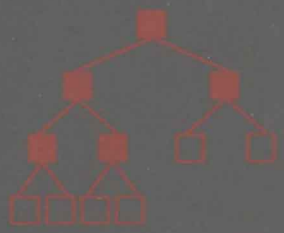

North-Holland

PERFORMANCE MEASUREMENT AND VISUALIZATION OF PARALLEL SYSTEMS

ADVANCES IN PARALLEL COMPUTING 7

G. Haring
G. Kotsis
Editors

PERFORMANCE MEASUREMENT AND VISUALIZATION OF PARALLEL SYSTEMS

Proceedings of the Workshop on
Performance Measurement and Visualization
Moravany, Czechoslovakia, 23-24 October, 1992

Edited by

Günter Haring
and
Gabriele Kotsis
Institute for Statistics and Informatics
University of Vienna
Vienna, Austria

1993

NORTH-HOLLAND
AMSTERDAM • LONDON • NEW YORK • TOKYO

ELSEVIER SCIENCE PUBLISHERS B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

Library of Congress Cataloging-in-Publication Data

Workshop on Performance Measurement and Visualization (1992 : Moravany nad Váhom, Czechoslovakia)
Performance measurement and visualization of parallel systems : proceedings of the Workshop on Performance Measurement and Visualization, Moravany, Czechoslovakia, 23-24 October 1992 / edited by Günter Haring and Gabriele Kotsis.
p. cm. -- (Advances in parallel computing ; v. 7)
Includes bibliographical references.
ISBN 0-444-89902-2 (alk. paper)
1. Parallel computers--Congresses. I. Haring, Günter, 1943- . II. Kotsis, Gabriele, 1967- . III. Title. IV. Series.
QA76.58.W673 1992
005.2--dc20 93-9512
CIP

ISBN: 0 444 89902 2

Printed on acid-free paper.

Printed in The Netherlands

PERFORMANCE MEASUREMENT AND VISUALIZATION OF PARALLEL SYSTEMS

ADVANCES IN PARALLEL COMPUTING

VOLUME 7

Bookseries Editors:

Gerhard R. Joubert
(Managing Editor)
Aquariuslaan 60
5632 BD Eindhoven, The Netherlands

Udo Schendel
Institut für Mathematik I
Freie Universität Berlin
Germany

NORTH-HOLLAND
AMSTERDAM • LONDON • NEW YORK • TOKYO

Preface

The Workshop on Performance Measurement and Visualization of Parallel Systems was jointly organized by the Institute for Statistics and Informatics, Department of Applied Informatics at the University of Vienna, as a member of the Austrian Center for Parallel Computation (ACPC) and the Institute for Control Theory and Robotics at the Slovac Academy of Sciences. Internationally well known researchers working in the field of performance measurement and visualization of parallel processing systems were invited to present their recent research results at the workshop. All together 14 papers were presented and a general discussion on the future trends in this topic was held at the end of this workshop.

To write parallel programs is much more tedious and complex than developing sequential programs. This is particularily true for distributed memory multiprocessor systems and message passing programming models. Performance aspects play an important role in this context. The performance of parallel programs depends on many different facts. Therefore it is important for any parallel programming environment to have appropriate performance monitoring and visualization tools included in it to give the programmer a good understanding of the dynamic program behaviour.

The papers in these proceedings focus on new approaches for monitoring parallel processing systems and new performance visualization techniques. Beside the description of fundamental, basic approaches to these topics extensions and recent developments of existing tools and concepts for new tools are presented, especially emphasising on the integration of monitoring and visualization techniques in parallel program development environments. A couple of case studies reflect the practical importance of these approaches. Some papers tackled the scalability problem (with massiveley parallel systems), both from the monitoring and the visualization point of view.

We would like to thank the Austrian Federal Ministry for Science and Research for sponsoring this workshop under the research grant no. 613.542, and the members of the organizing institutions whose efforts have been fundamental for the success of this event.

