A Community Databank for Performance Tracefiles

Ken Ferschweiler[1], Mariacarla Calzarossa[2], Cherri Pancake[1],
Daniele Tessera[2] and Dylan Keon[1]

[1]Northwest Alliance for Computational Science & Engineering, Oregon State University
[2]Dipartimento di Informatica e Sistemistica, Università di Pavia

Tracefiles provide a convenient record of the behavior of HPC programs, but are not generally archived because of their storage requirements. This has hindered the developers of performance analysis tools, who must create their own tracefile collections in order to test tool functionality and usability. This paper describes a shared databank where members of the HPC community can deposit tracefiles for use in studying the performance characteristics of HPC platforms as well as in tool development activities. We describe how the Tracefile Testbed was designed and implemented to facilitate flexible searching and retrieval of tracefiles. A Web-based interface provides a convenient mechanism for browsing and downloading collections of tracefiles and tracefile segments based on a variety of characteristics. The paper discusses the key implementation challenges.

1 The Tracefile Testbed

Tracefiles are a valuable source of information about the properties and behavior both of applications and of the systems on which they are executed. They are typically generated by the application programmer as part of the performance tuning process. Our field studies of HPC programmers indicate that many experienced programmers also create suites of simple pseudo-benchmark codes and generate tracefiles to help establish basic performance characteristics when they move to new HPC platforms. The intent in both cases is to help the user better understand and tune his/her applications.

The developers of trace-based performance analysis and performance prediction tools (cf. [7, 8, 10, 9, 3]) also generate suites of tracefiles. In this case, the objective is to assist in the process of testing and fine-tuning tool functionality. According to the subjects interviewed in our field studies, tool developers do not often have access to "real" applications for these activities; rather, they construct artificial codes designed to generate tracefiles that will stress the tool's boundary conditions or generate demonstration visualizations.

Tool users and developers alike have indicated in several public forums (e.g., Parallel Tools Consortium meetings, BOF sessions at the SC conference, community workshops on parallel debugging and performance tuning tools) that it would be useful to construct a generally accessible testbed for tracefile data. This would make it possible for users to see if tracefiles from related applications can be of use in the design and tuning of their own application. It would also provide a more realistic foundation for testing new performance tools. Further, since tracefiles are typically large and unwieldy to store (the recording of key program events during one application run can generate

gigabytes of data), a centralized repository could encourage programmers to archive their tracefiles rather than deleting them when they are no longer of immediate use.

In response to this need, we created the Tracefile Testbed. The objective was to develop a database that not only supports convenient and flexible searching of tracefile data generated on HPC systems, but maximizes the benefit to others of performance data that was collected by a particular programmer or tool developer for his/her own purposes.

The Tracefile Testbed was implemented as a joint project of NACSE and the Università di Pavia. The work was supported by the NAVO (Naval Oceanographic Office) MSRC, through the PET program of the High Performance Computing Modernization Office, and will be maintained with support of the Parallel Tools Consortium. It was structured according to a data model that describes both the static and dynamic behavior of parallel applications, as captured in tracefiles. The tracefiles are maintained as separate file units. The source code that generated the tracefiles is also available (unless that code is proprietary). Metadata encapsulating the performance behavior and run-time environment characteristics associated with the tracefiles are maintained in a relational database using Oracle 8i.

A key aspect of tracefile storage is their size. This can pose difficulties for prospective users, who may find that storing many downloaded copies is quite resource-intensive. To reduce that burden, all file locations are maintained in the Tracefile Testbed's metadata database as URLs. This will allow users to "maintain" their own subsets of tracefiles by simply storing links or shortcuts to the files, rather than the files themselves. A secondary advantage of this approach is that it allows us to distribute the repository itself. That is, the actual tracefiles may be located on multiple servers, which can be different from the server(s) hosting the tool interface and the metadata database. The initial implementation involves three servers: a Web server maintains the interface, a relational database server hosts the metadata, and the tracefiles are stored on a separate file server. This architecture is illustrated in Figure 1.

