The Tracefile Testbed: a community repository for identifying and retrieving HPC performance data

Ken Ferschweiler*
Northwest Alliance for Computational Science and Engineering,
218 CH2M Hill Alumni Center, Oregon State University,
Corvallis, 97331 OR, USA
Fax: 541 737 6609 E-mail: kennino@nacse.org
*Corresponding author

Scott Harrah
Weatherhead School of Management,
Case Western Reserve University,
10900 Euclid Avenue Cleveland, 44106 OH, USA
E-mail: scott_harrah@yahoo.com

Dylan Keon
Northwest Alliance for Computational Science and Engineering,
218 CH2M Hill Alumni Center, Oregon State University,
Corvallis, 97331 OR, USA
E-mail: keon@nacse.org

Mariacarla Calzarossa
Dipartimento di Informatica e Sistemistica,
Università di Pavia, via Ferrata, 1 I-27100 Pavia, Italy
Fax: +39 0382 985373 E-mail: mcc@unipv.it

Daniele Tessera
Dipartimento di Matematica e Fisica,
Università Cattolica del Sacro Cuore,
via Musei, 41 I-25121 Brescia, Italy
Fax: +39 030 2406 742 E-mail: d.tessera@dmf.unicatt.it

Cherri Pancake
School of Electrical Engineering and Computer Science,
102 Dearborn Hall, Oregon State University,
Corvallis, 97331 OR, USA
E-mail: pancake@eecs.orst.edu

Abstract: HPC programmers utilise tracefiles, which record program behaviour in great detail, as the basis for many performance analysis activities. The lack of generally accessible tracefiles has forced programmers to develop their own testbeds in order to study the basic performance characteristics of the platforms they use. Because tracefiles serve as input to performance analysis and performance prediction tools, tool developers have also been hindered by the lack of a testbed for verifying and fine-tuning tool functionality. A community repository that meets the needs of both application and tool developers has been created in this study. In this paper, we describe how the Tracefile Testbed was designed to facilitate flexible searching and retrieval of tracefiles based on a variety of characteristics has been described. Its web-based interface provides a convenient

mechanism for browsing, downloading, and uploading collections of tracefiles and tracefile segments, as well as viewing statistical summaries of performance characteristics.

Keywords: computer science; data communication; database management systems; high performance computing; performance tuning; performance monitoring; tracefiles.

Reference to this paper should be made as follows: Ferschweiler, K., Harrah, S., Keon, D., Calzarossa, M., Tessera, D. and Pancake, C. (2005) 'The Tracefile Testbed: a community repository for identifying and retrieving HPC performance data', *Int. J. High Performance Computing and Networking*, Vol. 3, Nos. 2/3, pp.95–102.

Biographical notes: Ken Ferschweiler is Technical Coordinator of the Northwest Alliance for Computational Science and Engineering at Oregon State University. He received his BS (Engineering) Degree from the University of Portland, Oregon in 1978. His research interests are in the area of providing scientists and engineers with creative and flexible ways of accessing computing facilities and exploring complex data.

Scott Harrah participated in this project while a graduate student at Oregon State University. After graduating with an MS in Computer Science in 2001, he was employed by Case Western Reserve University.

Dylan Keon is GIS Coordinator at the Northwest Alliance for Computational Science and Engineering at Oregon State University. He received an MS in Ecology and GIS/Statistics from Oregon State University in 2001. His research interests include the use of open source tools to develop web-based mapping interfaces that dynamically integrate spatial content from multiple sources, including relational databases.

Mariacarla Calzarossa is Professor of Computer Science at the Engineering School of the University of Pavia, Italy. She received a Laurea degree in Mathematics from the University of Pavia. She has been Visiting Scientist at the University of California at Berkeley and at Duke University. Her research interests center on performance evaluation of complex systems and applications, specifically addressing performance analysis and debugging of parallel applications and workload characterisation of mail servers and Web systems. She is a senior member of the IEEE and a member of the IFIP WG 7.3.