Günter Haring
Gabriele Kotsis

Table of Contents

Performance Measurement and Visualization of Parallel Systems
G. Haring and G. Kotsis (Editors)

The Massively Parallel Monitoring System a truly parallel approach to parallel monitoring

B. Tourancheau [a]
X. Vigouroux [b]
M.G. van Riek [c]

[a]LIP/IMAG, ENS Lyon, 69364 Lyon Cedex 07, France, btouranc@lip.ens-lyon.fr
This work was supported by ARCHIPEL SA under contract 820542, by the CNRS and by PRC C3.

[b]LIP/IMAG, ENS Lyon, 69364 Lyon Cedex 07, France, vigourou@lip.ens-lyon.fr
This work was supported by ARCHIPEL & ANRT under grant CIFRE 920 335

[c]ARCHIPEL, PAE des glaisins, 74940 Annecy le vieux, France, vanriek@archipel.fr

Abstract

The monitoring of parallel systems involves the collection, interpretation and display of information concerning the execution of parallel processes. This information can support the debugging, the performance evaluation, the understanding of the behavior of parallel programs. When jumping from parallel to massively parallel systems, the monitoring tools have to face the problem of scalability.

Concerning the user interface, the "displaying" of information (graphics, sounds, ...) must follow a more hierarchical and abstract point of view and on the user response-time point of view, the tool must implement a scalable no-bottleneck software architecture. In this paper, we try to answer that problem with a distributed monitoring solution, where most of the data treatment is done on the parallel target machine and where the classical data gathering bottleneck is avoided with a distributed storage and a parallel treatment of the interpretation. The "views" are also handled by different processors on different external displays. Our approach thus becomes fully scalable (in terms of collecting power, storage, computing and displaying power). Our only assumption is that the massively parallel target machine is able, for at least a part of its nodes, to run Unix-like system calls. It is realistic regarding what will be provided in the next generation of parallel machines. We also present the protocol established for the data communications in an implementation which runs on a LAN of SUN workstations using the Parallel Virtual Machine system.

1. Introduction

Programs for parallel MIMD distributed memory multicomputers are difficult to write, understand, evaluate and debug, while at the same time parallel algorithm design is much more complex than the sequential one. The motivation behind the monitoring of parallel

programs is to improve parallel program development through analysis of the execution of parallel programs. Monitoring tools play thus an essential role in this improvement process and form as such an essential part of future programming environments.

The GPMS (General Parallel Monitoring System) project of the LIP has been a first step in parallel monitoring systems and provided us with a complete monitoring environment. The portability of our approach has been demonstrated by the porting of the GPMS system under two different operating systems.

While working on the GPMS project, it became clear that the classical centralized approach to monitoring is non-scalable and thus unsuitable for massively parallel architectures. At the same time, our experience with the porting of the environment to different operating systems strengthened our idea that it is up to the vendors to provide the low-level monitoring functions (A fact that is verified by nowadays evolution of parallel machines).

One of most common approaches to monitoring and the one used in GPMS is a centralized post-mortem approach, in which runtime information is generated during the execution of a program and centralized in one location (usually a file). The analysis takes place in a post-mortem fashion [10]. Three different activities can be distinguished in the monitoring of programs [17, 18, 12] : the generation of the runtime information, the transport of the runtime information and the utilization (reduction, analysis) of the gathered information.

The gathering of the runtime information can either be performed by software (using software probes inserted in the code and executed like instructions), or with special dedicated hardware. The gathering of the runtime information is in any case a truly distributed activity that takes place locally on all the nodes of a parallel machine and thus is scalable.

The transportation strategy defines the way the information is transported from the location of generation to the location of usage. Usually, the adopted transportation strategy consists of centralizing all the runtime information at one point in the system (one trace file). The cost of this centralization is considerable in communication bandwidth and results for larger systems into a communication bottleneck and is therefore not scalable. Only the distribution of the runtime information over different locations can possibly result in a monitoring system that scales up to massively parallel systems.

Whereas most of nowadays monitoring systems use the host for the processing and representation of the generated runtime information, this approach to the processing is limited by the power of the host processor and by the storage capacities of the host. Scaling centralized processing and access of the runtime information beyond 64 nodes can easily be shown to be hardly feasible ([15]).