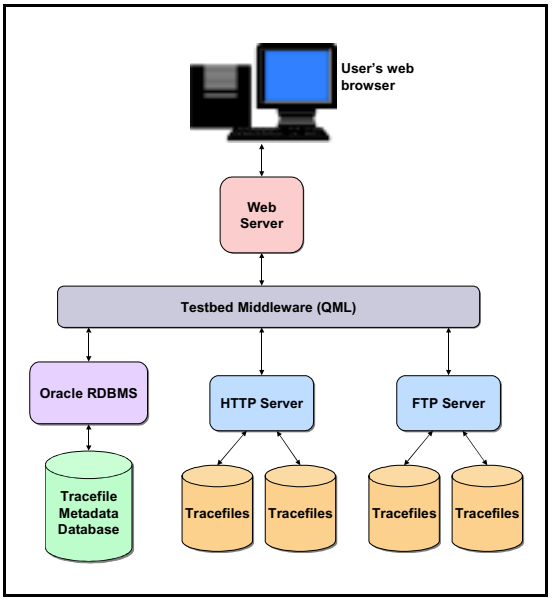

Figure 1. Architecture of the Tracefile Testbed

A Web-based interface allows users to navigate through the repository, select tracefiles and segments from one or more applications, browse their characteristics, and download the data. The interface makes use of QML (Query Markup Language), a middleware product developed and distributed by NACSE. Performance data can be identified and extracted based on various selection criteria, such as

"all data related to a given application," "data related to a class of applications," "data from programs executed on a particular system architecture," etc. The Tracefile Testbed provides performance summaries of selected trace data; alternatively, the tracefile data may be downloaded for analysis using available tools in order to derive detailed performance figures.

2 Data Model

In order to categorize and maintain tracefile data, we require a data model with the power to describe the characteristics of parallel applications and the performance measurements collected during execution. In large part, the framework we have chosen to describe tracefiles is based on user needs in searching the tracefile collection. Based on previous usability studies, we determined that users wish to select entire tracefiles or segments thereof, on the basis of machine architecture types and parameters, information related to the tracefile itself, and information related to the tracefile segments. Users should also be able to perform searches based on arbitrary keywords reflecting system platforms, problem types, and user-defined events.

2.1 Structure of the Data Model

The model must capture not just parallel machine characteristics, but also the design strategies and implementation details of the application. For this purpose, the information describing a parallel application has been grouped into three layers:

The *system layer* provides a coarse-grained description of the parallel machine on which the application is executed. The other two layers comprise information derived from the application itself; the *application layer* describes its static characteristics, whereas the *execution layer* deals with the dynamic characteristics directly related to measurements collected at run time. Most of the information comprising the system and application layers is not available in the tracefile, but must be supplied by the application programmer in the form of metadata. Execution layer information can be harvested directly from the tracefiles.

The system layer description includes machine architecture (e.g., shared memory, virtual shared memory, distributed memory, cluster of SMPs), number of processors, clock frequency, amount of physical memory, cache size, communication subsystem, I/O subsystem, communication and numeric libraries, and parallelization tools.

The static characteristics of the application layer range from the disciplinary domain (e.g., computational fluid dynamics, weather forecasting, simulation of physical and chemical phenomena) to the algorithms (e.g., partial differential equation solvers, spectral methods, Monte Carlo simulations) and programming languages employed. They also include information about the application program interface (e.g., MPI, OpenMP, PVM) and links to the source code. Problem size, number of allocated processors, and work and data distributions are further examples of static characteristics.

The execution layer provides a description of the behavior of a parallel application in terms of measurements generated at run time. These measurements are typically timestamped descriptions which correspond to specific events (I/O operation, cache miss, page fault, etc.) or to instrumentation of the source code (e.g., beginning or end of an arbitrary section of code, such as a subroutine or loop). The type and number of measurements associated with each event depend on the event type and on the monitoring meth-

ods used to collect the measurements. Application behavior might be described by the time to execute a particular program section or the number of events recorded in a particular time span.

2.2 Describing Tracefile Content

To maintain the system, application, and execution information describing the tracefile repository, we implemented a database of descriptive metadata. These exist at multiple levels: they include descriptions of individual tracefiles, sets of tracefiles, and segments of tracefiles. The use of off-the-shelf rDBMS software allows us to maintain and search these metadata with a great deal of power, flexibility, and robustness, and with a minimum of investment in software development.