Daniele Tessera participated in this project while a post-doc at the Università di Pavia. Currently he is a Researcher at the Dipartimento di Matematica e Fisica, Università Cattolica at Brescia. His research interests include performance analysis and debugging, and workload characterisation of complex systems, such as parallel and distributed machines and internet distributed applications.

Cherri M. Pancake is Professor of Electrical Engineering and Computer Science at Oregon State University and Director of the Northwest Alliance for Computational Science & Engineering. She received a PhD in Computer Engineering from Auburn University. Her research interests center on usability engineering, specifically studying how complex software and data systems can be made more useful and effective for practicing scientists and engineers. She is a Fellow of the ACM and the IEEE.

1 BACKGROUND AND MOTIVATION

A high-performance computing (HPC) application is characterised by many variables that control its execution and determine its performance. Variables, such as algorithm type, problem size, input parameters, programming languages and paradigms, libraries, hardware architecture, etc., can have very significant effects on program behaviour. It is important to understand the role played by each variable and the ways they combine to influence the performance achieved, or achievable, by the application.

Two approaches are commonly used for the purpose of understanding these effects: performance profiling and performance prediction. Profiling (Reed et al., 1998; Shande et al., 1999) captures the behaviour of an application by monitoring its execution. Monitoring can be based on hardware counter sampling or it can require the instrumentation of the application's source code or its binary executable. The data produced by monitoring may be analysed on–the–fly or stored as tracefiles for post-mortem analysis. Many of the tools currently available for HPC performance analysis are based on tracefiles. Selected examples are described below.

- Jumpshot (Zaki et al., 1999) analyses tracefiles and provides multiple time-space diagrams of program behaviour.
- Continuous Monitoring (Perl et al., 1998) captures logs of appropriately instrumented applications, while they are being executed with the objective of automating the testing of performance properties of complex systems.

- Paradyn (Miller et al., 1995) employs historical performance data, gathered in previous executions of an application, to improve the effectiveness of automated performance diagnosis. The Performance Consultant component, within Paradyn, extracts knowledge from the performance data collected over the life of an application (Karavanic and Miller, 1999).

Performance prediction (Fahringer and Pozgaj, 2000) takes a different approach. These techniques attempt to provide estimates of the performance achievable by an application by analysing its structure and the influences of compiler transformations and the system architecture using symbolic analysis, simulation, or other model-based methods. Prediction tools often rely directly or indirectly on tracefiles. The data from tracefiles can serve as the basis for constructing or validating the performance model, or the data can be used directly by the tool to adjust the model to the characteristics of a particular application (e.g., Yan et al., 1995).

Tracefiles are typically generated by the application programmer as part of the performance tuning process. These field studies of HPC programmers indicate that many 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 intention in both cases is to help the user to understand and tune their applications better.

The developers of trace-based tools 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 here, 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 visualisations.

Tracefiles are a valuable source of information about the properties and behaviour both of applications and of the systems on which they are executed. Tool users and developers have indicated alike in several public forums (e.g., Parallel Tools Consortium meetings, Birds of a Feather (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. Furthermore, because tracefiles are typically large and unwieldy to store (the recording of key program events during one application run can generate gigabytes of data), a centralised repository could encourage programmers to archive their tracefiles, rather than deleting them when they are no longer of immediate use.

2 THE TRACEFILE TESTBED

With support from the Department of Defense (DOD) HPC Modernization Program, the creation of a community repository called the Tracefile Testbed was undertaken. The objective was to develop a database that not only supports convenient and flexible searching of tracefile data generated on HPC systems, but also to allow others to benefit from performance data that was collected by a programmer or tool developer for their own purposes.

The Tracefile Testbed was implemented as a joint project of NACSE and the Università di Pavia. It was structured according to a data model that describes both the static and dynamic behaviour 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 behaviour and run-time environment characteristics associated with the tracefiles are maintained in a relational database using Oracle9*i*.