In a similar manner, the number of different views on the same data raises a performance issue, whose impact can be easily realized by using the state-of-the-art performance visualization tool ParaGraph with a lot of views at the same time ([19]).

The Massively Parallel Monitoring System (MPMS) project is the logical consequence of our experience with GPMS and aims to provide a truly massively parallel monitoring environment. As such MPMS addresses the bottlenecks of the classical centralized monitoring approach.

As opposed to the lack of scalability of the preceding approach, MPMS proposes a

solution where the parallel machine is used to produce, reduce, and display information about the execution : i.e. to perform all of the monitoring steps.

In the following sections, we will start be outlining the assumptions that underlie the MPMS project. Next, the characteristics of the MPMS system will be described, whereas a third part presents some examples of views that we have constructed on top of the data communication system. We will conclude with the implementation of MPMS on a LAN of workstations simulating the targeted Massively Parallel Computer.

2. Definitions and assumptions

2.1. Presentation of the massively parallel machine model

The massively parallel machine that is targeted in the MPMS project is a MIMD type distributed-memory machine composed of several thousands of nodes. At least some of the nodes in the machine provide high-level OS capabilities, such as X-like client commands (displays control) and secondary storage. These capabilities are fundamental for the distribution of the data and processing as we will see later.

Although the described machine does not correspond to any existing machine at the moment, it closely looks like the projected architecture of both US and EC supercomputing projects. Please note that several new generation systems are also very close to these assumptions ([8, 16]).

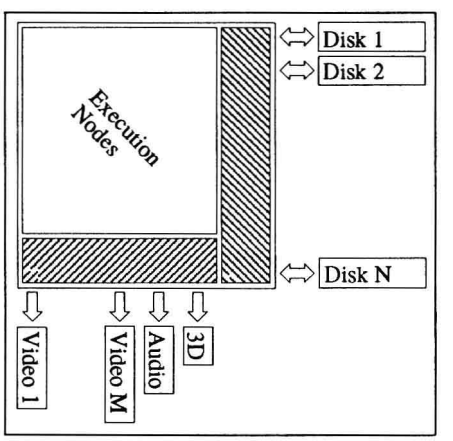

Figure 1. A massively Parallel Machine : the gray area represents the nodes with high OS capabilities.

2.2. Assumptions related to the available monitoring information

The runtime information is supposed to be available in the form of event-records, that describe events during the execution. Our actual visualization tools assume that a global timestamp is associated with each traced event, but no assumption is made about how this timestamp is obtained. Several algorithmic methods exist in [5, 9, 13, 4]. Remark

that all the assumptions about the ordering of the events are related to the developed views, but are not a constraint.

A *trace file* is defined as a set of event-records concerning the execution of a program on a set of nodes of a machine. Depending on the distribution of the runtime information, several trace files may thus be necessary to completely describe the execution of a parallel program. Trace files are usually stored on secondary storage media, but for real-time purposes it could be imagined that each node possesses its own RAM-disk secondary storage. For the sake of simplicity, we assume that, in each file, the records are sorted by processor number and timestamp.

2.3. Representation of the runtime information

By representation, we mean any ways of interpreting the data for the user's understanding and by view their realizations. All the human senses should be employed to overload the user so that he could synthesize very accurately the information. For example, we may think about the use of stereo sound to represent each node efficiency and their position (according to the topology) using sound widgets [15]. The use of the stereo, also, permits us to show the position of the messages in the topology. No assumption about the trace record format is made, a self-defining format [15] or an external file description format [14] or any internal format can be used.

3. MPMS - the main ideas

The main idea in the MPMS project is to develop on top of the vendor's low-level monitoring functions (generation of event-records), a powerful, scalable and flexible tool for the analysis of parallel program behavior. Besides the independence of the underlying monitoring data generated, MPMS must include the possibility to integrate future developments (extensibility) and support the increase in the number of processing elements (scalability).