As discussed previously, the choice of which metadata to maintain – the data model – was based on our assessment of user needs in searching the tracefile collection. The Tracefile Testbed provides the ability to search on machine, application, or execution parameters. The versatility of the database allows us to search based on flexible combinations of these parameters, but careful database design was required to make full use of the power of the rDBMS. Figure 2 presents a conceptual view of the database schema supporting user searches.

Note that tracefiles do not typically stand alone; they are usually generated in sets of related files pertaining to a larger project, or *experiment*. The metadata database allows us to maintain this information about the origin of tracefiles. In other cases, a number of tracefiles that were not generated together may still form a naturally cohesive

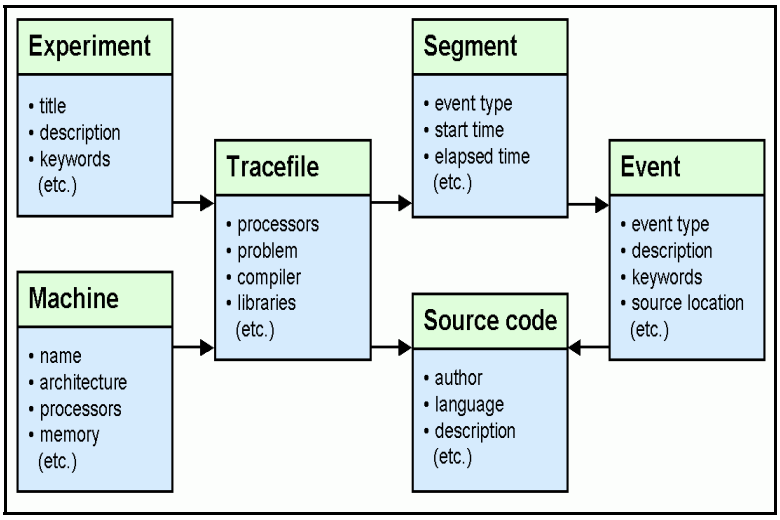

Figure 2. General structure of tracefile metadata

set (e.g., they may demonstrate a common computational approach, or illustrate the effects of varying a particular parameter). Since cohesion of such sets would not always be apparent from the metadata described above, the system allows specification of *virtual experiments* – groups of tracefiles which, though not related in origin, have an *ex post*

facto relationship which is useful to some researcher. This structure allows tracefiles to belong to multiple sets which cut across each other, allowing individual users to superimpose organizational schemes which fit their particular needs.

A key requirement for the Tracefile Testbed is that it be easy for members of the HPC community to add new tracefiles. We were fortunate in having access to a sizeable collection of files from a variety of machine and problem types to use as the initial population of the repository. We gathered almost 200 files in our benchmarking work with the SPEC suite [1]. Given the number of files we anticipate gathering from the APART (Automated Performance Analysis: Resources and Tools) working group and other members of the HPC community, it was important to be able to parse the files in batch mode, and our initial parser reflects this bias. A Web-based tool for uploading tracefiles will be implemented in the next phase of the project.

To ensure that metadata are available for all tracefiles in the Testbed, they must be supplied as part of the uploading mechanism. As discussed previously, information such as system- and application-level metadata does not exist *a priori* in the tracefiles, but must be provided by the programmer or benchmarker. The originator of the tracefiles is also the source of descriptive information about user-defined events in the execution-level metadata. To facilitate the input of that information, we developed a tracefile metadata format and a corresponding parser. Most of the metadata elements are likely applicable to a whole series of tracefiles, so the format and uploading tool were designed to facilitate metadata reuse.

3 Tracefile Segments

While tracefiles are typically quite large, the portion of a tracefile which is of interest for a particular purpose may be only a small fragment of the file. For instance, a researcher wishing to compare the performance of FFT implementations may want to work with a fragment that brackets the routine(s) in which the FFT is implemented. Similarly, a tool developer may be interested in testing tool functionality in the presence of broadcast operations; the remainder of the trace may be largely irrelevant. If the source code is appropriately instrumented at the time of tracefile creation, the sections of interest will be easily identifiable, but locating them in a large corpus of tracefile data may still be an onerous task. In order to simplify identification of tracefile fragments which are of interest, it is convenient to maintain a description of the internal structure of tracefiles. Some of this structure may be automatically generated from information in the tracefile, but the remainder must be supplied as metadata, typically by the programmer who contributes the file to the repository.

