Design and Implementation of a Parallel Performance Data Management Framework

Kevin A. Huck, Allen D. Malony, Robert Bell, Alan Morris
Performance Research Laboratory, Department of Computer and Information Science
University of Oregon, Eugene, OR, USA
{*khuck,malony,bertie,amorris*}*@cs.uoregon.edu*

Abstract

*Empirical performance evaluation of parallel systems and applications can generate significant amounts of performance data and analysis results from multiple experiments as performance is investigated and problems diagnosed. Hence, the management of performance information is a core component of performance analysis tools. To better support tool integration, portability, and reuse, there is a strong motivation to develop performance data management technology that can provide a common foundation for performance data storage, access, merging, and analysis. This paper presents the design and implementation of the Performance Data Management Framework (*PerfDMF*). PerfDMF addresses objectives of performance tool integration, interoperation, and reuse by providing common data storage, access, and analysis infrastructure for parallel performance profiles. PerfDMF includes an extensible parallel profile data schema and relational database schema, a profile query and analysis programming interface, and an extendible toolkit for profile import/export and standard analysis. We describe the PerfDMF objectives and architecture, give detailed explanation of the major components, and show examples of PerfDMF application.*

1. Introduction

Performance evaluation of parallel programs and systems, whether for purposes of benchmarking or application tuning, requires the analysis of performance data taken from multiple experiments [7, 12, 26]. While sophisticated tools exist for parallel performance profiling and tracing, allowing in-depth analysis of a single execution run, there is significantly less support for the processing and storage of multiple performance datasets generated from a variety of experimentation and evaluation scenarios. It might be expected that each performance tool solve the problem of multi-experiment performance data and results management individually, but one can argue this is neither a reasonable expectation, since resources may be unavailable to build such support for some tool projects, nor a desired one, given the potential for building incompatible solutions. Instead, to promote performance tool integration and analysis portability, and to foster a multi-experiment performance evaluation methodology in general, there is strong motivation to develop open performance data management technology that can provide a common, reusable foundation for performance results storage, access, and sharing. Such technology could offer standard solutions for how to represent types of performance data, how to store performance information in a manageable way, how to interface with the performance storage system in a portable manner, and how to provide performance information services to a broad set of analysis tools and users. A performance data management system built on this technology could serve both as a core module in a performance measurement and analysis system, as well as a central repository of performance information contributed to and shared by several groups.

This paper presents the design and implementation of the *PerfDMF* parallel performance data management framework. The research is motivated by our work in parallel performance analysis and in the development of the TAU parallel performance system [27]. PerfDMF addresses critical requirements in TAU for the storage, maintenance, and processing of multi-experiment performance measurements and results. However, our broader goal with the PerfDMF project is to provide an open, flexible framework that can support common performance management tasks and be extended and re-targeted to enhance performance data integration as well as reuse across performance tools used in the parallel computing community.

The objectives of our performance data manage-

ment research and the PerfDMF project are discussed in Section 2. PerfDMF is designed for the management of parallel performance profile data. The PerfDMF architecture consists of four components: profile input/output, profile database, database query and management interface, and profile analysis packages. Section 3 describes the PerfDMF architecture in detail. As performance tool technologists, the implementation of PerfDMF reflects our strong concern for tool integration, reusability, portability, and open software. Implementation choices we have made and API examples are discussed in Section 4. PerfDMF was developed specifically to handle large-scale performance profiles and a large number of profile results. We have integrated PerfDMF into the TAU performance system and include it as part of TAU's distribution. However, PerfDMF provides importers for six common profile formats and could be useful for performance tools other than TAU. Section 5 demonstrates the application of PerfDMF in TAU. Other projects have considered performance databases for different purposes. We contrast PerfDMF with these parallel performance tools in Section 6. We conclude with a summary of the research work and our future PerfDMF plans.

2. Objectives

Research in performance tools for parallel systems is driven by two equally important concerns. The primary purpose of performance tools is, of course, the effective evaluation of performance problems. The creation of powerful techniques for instrumentation, measurement, analysis, and modeling are important for the characterization, understanding, and tuning of parallel performance, particularly for complex scientific applications on large-scale systems. Accomplishments across the performance tools research community attest to the significant advancements being made in performance evaluation methods.