File size is a key consideration when storing tracefiles. Although our organisation has committed itself to maintaining the repository as a contribution to the HPC community, size was also considered from the perspective of the users, who will find that storing many downloaded copies is quite resource-intensive. We accommodated this usability consideration in the following way. Within the Tracefile Testbed, all file locations are maintained in the metadata database as URLs. This allows users – if they choose – 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 that can be different from the server(s) hosting the tool interface and the metadata database. The initial implementation involves three servers: a web server maintaining the interface, a relational database server hosting the metadata, and the tracefiles that are stored on a separate file server. The general architecture of the Tracefile Testbed browser is illustrated in Figure 1.

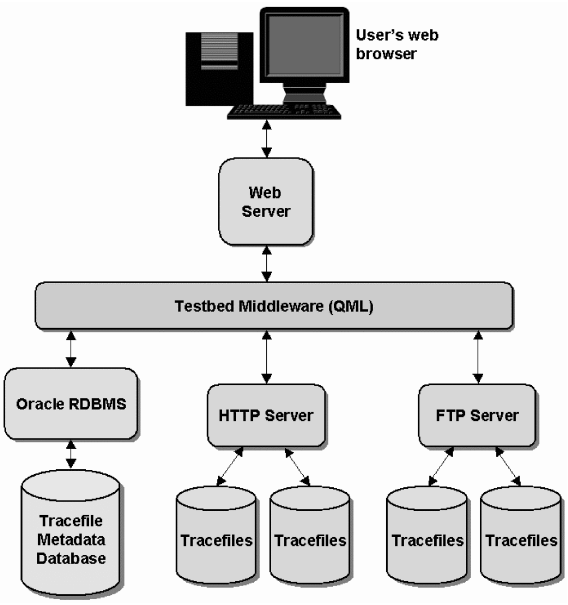


Figure 1 *Architecture of the Tracefile Testbed*

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. 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", "data from runs that performed global broadcast operations", 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.

There are several significant challenges to be addressed in creating a repository of this nature.

- How can we represent the characteristics of a parallel application and its associated tracefile(s) in such a way that testbed users can easily find and select appropriate performance data?
- How much metadata can be gleaned from the tracefiles themselves, rather than being supplied by the user submitting the files?
- How can tracefiles be subdivided into smaller segments to minimise the amount of data that must be downloaded for a particular purpose? What is the proper abstraction for those segments, given that we cannot guarantee that events on different processors occurred near-simultaneously (or even that they occurred at all)?
- How can we ensure that download operations always yield useful data? How can we reduce the need to download tracefiles? Can we allow users to maintain shortcuts to the appropriate files, without having to copy the files themselves?
- How can tracefile segments be structured, so that they can serve as input to trace-based tools when the user has not downloaded the complete file?
- Can the repository reduce the need for programmers to write simple analysis routines? Is there a way to provide a snapshot view that compares the performance recorded in multiple tracefiles?
- How can the effort required to enter metadata be minimised in order to encourage fully annotated submissions?
- What mechanisms for searching, selecting, and browsing tracefile data are powerful and flexible enough to help programmers understand application behaviour?
- Are the same mechanisms appropriate for use by tool developers? If not, what type of specialised support is required?
- To what extent can the user interface guide the user through the repository, so that totally unfamiliar users can quickly arrive at the most useful information?

Clearly, many of the issues are related to the usability of the repository, rather than structural aspects of the database itself. The sections below discuss how each issue was addressed in developing the Tracefile Testbed.

3 DATA MODEL

In order to categorise 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 their execution. In large part, the framework chosen to describe tracefiles was derived from user needs in searching the tracefile collection. Based on previous usability studies, it was 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.

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 parallelisation 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 behaviour of a parallel application in terms of measurements generated at run time. These measurements are typically

time-stamped descriptions that 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 methods used to collect the measurements. Application behaviour might be described by the time to execute a particular program section or the number of events recorded in a particular time span.

