Online Remote Trace Analysis of Parallel Applications on High-Performance Clusters

Holger Brunst[1,2], Allen D. Malony[1], Sameer S. Shende[1], and Robert Bell[1]

[1] Department for Computer and Information Science
University of Oregon, Eugene, USA
{brunst, malony, sameer, bertie}@cs.uoregon.edu
[2] Center for High Performance Computing
Dresden University of Technology, Germany
brunst@zhr.tu-dresden.de

Abstract. The paper presents the design and development of an online remote trace measurement and analysis system. The work combines the strengths of the TAU performance system with that of the VNG distributed parallel trace analyzer. Issues associated with online tracing are discussed and the problems encountered in system implementation are analyzed in detail. Our approach should port well to parallel platforms. Future work includes testing the performance of the system on large-scale machines.
Keywords: Parallel Computing, Performance Analysis, Performance Steering, Tracing, Clusters

1 Introduction

The use of clusters for high-performance computing has grown in relevance in recent years [5], both in terms of the number of platforms available and the domains of application. This is mainly due to two factors: 1) the cost attractiveness of clusters developed from commercial off-the-shelf (COTS) components and 2) the ability to achieve good performance in many parallel computing problems. While performance is often considered from a hardware perspective, where the number of processors, their clock rate, the network bandwidth, etc. determines performance potential, maximizing the price/performance ratio of cluster systems more often requires detailed analysis of parallel software performance. Indeed, experiences shows that the design and tuning of parallel software for performance efficiency is at least as important as the hardware components in delivering high "bang for the buck."

However, despite the fact that the brain is humankind's most remarkable example of parallel processing, designing parallel software that best optimizes available computing resources remains an intellectual challenge. The iterative process of verification, analysis, and tuning of parallel codes is, in most cases, mandatory. Certainly, the goal of parallel performance research has been to provide application developers with the necessary tools to do these jobs well. For example, the ability of the visual cortex to interpret complex information through pattern recognition has found high value in the use of trace analysis and visualization for detailed performance verification. Unfortunately, there is a major disadvantage of a trace-based approach – the amount of trace

data generated even for small application runs on a few processors can become quite large quite fast. In general, we find that building tools to understand parallel program performance is non-trivial and itself involves engineering tradeoffs. Indeed, the dominant tension that arises in parallel performance research is that involving choices of the need for performance detail and the insight it offers versus the cost of obtaining performance data and the intrusion it may incur.

Thus, performance tool researchers are constantly pushing the boundaries of how measurement, analysis, and visualization techniques are developed to solve performance problems. In this paper, we consider such a case – how to overcome the inherent problems of data size in parallel tracing while providing online analysis utility for programs with long execution times and large numbers of processes. We propose an integrated online performance analysis framework for clusters that addresses the following major goals:

- Allow online insight into a running application.
- Keep performance data on the parallel platform.
- Provide fast graphical remote access for application developers.
- Discard uninteresting data interactively.

Our approach brings together two leading performance measurement and analysis tools: the TAU performance system [4], developed at University of Oregon, and the VNG prototype for parallel trace analysis, developed at Dresden University of Technology in Germany.

The sections below present our system architecture design and discuss in detail the implementation we produced for our in-house 32-node Linux cluster. We first describe the TAU performance system and discuss issues that affect its ability to produce traces for online analysis. The section following introduces VNG and describes its functional design. The integrated system combining TAU and VNG is then presented. The paper concludes with a discussion of the portability and scalability of our online trace analysis framework as part of a parallel performance toolkit.