3.1 Dividing Tracefiles into Segments

Since a tracefile is essentially a list of timestamped events (with some descriptive header information), it is easy to identify a subset of a tracefile corresponding to the events occurring during a particular time interval. The obvious choices for defining the interval are the begin and end timestamps of a user-defined event (such as the FFT routine mentioned above). We discuss user-defined events because system-defined events are typically atomic; that is, they do not have start and end markers. However, such a view may be an oversimplification that does not capture the behavior of interest during the time interval. Since the tracefile is a straightforward list of per-processor events, it

is considerably more difficult to define events which pertain to the entire parallel machine. The idealized view of a data-parallel application would have all processors participating in all events (i.e., executing the same segment of code) approximately simultaneously; however, there is no guarantee in an actual application that any event will include all processors, simultaneously or not.

Consequently, a user who wishes to extract a subset of a tracefile to capture system performance during a particular event is faced with a difficulty. Although the user may know that particular events on one processor correspond to events on other processors, it is not clear from the tracefile how these correspondences can be automatically inferred. We have used a heuristic approach to identifying machine-wide events. A *machine-wide event* includes all of the same-type per-processor events whose starting markers in the tracefile are separated by fewer than K*N events, where N is the number of processors in the machine, and K is a definable constant (currently set to 4). The per-processor events which comprise a machine-wide event may or may not overlap in time, but discussion with users of parallel performance evaluation systems indicate that they expect this criterion to effectively capture the corresponding events.

The machine-wide event, defined as a starting timestamp (and, for user-defined events, an ending timestamp) in a particular tracefile, is the basic unit of tracefile data which our system maintains; we allow users to attach descriptions, keywords, and source code references to these events.

3.2 Using Tracefile Segments

To many HPC users, the principal reason for creating and maintaining tracefiles is to be able to use them as input to performance-analysis software. To support this requirement, the Tracefile Testbed provides single-keystroke operations for downloading tracefiles to the user's local machine via http or ftp.

The issue of tracefile segments introduces problems with respect to tool compatibility. Trace-based performance tools require “legal” tracefiles as input; while there is no single *de facto* standard for tracefile format, we assume that a tracefile which is usable by popular performance analysis packages will also be suitable for HPC users who write their own analysis tools. A fragment naively extracted from a tracefile will not, in general, be of a legal format. In particular, it will lack header information and will probably contain unmatched markers of entry to and exit from instrumented program regions. To make segments useful, the Tracefile Testbed modifies the fragment inorder to generate a legal tracefile which describes as closely as possible the behavior of the application in the region which the user has selected.

4 User Interface

Users faced with powerful software and complex interfaces quickly discover the features they need to solve their immediate problems and ignore the rest. We made a conscious decision to supply only those features which will be useful to a large community of users, allowing us to develop a concise and intuitive user interface. Our choice for user interface platform was the ubiquitous web browser, which offers near-universal portability. The interface was developed using an existing web-to-database middleware package, QML (Query Markup Language [5]. QML allowed us to quickly develop prototype implementations of interfaces exploring various search strategies and to support

a "drilling-down" style of search. It also supplies a builtin mechanism for downloading tracefiles or segments for further analysis.

Figure 3 shows an example of a tracefile query page. Selectable lists (machine types, application types, keywords, etc.) are generated dynamically from the metadata database contents; the user interface requires no updating to accommodate new additions to the database. The user may drill down through the layers supported in the data mode –system, application, and execution– refining his search at each layer. At any stage in the search, the options are limited to those still available based on the constraints which the user has supplied in earlier layers.

4.2 Performance Summaries

In many cases, the information a user wants from one or more tracefiles may be easily summarized without recourse to other performance analysis software. This is particularly the case when an application programmer wishes to compare some measure of "overall" performance across several different files. To simplify such tasks, the Tracefile Testbed provides a statistical performance summary functions which may be performed on selected tracefiles or segments.

Over the next year, we are scheduled to add features providing graphical summaries of application behavior. These will allow the user to compare tracefiles at a synopsis level before deciding to examine them in more detail.