4 DESCRIBING TRACEFILE CONTENT

To maintain the system, application, and execution information describing the tracefile repository, a database of descriptive metadata is implemented. These metadata exist at multiple levels; they include descriptions of individual tracefiles, sets of tracefiles, and segments of tracefiles. The use of off-the-shelf relational database management (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 the 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.

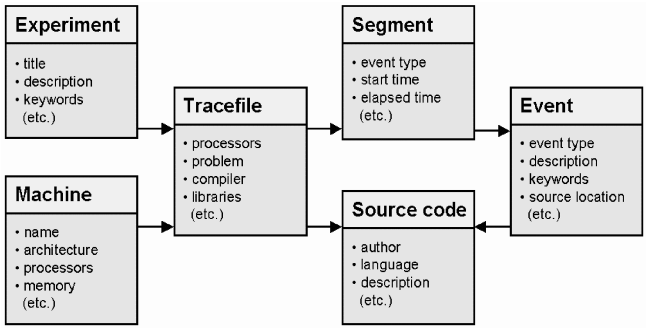


Figure 2 *General structure of tracefile metadata*

Note that tracefiles do not typically stand alone; they are usually generated as *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. The sets of tracefiles provide a convenient grouping mechanism, and allow users to view information on all tracefiles generated during a *physical experiment*, or suite of related executions. In other cases, a number of tracefiles that were not generated together may still form a naturally cohesive 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 that is useful to a researcher. This structure allows tracefiles to belong to multiple sets that cut across each other, allowing individual users to superimpose organisational schemes that 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 to the repository. We were fortunate in having access to a sizeable collection of tracefiles from a variety of machine and problem types to use as the initial population of the repository. We have gathered about 100 files over the last few years in our benchmarking work with the SPEC suite (Eigenmann and Hassanzadeh, 1996). 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 has also been implemented.

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 bench marker. 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, a tracefile metadata format and a corresponding parser have been developed. Most of the metadata elements are likely to be applicable to a whole series of tracefiles, so the format and uploading tools were designed to facilitate metadata reuse and to ease the task of uploading multiple tracefiles.

5 IDENTIFYING TRACEFILE EVENTS AND SEGMENTS

Although tracefiles are typically quite large, the portion of a tracefile that 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 that 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.

Because a tracefile is essentially a list of time-stamped 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 choice for defining such a time interval is the begin and end timestamp of a user-defined event (such as the FFT routine mentioned above). User-defined events are discussed because system-defined events in MPI are atomic; that is, they do not have start and end markers. However, such a view may be an oversimplification that does not capture the behaviour of interest during the time interval. Because the tracefile is a straightforward list of per-processor events, it is considerably more difficult to define events that pertain to the entire parallel machine. The idealised 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 identify machine-wide events. A machine-wide event includes all 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 that comprise a machine-wide event may, or may not, overlap in time, but in discussions with users of parallel performance evaluation systems it is found that the users 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 that our system maintains. We allow users to attach descriptions, keywords, and source code references to these events. Furthermore, it is possible to search, browse, and download just the portions of a tracefile that are of interest to a particular user. A *tracefile segment* is defined as the portion of the tracefile between where a machine-wide event begins and ends. A given tracefile may have thousands of segments; they can be accessed individually or in groups sharing some characteristic (e.g., all segments corresponding to global summation operations).

6 FORMING TRACEFILE SEGMENTS

The principal reason many HPC users create and maintain 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 conformant tracefiles as input; although there is no single standard for tracefile format, we assume that a tracefile that 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, in order to generate a legal tracefile that describes, as closely as possible, the behaviour of the application in the region that the user has selected.

7 PERFORMANCE SUMMARIES

In many cases, the information that a user wants from a tracefile or set of tracefiles may be easily summarised 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 tracefiles. To simplify such tasks, the Tracefile Testbed provides some simple performance summary functions that may be performed on selected sets of tracefiles or tracefile segments. Available summary functions are summarised below.