2 Trace Instrumentation and Measurement (TAU)

An online trace analysis system for parallel applications depends on two important components: a parallel performance measurement system and a parallel trace analysis tool. The TAU performance system is a toolkit for performance instrumentation, measurement and analysis of parallel programs. It targets a general computational model consisting of shared-memory compute nodes where multiple contexts reside, each providing a virtual address space shared by multiple threads of execution. TAU supports a flexible instrumentation model that applies at different stages of program compilation and execution [3]. The instrumentation targets multiple code points, provides for mapping of low-level execution events to higher-level performance abstractions, and works with multi-threaded and message passing parallel computation models. Instrumentation code makes calls to the TAU measurement API. The measurement library implements performance profiling and tracing support for performance events occurring at function,

method, basic block, and statement levels during program execution. Performance experiments can be composed from different measurement modules (e.g., hardware performance monitors) and measurements can be collected with respect to user-defined performance groups. The TAU data analysis and presentation utilities offer text-based and graphical tools to visualize the performance data as well as bridges to third-party software, such as Vampir [2] for sophisticated trace analysis and visualization.

2.1 TAU Trace Measurement

TAU's measurement library treats events from different instrumentation layers in a uniform manner. TAU supports message communication events, timer entry and exit events, and user-defined events. Timer entry and exit events can apply to a group of statements as well as routines. User-defined events are specific to the application and can measure entry/exit actions as well as atomic actions. TAU can generate profiles by aggregating performance statistics such as exclusive and inclusive times spent in routines, number of calls, etc. These profiles are written to files when the application terminates. With the same instrumentation, TAU can generate event traces, which are time-ordered event logs. These logs contain fixed-size trace records. Each trace record is a tuple comprising of an event identifier, a timestamp, location information (node, thread) and event specific parameters. The trace records are periodically flushed to disk when private in-memory buffers overflow, or when the application executes a call to explicitly flush the buffers. TAU also writes event description files that map event identifiers to event properties (such as event names, group names, tags, and event parameter descriptions).

2.2 Dynamic Event Registration

For runtime trace reading and analysis, it is important to understand what takes place when TAU records performance events in traces. The first time an event takes place in a process, it registers its properties with the TAU measurement library. Each event has an identifier associated with it. These identifiers are generated dynamically at runtime as the application executes, allowing TAU to track only those events that take actually occur. This is in contrast to static schemes that must predefine all possible events that could possibly occur. The main issue here is how the event identifiers are determined. In a static scheme, event IDs are drawn from a pre-determined global space of IDs, which restricts the scope of performance measurement scenarios. In our more general and dynamic scheme, the event identifiers are generated on-the-fly, local to a context. Depending on the order in which events first occur, the IDs may be different for the same event (i.e., events with the same name) across contexts. When event streams are later merged, these local event identifiers are mapped to a global identifier based on the event name. The TAU trace merging operation is discussed in section 4.3.

3 Remote Trace Data Analysis (VNG)

The distributed parallel performance analysis architecture applied in this paper has been recently designed by researchers from the Dresden University of Technology in Dres-

den, Germany. Based on the experience gained from the development of the performance analysis tool Vampir [1], the new architecture uses a distributed approach consisting of a parallel analysis server running on a segment of a parallel production environment, and a visualization client running on a remote graphics workstation. Both components interact with each other over a socket based network connection. In the discussion that follows, the parallel analysis server together with the visualization client will be referred to as *VNG*. The major goals of the distributed parallel approach are:

1. Keep performance data close to the location where they were created.
2. Analyze event data in parallel to achieve increased scalability.
3. Provide fast and easy remote performance analysis on end-user platforms.

VNG consists of two major components: an analysis server (`vngd`) and visualization client (`vng`). Each is supposed to run on a different platform. Figure 1 shows a high-level view of the VNG architecture. Boxes represent modules of the components whereas arrows indicate the interfaces between the different modules. The thickness of the arrows gives a rough measure of the data volume to be transferred over an interface, whereas the length of an arrow represents the expected latency for that particular link.