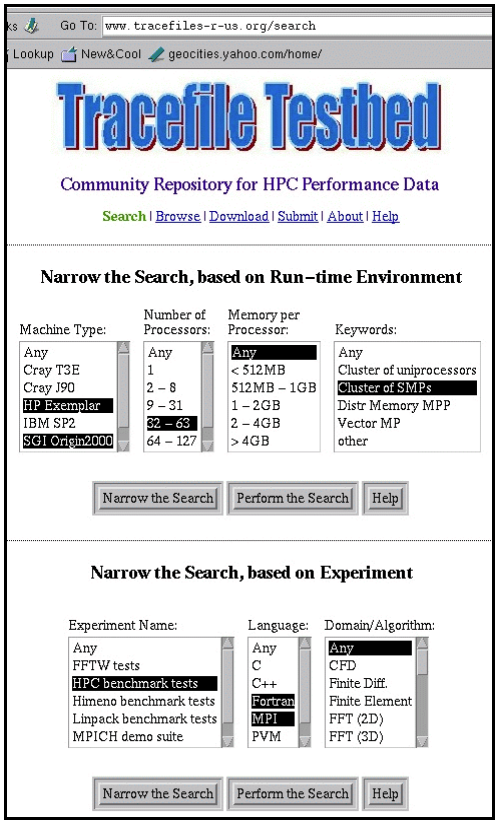

Figure 3. Tracefile Search Interface

5 Summary

Responding directly to a requirement that has been expressed in a variety of community forums, the Tracefile Testbed uses web-to-database technology to provide HPC programmers and tool developers with access to a repository of tracefiles. A database of metadata describing the systems, applications, and execution-level information of each tracefile supports a variety of search approaches. Performance summaries assist users to assess the relevance of files and segments before they are examined in detail. Individual files and/or segments may be downloaded to the user's local system for further analysis and comparison. Application programmers should find this community repository useful both in predicting the behavior of existing programs and in the development and optimization of new applications. Developers of

performance analysis and prediction tools will find the Tracefile Testbed to be a convenient source of tracefiles for testing the functionality and display capabilities of their tool.

6 Acknowledgments

Scott Harrah and Tom Lieuallen of Oregon State University, and Luisa Massari of the Università di Pavia contributed to the implementation of the Tracefile Testbed. The project was supported by the NAVO (Naval Oceanographic Office) MSRC, through the PET program of the HPC Modernization Office.

7 Bibliography

1. R. Eigenmann and S. Hassanzadeh. Benchmarking with Real Industrial Applications: The SPEC High-Performance Group. *IEEE Computational Science and Engineering,* Spring Issue, 1996.
2. T. Fahringer and A. Pozgaj. P3T+: A Performance Estimator for Distributed and Parallel Programs. *Journal of Scientific Programming,* 7(1), 2000.
3. B.P. Miller et al. The Paradyn Parallel Measurement Performance Tool. *IEEE Computer,* 28 (11):37-46, 1995.
4. K.L. Karavanic and B.P. Miller. Improving Online Performance Diagnosis by the Use of Historical Performance Data. In *Proc. SC'99,* 1999.
5. C. M. Pancake, M. Newsome and J. Hanus. 'Split Personalities' for Scientific Databases: Targeting Database Middleware and Interfaces to Specific Audiences. *Future Generation Computing Systems*, 6: 135-152, 1999.
6. S.E. Perl, W.E. Weihl, and B. Noble. Continuous Monitoring and Performance Specification. Technical Report 153, Digital Systems Research Center, June 1998.
7. D.A. Reed et al. Performance Analysis of Parallel Systems: Approaches and Open Problems. In *Joint Symposium on Parallel Processing,* pages 239-256, 1998.
8. S. Shende and A. Malony and J. Cuny and K. Lindlan and P. Beckman and S. Karmesin, Portable Profiling and Tracing for Parallel Scientific Applications using C++. In *Proc. SPDT'98: ACM SIGMETRICS Symposium on Parallel and Distributed Tools,* pages 134-145, 1998.
9. J. Yan, S. Sarukhai, and P. Mehra, "Performance Measurement, Visualization and Modeling of Parallel and Distributed Programs Using the AIMS Toolkit," *Software – Practice and Experience*, 25 (4): 429--461, 1995.
10. O. Zaki, E. Lusk, W. Gropp, and D. Swider. Toward Scalable Performance Visualization with Jumpshot. *The International Journal of High Performance Computing Applications,* 13(2):277-288, 1999.