- mean and standard deviation of segment length (in elapsed time, in a particular set of tracefile segments)
- identification and length of the shortest and longest segments (in a particular set of tracefile segments)
- number of identifiable segments in a tracefile
- elapsed time of a tracefile
- per-processor mean and standard deviation of the elapsed time of a particular type of event (e.g., I/O operation, cache miss).

Processor utilisation during a parallel event (e.g., how much processor time is spent waiting during a barrier synchronisation) should also be taken into account.

8 THE USER INTERFACE

The user interface to the Tracefile Testbed was implemented using web technology to emphasise portability and convenience. Two interfaces were created: one for searching the testbed and downloading performance data, the other for uploading tracefiles and corresponding metadata. Perhaps the most important concern in the design of the interfaces was scalability to the potential size of the testbed. This study's goal was to enable users to search and locate

performance data in the most efficient manner possible and seamlessly download the appropriate files or segments.

The search and upload interfaces permit users to move freely among querying, downloading, uploading, and help activities. Definitions for all operations are available in pop-up windows activated when the mouse is positioned over instances of the term in the interface. Although the relational nature of the testbed would allow users to query and view the values of all data fields, previous experiences in usability of database interfaces (Newsome et al., 1999) indicated that presenting the user with so many choices at once would be confusing. Instead, a drill-down approach was applied, where users are presented with a subset of the selectable fields at each step, allowing the development of a comprehensive, yet concise and intuitive, user interface.

Throughout the interface, users have the option of returning to previous stages in their search by using the Return buttons. The advantage of providing these, rather than simply using the Back button provided by the web browser, is that we can provide a descriptive label for the button (e.g., Return to Tracefile Listing) so that users can know exactly as to which step in the search sequence they will be moved.

The search interface was developed using QML (Query Markup Language, http://www.nacse.org/qml), a web-to-database middleware package developed and distributed by NACSE. QML facilitates the dynamic generation of selectable lists by pre-fetching values from the testbed, meaning that the interface does not require updating to accommodate additions to the database. The initial query screen is displayed in Figure 3.

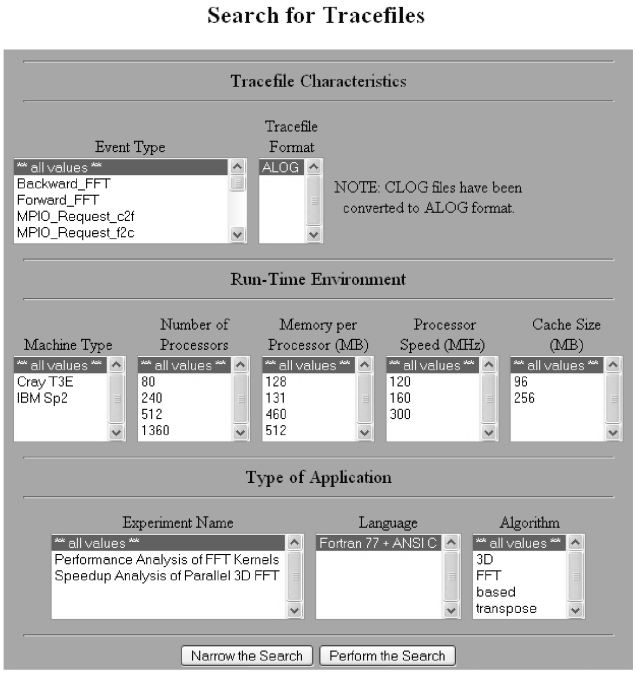


Figure 3 *Query interface-starting page*

The selection criteria available on the initial query interface page are those identified by representative users as the most useful in terms of facilitating discrimination among tracefiles in the testbed. Criteria are displayed in three logical groupings to improve legibility and selection efficiency. Tracefile-related choices include tracefile format and event types. Selectable machine environment variables are machine type, number of processors, memory per processor, processor speed, and cache size. The query choices relating to the application are experiment name (both physical and virtual experiments are displayed), source code language, and algorithm. The user can make multiple selections from any of the lists, in which case the *union* (logical OR) of the matching records will be returned. After making arbitrary selections, the user can choose to narrow the search by eliminating choices that are unavailable due to constraints imposed by other selections. This drill-down operation repopulates the lists with data reflecting the selected constraints. The procedure can be repeated as many times as the user chooses before the actual search is activated.