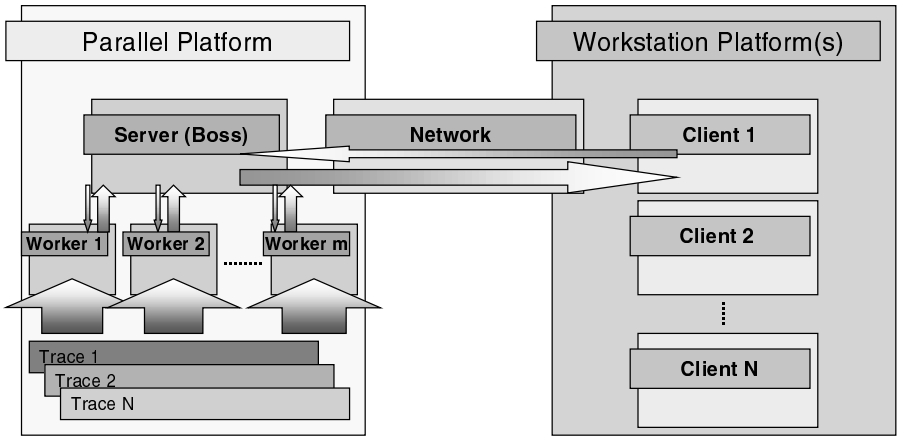

Fig. 1. VNG Architecture

On the left hand side of Figure 1 we can see the analysis server, which is intended to execute on a dedicated segment of a parallel machine. The reason for this is two-fold. First, it allows the analysis server to have closer access to the trace data generated by an application being traced. Second, it allows the server to execute in parallel. Indeed, the server is a heterogeneous parallel program, implemented using MPI and pthreads, which consists of worker and boss processes. The workers are responsible for trace data storage and analysis. Each of them holds a part of the overall trace data to be analyzed. The bosses are responsible for the communication to the remote clients. They decide

how to distribute analysis requests among the workers. Once the analysis requests are completed, the bosses also merge the results into a single packaged response that is to be sent to the client.

The right hand side of Figure 1 depicts the visualization client(s) running on a local desktop graphics workstation. The idea is that the client is not supposed to do any time consuming calculations. It is a straightforward sequential GUI implementation with a look-and-feel very similar to performance analysis tools like Vampir and Jumpshot [6]. For visualization purposes, it communicates with the analysis server according to the user's preferences and inputs. Multiple clients can connect to the analysis server at the same time, allowing simultaneous distributed viewing of trace results.

4 Enabling Online Trace Analysis

Up to now, TAU and VNG could only provide post-mortem trace analysis. The inherent scalability limitations of this approach due to trace size motivated us to evaluate the extensions and modifications needed to enable online access to trace data. The following discusses the technical details of our proposed online tracing solution.

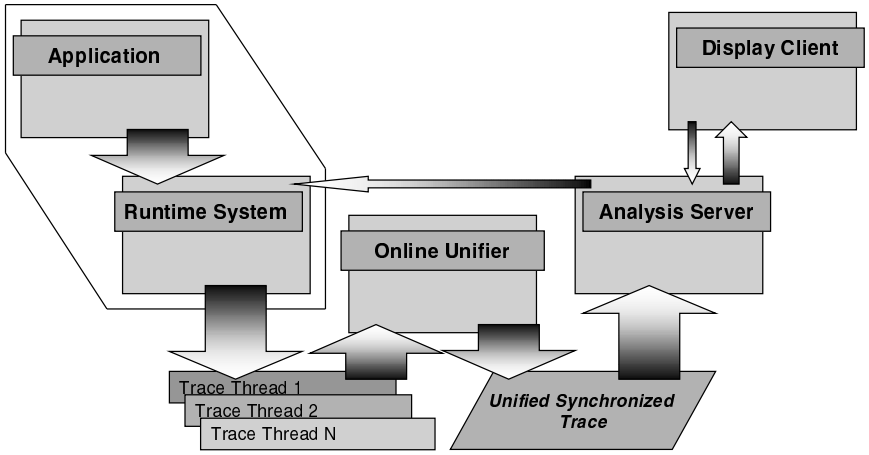

Fig. 2. Online Analysis Architecture

4.1 Overview

Figure 2 depicts the overall architecture of our online trace analysis framework. On the far left we see a running application instrumented by TAU. The inserted probes call the TAU measurement library which is responsible for the trace event generation. Periodically, depending on external or internal conditions (see section 4.2 for details), each application context will dump its event trace data. In our present implementation,

this is done to disk via NFS on a dedicated, high-speed network separate from the MPI message passing infrastructure. An independent trace writing network is a nice feature of our cluster that reduces tracing intrusion on the application.

