tools are built around the monitor to support that activities. The goal of proposed monitoring/kernel tool is to be a part of integrated monitoring and debugging system in a HPCTI project. In this paper we describe the design and implementation of a monitoring/kernel tool (MKT) that provides continuous monitoring capabilities in the PVM environment by using software support.

1. Introduction

Due to deficiency of general methods for predicting dynamic behaviour of parallel programs running on parallel processing systems, it seems that the new programming tools and environments are keys to a greater success in a parallel software developing. Both, an effective application program and a system software tuning, they have forced the development of the new proper techniques for a parallel performance evaluation, with the primary goal of increasing a performance observability. The main objective of monitoring parallel applications is to focus on an anomalous program behaviour and critical performance bottlenecks. In the first case, the information on a program behaviour is collected in order to exclude the presence of errors on a system level or to correct any fault, given by an unexpected behaviour of a parallel program. In the second case, the time spent by the program in each activity of an application code is measured to allow the performance tuning based on adequate feedback techniques. Distributed systems lack both, a global state information and a sense of a global time. There is no total ordering defined over events that occur on different hosts in the PVM environment. Monitored data must be collected from several sites and integrated to obtain a coherent view of the execution of parallel program parts.

2. Overview of Monitoring/kernel Tool

A software post-mortem monitoring approach has been chosen for the MKT. The proposed monitoring/kernel tool uses software integrated into the system call libraries and the programmer does not need to add a special code to applications. The generation of the runtime information is performed during the execution of a parallel application and the utilization (reduction, analysis) of the gathered information takes place in a post-mortem fashion. In order to reduce the inherence from monitor, MKT is realized through distributed software parts that run on each hosts machine in the PVM environment. A software monitoring layer is shown in figure 1. MKT provides the distributed data gathering mechanism and the distributed transportation strategy. A MKT daemon processes coordinate and retrieve the data from applications, standardize the data (transform data to SDDF format), compress and log on a local workstation (PVM console).