In subsequent screens, the user can browse the search results. Tracefiles are grouped into *tracefile classes* based on the unique combinations of language, source size, machine type, algorithm, compiler, and number of processors found. This helps users restrict the number of results before they view individual tracefiles, since queries may easily return hundreds of tracefiles (Figure 4).

When one or more tracefiles have been selected, the user may download them for use with a performance analysis tool. To allow users to view summary information without special tools, and to allow users to download tracefiles exhibiting particular performance characteristics, three types of performance summaries can be generated. One compares performance across tracefile classes, while the other two present timing information on individual events and segments, within the selected tracefile(s). The performance summary screen for tracefile classes is shown in Figure 5. From this point, the user can choose to download one or more entire classes or view more information on tracefiles within the class(es).

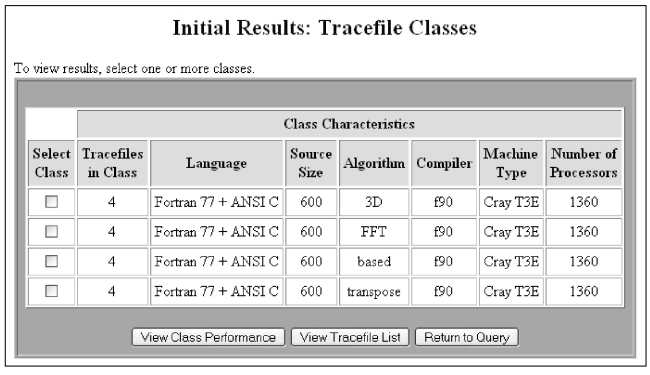
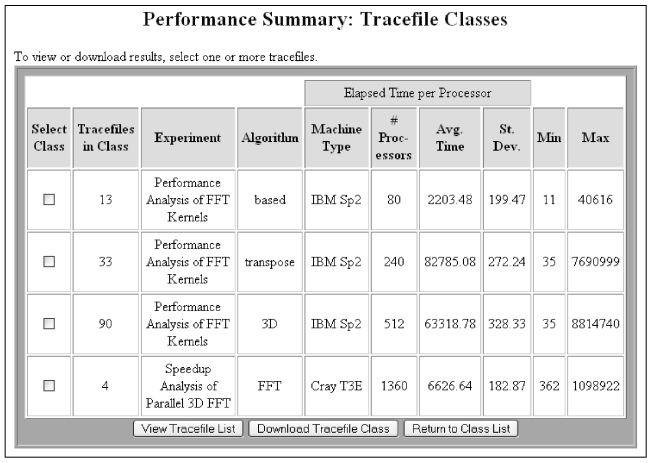


Figure 4 *Search results, grouped by class*



Performance Summary: Tracefile Classes

To view or download results, select one or more tracefiles.

Select Class	Tracefiles in Class	Experiment	Algorithm	Elapsed Time per Processor: Machine Type	# Proc-essors	Avg. Time	St. Dev.	Min	Max
☐	13	Performance Analysis of FFT Kernels	based	IBM Sp2	80	2203.48	199.47	11	40616
☐	33	Performance Analysis of FFT Kernels	transpose	IBM Sp2	240	82785.08	272.24	35	7690999
☐	90	Performance Analysis of FFT Kernels	3D	IBM Sp2	512	63318.78	328.33	35	8814740
☐	4	Speedup Analysis of Parallel 3D FFT	FFT	Cray T3E	1360	6626.64	182.87	362	1098922

View Tracefile List | Download Tracefile Class | Return to Class List

Figure 5 *Performance summary, by tracefile class*

Performance tool developers will want to use the tracefiles for testing their own tool functionalities; they may also be interested in graphical or more detailed performance summary information than that offered by the testbed. The Tracefile Testbed provides facilities for downloading tracefiles or relevant segments of tracefiles. Downloading entire tracefiles is accomplished through the 'Individual Tracefiles' portion of the interface, which provides a link to the tracefile in the testbed's ftp server. Additionally, users may download selected tracefile segments. To download selected segments, users mark the appropriate segments in the Segment Performance screen and select 'Download Segments'. This prompts a cgi program to parse the tracefile and create a new file containing only the original file's header information and the desired segments.