Concurrently, an independent TAU process runs on a dedicated node and merges the parallel traces streams. Its responsibility is to produce a single globally-consistent trace with synchronized timestamps. This stream is then passed to the VNG analysis server which is intended to run on a small subset of dedicated cluster nodes. From there, pre-calculated performance profiles and event timelines are sent to the visualization client running on the user's local computer. This multi-layer approach allows to event data to be processed online and in parallel, close to its point of origin on the cluster. Furthermore, the interactive access to the data, coupled with runtime instrumentation and measurement control, allows the detailed analysis of long production runs.

4.2 Triggers for Trace Dumping

The interactive character of online trace analysis requires a strategy for triggering the TAU runtime system to dump trace data. We considered the four different options:

1. **Buffer size driven:** The tracing library itself makes the decision when to store trace data. One approach is to dump data whenever the internal memory buffers are full. It is easy to implement and does not require any external interaction with the measurement library. Unfortunately, it may produce large amounts of data when not combined with any filtering techniques. Also, the different contexts (i.e., processes) will produce the trace data at different times, depending on when their buffers fill up.
2. **Application driven:** The tracing library could provide the application with an API to explicitly dump the current trace buffers. (Such an API is already available in TAU.) The advantage of this approach is that the amount and frequency of trace dumping is entirely under the control of the application. We expect this approach to be desired in many scenarios.
3. **Timer driven:** To make triggering more periodic, a timer could be used to generate an interrupt in each context that causes the trace buffers to be dumped, regardless of the current amount of trace data they hold. In theory, this is simple to implement. Unfortunately, there is no general, widely portable solution to interrupt handling on parallel platforms.
4. **User driven:** Here the user decides interactively when to trigger the dump process. Assuming the user is sitting in front of a remote visualization client (e.g., `vng`), the trigger information needs to be transported to the cluster and to the running application (i.e., the trace measurement system). Again, this requires some sort of inter-process signaling. From the options we discussed so far, we regard this approach to be the most challenging, but also the most desirable.

For the work presented here, we implemented the buffer size and application driven triggering mechanisms. These are generally termed "push" models, since they use internal decision strategies to push trace data out to the merging and analysis processes. In contrast, the "pull" models based on timer or user driven approaches require some

form of inter-process signalling support. Our plan was to first use the simpler push approaches to validate the full online tracing system before implementing additional pull mechanisms.

4.3 Background Merging and Preparation

Previously, TAU wrote the event description files to disk when the application terminated. While this scheme was sufficient for post-mortem merging and conversion of event traces, it could not be directly applied for online analysis of event traces. This was due to the absence of event names that are needed for local to global event identifier conversion. To overcome this limitation, we have re-designed our trace merging tool, `TAUmerge`, so it executes concurrently with the executing application generating the trace files. From each process's trace file, `TAUmerge` reads event records and examines their globally synchronized timestamps to determine which event is to be recorded next in the ordered output trace file. When it encounters a local event identifier that it has not seen before, it reads the event definition file associated with the given process and updates its internal tables to map that local event identifier to a global event identifier using its event name as a key. The trace generation library ensures that event tables are written to disk before writing trace records that contain one or more new events. A new event is defined as an event whose properties are not recorded in the event description file written previously by the application. This scheme, of writing event definitions prior to trace records, is also used by the `TAUmerge` tool while writing a merged stream of events and event definitions. It ensures that the trace analysis tools down the line that read the merged traces also read the global event definitions and refresh their internal tables when they encounter an event for which event definitions are not known.