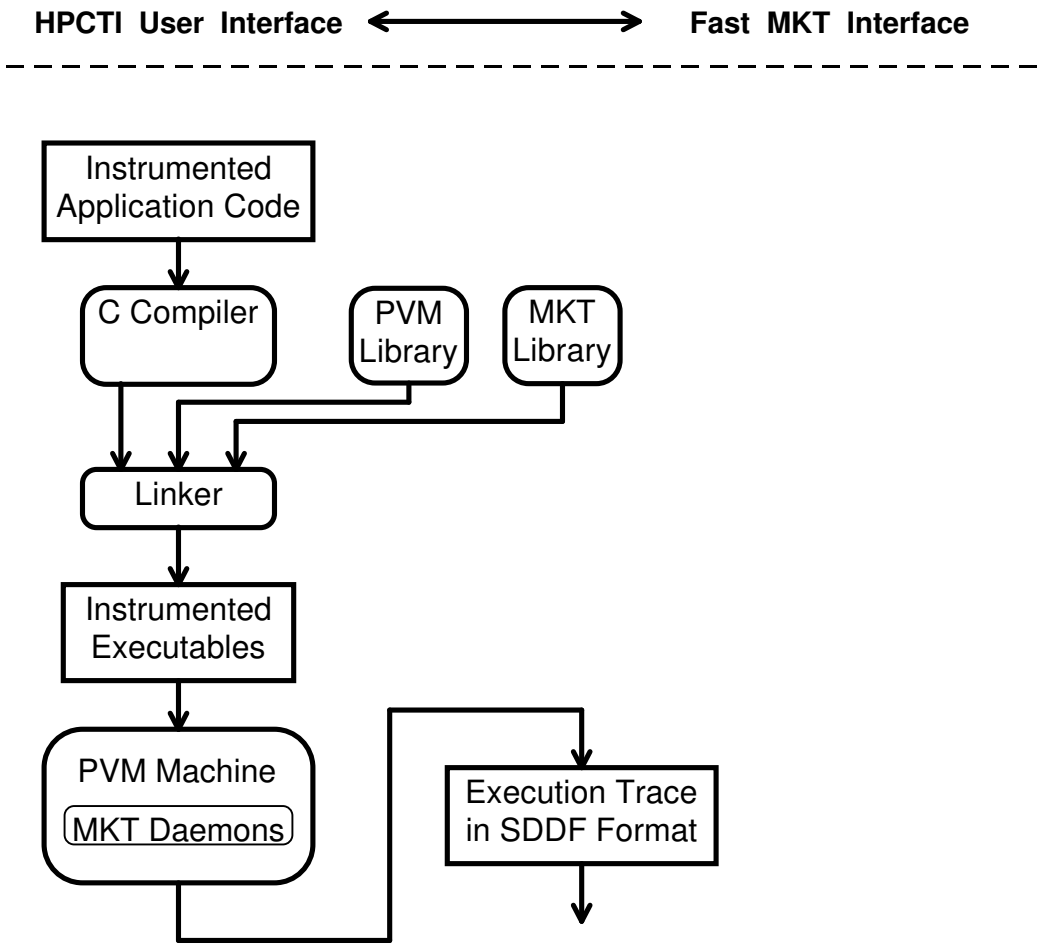

Fig. 1. A Software Monitoring Layer

We assume the data on monitored Grapnel relevant events is acquired through monitoring code to be included into GRAPNEL call libraries. In order to test our design of the MKT prototype and to determine the interference from the proposed MKT a temporary solution has been realised in the current version of the MKT prototype with including the monitoring code into a user source code by the help of a fast MKT interface. This fast tool interface enables to choose interesting locations of the application (event, process) for monitoring by the user. User-driven instrumentation and user-transparent instrumentation of the source code are assumed to be used simultaneously, if necessary. The user-driven instrumentation allows to keep perturbations small. The user-transparent instrumentation prevents programmers from inconvenience of detecting " bottleneck " locations manually. The structure of MKT environment is shown in figure 2. Measurements are done by distributed monitor processes which extract data using the instrumentation code. A trace file is produced after the program run by merging the locally buffered information and standardizing it into SDDF format.

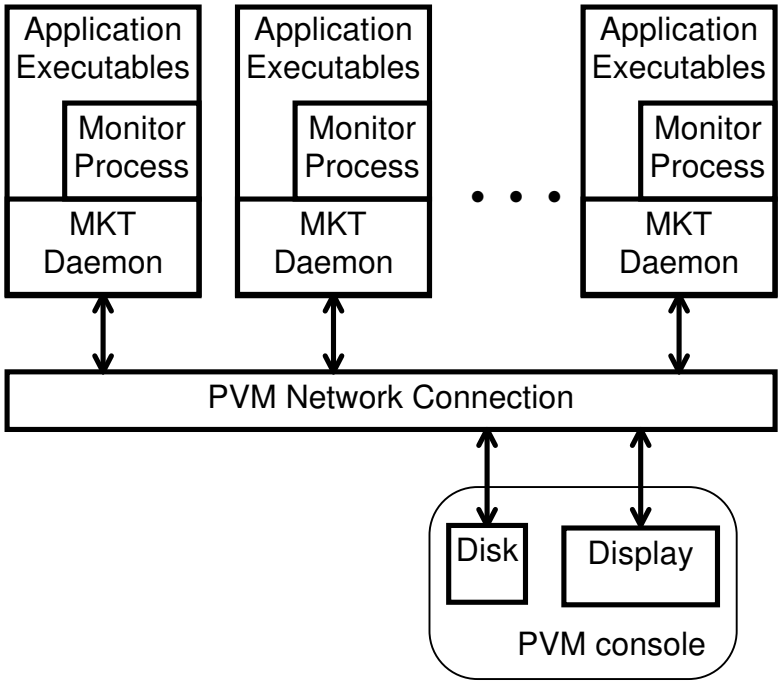

Fig. 2. The Conceptual Structure of the Distributed MKT Environment

3. The Functionality of the MKT Software

At the beginning of a measurement on each PVM host MKT daemon programs are started. MKT daemons initiate distributed monitor processes which are responsible for the treatment of the measurement data during runtime. They handle the measurement information locally on the respective PVM host - adding of time stamps and storing them in a buffer. The Interface between the distribute software MKT and other evaluation tools is a global trace file which contains the data of all hosts. It is produced after program run by merging the locally buffered information at the host machines. After an application terminated, the MKT daemon orders the gathered data in the local buffers by time, transforms it into the SSDF format, transports compressed event log data and stores it in a file at the user console (figure 3).