An upload interface was designed with the goal of encouraging users to supply adequate amounts of quality metadata, without being discouraged by the level of effort required. This was a challenge, given the number of metadata elements required for the testbed. While creating a virtual experiment is easy, since most metadata are already available in the database, the uploading of new tracefiles requires a significant amount of new metadata to be entered. While addressing this problem, we chose to put the form on as few pages as possible, rather than breaking it into smaller components over multiple pages. Thus, it is immediately clear how much information is required. In addition, it is endeavoured to minimise the amount of typing required by allowing users to copy and modify the metadata from an existing tracefile.

9 SUMMARY

Responding directly to a requirement that has been expressed in a variety of community forums, the Tracefile Testbed provides HPC programmers and tool developers with web 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 behaviour of existing programs and in the development and optimisation 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.

ACKNOWLEDGEMENT

This work was supported in part by the US Department of Defense HPC Modernization Program (Contract Number DAHC94-96-C-0008).

REFERENCES

Eigenmann, R. and Hassanzadeh, S. (1996) 'Benchmarking with real industrial applications: the SPEC high-performance group', *IEEE Computational Science and Engineering*, Spring Issue.

Fahringer, T. and Pozgaj, A. (2000) 'P3T+: a performance estimator for distributed and parallel programs', *Journal of Scientific Programming*, Vol. 8, No. 2, pp.73–93.

Karavanic, K.L. and Miller, B.P. (1999) 'Improving online performance diagnosis by the use of historical performance data', *Proc. SC'99*, Portland, Oregon.

Miller, B.P., Callaghan, M.D., Cargille, J.M., Hollingsworth, J.K., Irvin, R.B., Karavanic, K.L., Kunchithapadam, K. and Newhall, T. (1995) 'The paradyn parallel measurement performance tool', *IEEE Computer*, Vol. 28, No. 11, pp.37–46.

Newsome, M., Pancake, C.M. and Hanus, J. (1999) 'Split personalities' for scientific databases: targeting database middleware and interfaces to specific audiences', *Future Generation Computing Systems*, Vol. 6, pp.135–152.

Perl, S.E., Weihl, W.E. and Noble, B. (1998) *Continuous Monitoring and Performance Specification*, Technical Report 153, Digital Systems Research Center, June.

Reed, D.A., Aydt, R.A., DeRose, L., Mendes, C.L., Ribler, R.L., Shaffer, E., Simitci, H., Vetter, J.S., Wells, D.R., Whitmore, S. and Zhang, Y. (1998) 'Performance analysis of parallel systems: approaches and open problems', *Joint Symposium on Parallel Processing (JSPP)*, June, pp.239–256.

Shende, S., Malony, A., Cuny, J., Lindlan, K., Beckman, P. and Karmesin, S. (1999) 'Portable profiling and tracing for parallel scientific applications using C++', *Proc. SPDT'98: ACM SIGMETRICS Symposium on Parallel and Distributed Tools*, pp.134–145.

Yan, J., Sarukhai, S. and Mehra, P. (1995) 'Performance measurement, visualization and modeling of parallel and distributed programs using the AIMS toolkit', *Software Practice and Experience*, Vol. 25, No. 4, pp.429–461.

Zaki, O., Lusk, E., Gropp, W. and Swider, D. (1999) 'Toward scalable performance visualization with jumpshot', *The International Journal of High Performance Computing Applications*, Vol. 13, No. 2, pp.277–288.