4.4 Trace Reader Library

To make the trace data available for runtime analysis, we implemented the TAU trace reader library. It can parse binary merged or unmerged traces (and their respective event definition files) and provides this information to an analysis tool using a trace analysis API. This API employs a callback mechanism where the tool registers callback handlers for different events. The library parses the trace and event description files and notifies the tool of events that it is interested in, by invoking the appropriate handlers with event specific parameters. We currently support callbacks for finding the following:

- Clock period used in the trace
- Message send or receive events
- Mapping event identifiers to their state or event properties
- Defining a group identifier and associated group name
- Entering and leaving a state

Each of these callback routines have event specific parameters. For instance, a send event handler has source and destination process identifiers, the message length, and its tag as its parameters. Besides reading a group of records from the trace file, our API supports file management routines for opening, closing a trace file, for navigating

the trace file by moving the location of the current file pointer to an absolute or relative event position. It supports both positive and negative event offsets. This allows the analysis tool to read, for instance, the last 10000 events from the tail of the event stream. The trace reader library is used by VNG to analyze the merged binary event stream generated by an application instrumented with TAU.

4.5 Remote Steering and Visualization

Online trace analysis does not necessarily require user control if one looks at it as a uni-directional information stream that can be continuously delivered to an application developer. However, the high rate at which information is likely to be generated in a native tracing approach suggests the need for advanced user control. Ideally, the user should be able to retrieve new trace data whenever he thinks it is necessary. At the VNG display client, this could be accomplished by adding a "retrieve" option that allows the user to specify how many recent events are to be loaded and analyzed by the analysis server. Such a request could translate all the way to trace generation. Even more sophisticated online control of the runtime system by the application developer would include:

- Enabling and disabling of certain processes
- Filtering of unwanted events
- Switching between a profile only mode and a detailed trace mode

While our infrastructure was designed to provide the necessary communication methods for those control requests, our present implementation allows for the trace analyzer to only work with whatever trace data is presently available from the TAU trace reader interface. This is not a limitation for our current purposes of validating the system implementation.

4.6 System Partitioning

Figure 3 depicts how our cluster system might be partitioned for online trace analysis. Together with the master node, we dedicated four additional processors (29-32) for the purpose of trace data processing. Parallel applications are run on the rest of the machine (processors 01 to 28). The cluster nodes are interconnected by a redundant gigabit Ethernet network each with its own network switch. The two networks allow us to keep message passing communication independent from NFS traffic, although our approach does not require this explicitly.

During an application run, every worker process is equipped with an instance of the TAU runtime library. This library is responsible for collecting trace data and writing it to a temporary file, one per worker process. The TAU merger process on processor 32 constantly collects and merges the independent streams from the temporary files to a unified representation of the trace data. At this point, the analysis process of VNG takes over. Whenever a remote user client connects to the server on the master node, the VNG worker processes come into action on the processors 29 to 31 in order to update their data or perform new analysis requests. During the entire process, trace data stays put in