To meet the above requirements, we have created two different layers of functionality in the tool. At the low-level, the vendor environment will provide a distributed data generation mechanism for the runtime information and at the mid-level we provide the tools that allow the efficient, easy and scalable access to this information. On top of the mid-level layer, we provide the users with high level tools such as standard and new visual analysis software for analysis and replay of the execution. We also provide a framework which facilitates the construction of new tools that will better fit the application problems.

For the realization of the two layers, we designed a client-server architecture. Trace-servers integrate the functions for the access to the runtime information and their transportation. A trace-server can handle one or more trace files. The distribution of the runtime information is transparent to the clients and is handled through an inter-server protocol. The synchronization of multiple views on the same data is realized within the protocol by master-slave commands on clients.

Clients encapsulate completely the processing of the runtime information. They obtain this information by requesting the needed information to a trace-server independently of the location of the information. All the processing on the runtime information in done within a client and as such a client can be considered as an independent object.

In general, the amount of exchanged information is limited by the fact that information is delivered on-demand only. In comparison in many existing systems all the information is conveyed even if not needed.

Several advantages result from the above architecture. First of all, by adding new servers the storage bottleneck can be avoided, since there is no competition for the same central storage medium. The distribution of the processing to different clients, on the other hand, limits the performance bottleneck on the host processor. Since clients might, but do not have to use the same display, the display performance bottleneck that might appear with many different simultaneous views can also be avoided. Extensibility of the monitoring environment is guaranteed by the client-server protocol. As long as this protocol is respected, new clients can be added without interference with existing clients (except for the load of the trace-servers). This independence of clients also allows for the existence of completely separate programs using the same runtime information. One could for example imagine some clients using the same information for separate tasks such as map calculations and performance analysis.

Thus, the main ideas that lead us to the MPMS design are : a strict separation between the generation, data storage and transportation and usage of runtime information. This ensures a total independence from any pre-define trace format or from the type of trace generator (hardware or software). The independence of the different parts of the system allows their distribution in a parallel computing environment. Clients that encapsulate completely the processing of the runtime information and generate only requests for data to the associated server. They run on the MPM nodes that provides enough OS capabilities. Servers run on any nodes controlling secondary storage (RAM-disk or disk). The data exchanges are done within the MPM interconnection network.

Compared to classical monitoring of parallel machines, the first phase (generation) remains unchanged excepted that it is handled by the vendors and that the OS allows the creation of many trace files (each one corresponding to a part of the machine), the second phase (transport) is delayed and reduced because the collected data stays in the parallel machine storage area and is filtered and compressed so that only the necessary records will be communicated to the client views.

The project is thus also twofold, it consists in designing a parallel program (SERVER) that manages the trace files and designing scalable visualizations (CLIENTS) that use the system servers for gathering the data they need. The link between these two parts is realized with a classical client/server protocol. Requests are sent from the views to the servers which send back the desired filtered information.

Hence, the typical performance bottlenecks of classical monitoring systems are avoided and the system is scalable. MPMS thus provides a truly parallel monitoring tool.

4. The trace servers

We call *trace server*, the tool that runs on a node of a parallel machine and that is able to read files, to communicate over the parallel machine network with other trace servers and its attached views. We assume that the execution generates a file for each logical parts of the machine that has been used. As the execution takes place on different part of the machine, a trace server will be started on the nodes that manage a disk which contains

trace data. At this level, the service will be as clever as possible to reduce the amount of data communication. With this approach, we do distribute most of the monitor elements and treatments.

The goal of this software is to allow everything that a view will need for the analysis of the massively parallel application. Thus a trace server must be able to manage several trace files, move inside them without any constraints.

A client can request data from all the nodes that participated to the given execution. The trace servers will keep a list of the trace servers that has worked for the computation in order to minimize the number of messages between trace servers.

As we want to be able to synchronize views, the trace servers must be able to know the monitoring system environment, i.e. the other trace servers and which views are existing. That means, once a view is created, the trace server that serves it, broadcasts the creation information to the servers. For the same reason, the same message exchange is done when a view is dismissed. The synchronization is done in a master-slave protocol way, the slaves receiving the information for their new position in time.