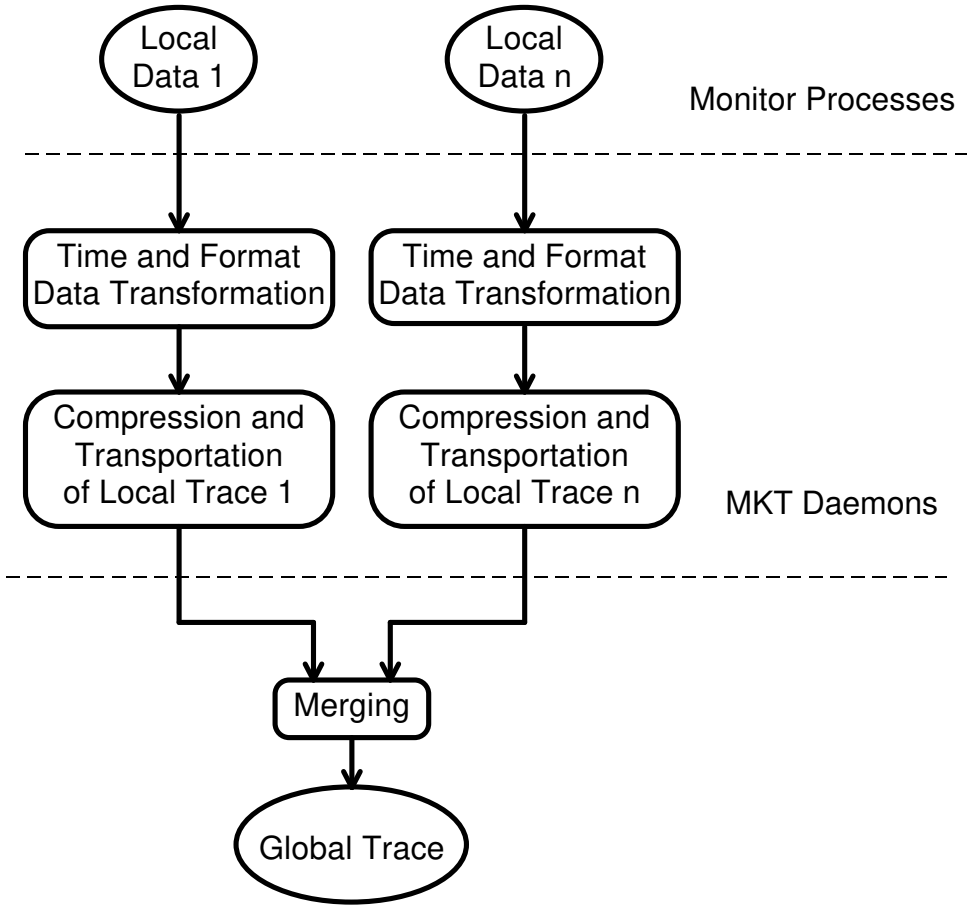

Fig. 3. A Sequence of a Global Trace Acquisition

To obtain a global system view, the MKT daemons provide data with a global valid time stamp which is obtained by the software solution using a global synchronization sequence at the beginning and at the end of the application's execution.

4. The Fast MKT Interface

The fast MKT interface have been developed as a part of an integrated HPCTI user interface. This graphical interactive tool enables programmers either an user-driven instrumentation method or an user-transparent instrumentation method to use. All instrumentation is considered to be performed automatically at the level of the source code. The tool provides programmer to run the same application with a different (small) set of events so the overhead of that monitoring is low because monitored events are relatively infrequent.

5. The Implementation Status

The realised prototype version of MKT (the monitor daemons that allow to use of a monitor library of commands) has been done in C language in a PVM environment. The idea of monotonically increasing timestaps was used for timing for every type of events. The values from two sequential communications (called header process and footer process) between any local -remote hosts pair are used to provide a transformation from local time measurements to global time values. Now we have working on implementation of fast MKT interface routines and we have testing the functionality of the MKT library. We plan a benchmark measurements with and without trace generation in order to assess the influence of MKT on a target system performance.

6. Conclusions

A software monitoring/kernel tool (MKT) was developed according to base objectives such as flexibility, easy to use, small intrusion, etc., as a part of monitoring and debugging system for HPCTI project. A distributed software MKT assists the programmer in writing efficient parallel programs. An additional work is continuing to support the full integration with other tools of HPCTI project.