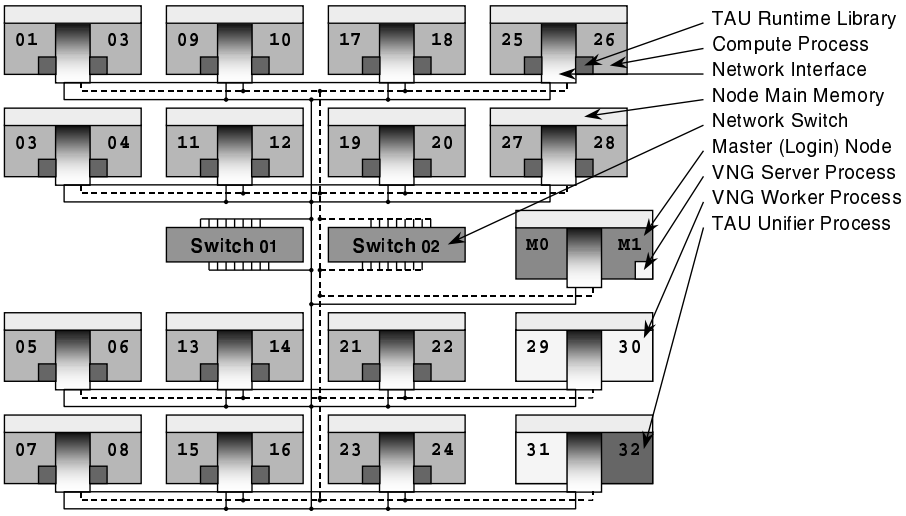

Fig. 3. System Partitioning

the cluster file system. This has tremendous benefits in GUI responsiveness at the client. Furthermore, the VNG analyzer communicates with the remote visualization client in a highly optimized fashion, guaranteeing fast response time even when run over a long distance connection with low bandwidth and high latencies (see Results).

5 Conclusion

There is growing interest in monitoring parallel applications and interacting with the running program as the computation proceeds. Some efforts are focussed on computational steering and support the creation of sensors to observe execution dynamics and actuators to change program behavior. Consistent with these directions, our attention is towards online performance evaluation using tracing as a measurement approach. The purpose is to offer the user the same rich functionality as off-line trace analysis, but without the penalties of large trace data management.

However, the development of a general-purpose online trace analysis system is difficult, especially if it is to be portable and scalable. The work presented here is a first step toward this goal. Combining the strengths of the TAU and VNG tools, we demonstrated a full-path, working system that allows interactive trace generation, merging, analysis, and visualization. In its present form, the work is quite portable across parallel platforms, as it is based on already portable existing tools and the file system and inter-process communication interfaces used are standard.

Our next step in this research is to conduct scalability performance tests. We expect the file system-based trace merging approach will suffer at some point. To address this problem at higher levels of parallelism, we are considering the use of the recent work

on MRNet, a multi-cast reduction network infrastructure for tools, to implement a tree-based parallel version of the TAU merger.

References

1. Brunst, H., Nagel, W. E., Hoppe, H.-C.: Group-Based Peformance Analysis for Multithreaded SMP Cluster Applications. In: Sakellariou, R., Keane, J., Gurd, J., Freeman, L. (eds.): Euro-Par 2001 Parallel Processing, No. 2150 in LNCS, Springer, (2001) 148–153
2. Brunst, H., Winkler, M., Nagel, W. E., Hoppe H.-C.: Performance Optimization for Large Scale Computing: The Scalable VAMPIR Approach. In: Alexandrov, V. N., Dongarra, J. J., Juliano, B. A., Renner, R. S., Kenneth Tan, C. J. (eds.): Computational Science – ICCS 2001, Part II, No. 2074 in LNCS, Springer, (2001) 751–760
3. Lindlan, K.A., Cuny, J., Malony, A.D., Shende, S., Mohr, B., Rivenburgh, R., Rasmussen, C.: Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates. Proceedings SC'2000, (2000)
4. Malony, A., Shende, S.: Performance Technology for Complex Parallel and Distributed Systems. In: Kotsis, G., Kacsuk, P. (eds.): Distributed and Parallel Systems From Instruction Parallelism to Cluster Computing. Proc. 3rd Workshop on Distributed and Parallel Systems, DAPSYS 2000, Kluwer (2000) 37–46
5. Meuer, H.W., Strohmaier, E., Dongarra J.J., Simon H.D.: Top500 Supercomputing Sites, 18th Edition, (2002) http://www.top500.org
6. Zaki, O., Lusk, E., Gropp, W., Swider, D.: Toward Scalable Performance Visualization with Jumpshot. The International Journal of High Performance Computing Applications 13(3) (1999) 277–288