Because some views can be killed without sending an announce to the trace server, there is a verification during every crucial phase that the context is still valid. For example, at the time of a synchronization between several views.

As the trace files are distributed among the massively parallel computer disks, the main problem is to communicate the information required by a view in an efficient way. This is realized by filtering of the data requested on each field of the event-records. Compressing is also possible if the client possesses the decompressing module. More complex pre-computations at the server level are possible such as summations, simple statistics (average, relative percentage, ...). All this commands are obtained by parameters in the request messages.

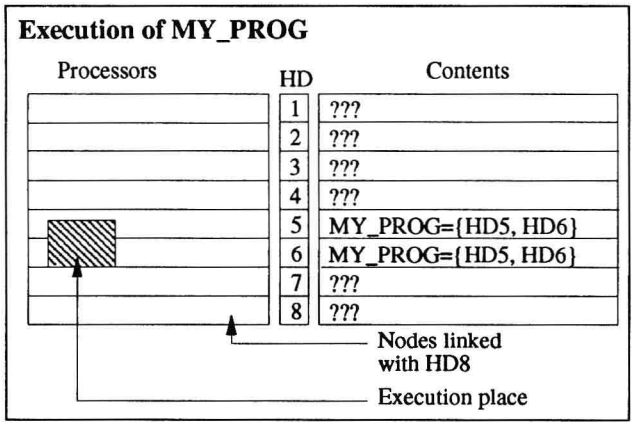

Figure 2. Example of an execution and the corresponding hard disk content.

A client can be started on any node by a command in a trace server control window. The client consists of a process that manages for example a window display and requests

its data to the trace server it was started from. In order to be completely independent, it encapsulates completely the processing of the runtime information and generates only requests for data to the server with appropriate filtering parameters. A client can allow to be synchronized if it contains the synchronization module.

A client view is running on one or several nodes which have enough OS capabilities for the I/O needs (Xwindows ...). This node communicates to one trace server at a time. The information asked to the trace server is either written on its associated hard disk or not. As the trace server got the list of the trace servers which compose the execution, it collects from them the requested events. The result is then sent to the client view.

Following our assumptions, the parallel machine can be linked to several human interfaces like screens where the high level OS nodes are able to display their views.

Hence the number of human interface devices can be raised when the number of nodes increase without any loose in scalability.

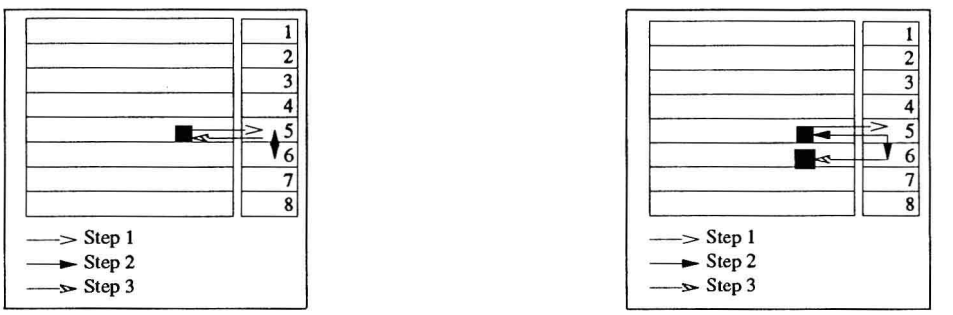

Figure 3. Simple and distributed views on MY_PROG

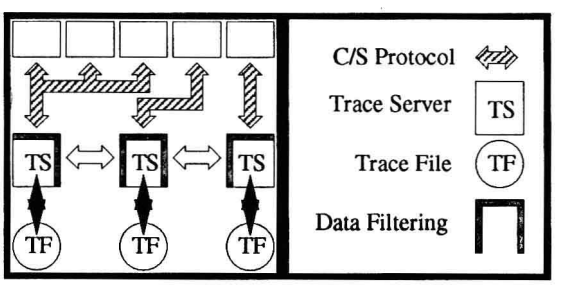

Figure 4. resulting software architecture

Our aim of efficiency implies that we try to keep as low as possible the data amount to be communicate between the views and the trace files. That's why we introduced in

the data requests the possibility to precisely define which type of data is needed. This is done by general filtering parameterized commands associated to the different fields of the recorded data.

The timestamp field allows for example to ask for only a period of time of the execution or the trace-type field to look only communication events, ... But to be able to reduce again the data transfer, we will allow a simple boolean algebra (and, or, not) on the filters.

Remark that as these filtering operations are managed at the lowest possible level (on the trace servers), so there is no communication of the whole trace files. The network load can be directly tuned by the user when choosing the tools to be started.

As in all software products, we try to have a modular software architecture. We preserve the independency of the monitoring tool from the trace generator. That means that hardware or software collection can be used by working on an internal trace-data format and providing input and output format translators. The server requests and answers can thus be encapsulated to provide any data format.

5. Views

The second part of our system consists in the means given to the users to interpret the performance variations and to look at the parallel behavior of their programs.

Our aim of scalability leads us to a complexity study which aims to show the ability of a view for scalability regarding several computation and visualization complexity as a function of the number of processors p.

Some classical views have been developed to show the power of the implemented version (we chose examples in the ParaGraph [7] views).

We also introduce the hierarchical decomposition of classical views. A mean which allow scalability for classical views by considering groups of processors, groups of groups, etc ... For instance, the problem of automatic research of the localization of a problem (efficiency to poor or this kind of information) The user can choose the minimal efficiency for a given group level and start a search. The tool will stop at the first group under the minimal bound and select it. Then the user can go down a level in th hierarchic display and restart the automatic search to precise the problem location.

The main characteristics of the views are : to be able to be synchronized with others, to be dynamically created and destroyed. Different views can ask for the same executions events at the same time (that permits comparisons between different instants in time for example). If the massively parallel program allows multiple processes per node, as soon as the files are accessible, the trace servers can access data and thus the views can start during the execution (pseudo real-time monitoring).

The clients can either be sequential (one process on one node) or distributed among different nodes of the machine. The second case is interesting because of its scalability in power but it causes communication problems that are let to the view programmer.

The figure 3 shows the different messages exchanged during one request. The first step consists of asking for the information. Then, the trace server asks its homologues the information they own. If the view is sequential, everything (after filtering) is merged by the first trace server and sent back to the view, otherwise, the merging step is avoided and each part of the information is sent to the different window processes.

5.1. Notations and classification

In this section,

p : the number of nodes,
T : the trace file total duration (last event timestamp - first event timestamp),
N : the number of events,
n : the current event,
d : the shortest distance between two represented nodes (in pixel, tone, ...).

This last variable gives the maximal number of representable processors and for example allows us to compute the surface of the view : if a node is represented by circle, this circle must have a diameter greater than a pixel on a workstation screen.

In order to reflect, referring to the time, the data needs of every views we introduce a qualitative classification of the data that will be requested by a view. A view can need the whole events of a trace file to compute its representation(s) (attribute WHOLE), the past (PRED), a part (PART), one event (EVENT).

These definitions allow us to have a quite formal approach to the needs of each representation and to partition them into classes. We are interested in :

- the amount of information needed to compute the representation,
- the surface of the view (for visual ones),
- the time to initiate the view at a timestamp,
- the time to move from one event to the next (or previous) one,
- the computational complexity of a representation (in number of events, operations, ...),

The ability of the views regarding scalability can thus be studied trough "objective" criteria. Remark that at least, none of the selected criteria can be of an order greater than $o(p)$ to respect the aim of scalability.

5.2. Implemented views

The first clients implemented in our prototype are described in the following. We try to take advantage of the trace servers capabilities at the lowest level, duplication of views, filtering, synchronization, etc...

5.2.1. The nodes load

This view is constituted of windows representing the load of the node with a classical perf-meter. There can be as many window as the screen can display.

5.2.2. Animation window

This window provides, a study of the communication under replay conditions. But here we use the power of the trace servers approach. The events are requested by groups corresponding to a time interval. Each time a user or the automatic replay scrolls the time to an event which is not in the known interval, the view asks for an interval centered