However, unless performance tools can be used in practice, in large-scale and diverse production environments, the advantages of new performance evaluation methods will go largely unnoticed. Performance technology must be built to higher tool engineering standards. Interestingly, the tool technology and engineering requirements that arise in parallel scientific computing pose research problems equally challenging to those in performance evaluation. These include tool portability (e.g., across architectures, computing models, and languages), scalability (e.g., to thousands of threads of execution), robustness (e.g., integrated with compilers, libraries, and runtime systems), and automation. There is also strong present motivation for tool integration, inter-operation, and reuse, and the development of performance technology following an open source methodology. We believe that by addressing these two general concerns – performance tool "science" and tool technology engineering – more sophisticated tools can be realized, built to high software quality standards, and readily assimilated in production environments.

In relation, the PerfDMF project represents primarily research in performance technology engineering. Most empirical parallel performance tools are targeted to the analysis of performance data from a single performance experiment. However, current interest in multi-experiment performance analysis is motivated by several purposes: benchmarking, procurement evaluation, modeling, prediction, and application optimization, to name a few. Indeed, several researchers have demonstrated the importance of multi-experiment methodology and tools [12, 16, 22, 26, 28]. Although there is an obvious degree of overlap in the profile data representation, organization, and basic methods of analysis in these projects, unfortunately, there is little technology sharing. From a tool engineering perspective, if the important profile data management features and functions could be captured in a common framework, performance tools could incorporate the framework in their design and interoperate with other tools that use the framework, resulting in several advantages for both enhancing performance tool capabilities and improving tool deployment.

The PerfDMF research focuses on the design and development of a common, reusable performance data management and analysis framework. The primary objectives of the PerfDMF research work are:

- Import/export of data from/to leading parallel profiling tools.
- Handle large-scale profile data and large numbers of experiments.
- Provide a robust profile data management system that is portable and easily reused.
- Support abstract profile query and analysis API that offers an alternative Data Management System (DMS) programming interface.
- Allow for extension and customization in the performance data schema and analysis API.

The following sections discuss how the PerfDMF design and implementation meets these objectives in more detail. It is important to note that we use the term "framework" to emphasize the intention of PerfDMF to be used as a component in an integrated performance

tool environment, one that can be configured and applied for the particular performance analysis purposes. It is the framework concept for performance data management and its representation in PerfDMF that is the primary contribution of our research.

3. PerfDMF Design Architecture

Empirical performance evaluation of parallel and distributed systems or applications often generates significant amounts of performance data and analysis results from multiple experiments and trials as performance is investigated and problems diagnosed. However, the management of performance data from multiple experiments can be logistically difficult, impeding the effective analysis and understanding of performance outcomes. The Performance Data Management Framework (PerfDMF) provides a common foundation for parsing, storing, querying, and analyzing performance data from multiple experiments, application versions, profiling tools and/or platforms. The PerfDMF design architecture is presented in this section. We describe the main components and their interoperation. Attention is also given to the profile database schema as the core of the PerfDMF database support.

3.1. PerfDMF Components

PerfDMF consists of four main components: *profile input/output*, *profile database*, *database query and analysis API*, and *profile analysis toolkit*. Figure 3.1 shows a representation of these four components, and their relationships. PerfDMF is designed to parse parallel profile data from multiple sources. This is done through the use of embedded translators, built with PerfDMF's data utilities and targeting a common, extensible parallel profile representation. Currently supported profile formats include gprof[9], TAU profiles[27], dynaprof[19], mpiP[29], HPMtoolkit (IBM)[6], and Perfsuite (psrun)[20]. (Support for SvPablo [24] is being added.) The profile data is parsed into a common data format. The format specifies profile data by node, context, thread, metric and event. Profile data is organized such that for each combination of these items, an aggregate measurement is recorded. The similarities in the profile performance data gathered by different tools allowed a common organization to be used. Export of profile data is also supported in a common XML representation. In the future, we may also offer exporters to a subset of the formats above.

The profile database component is the center of PerfDMF's persistent data storage. It builds on robust SQL relational database engines, some of which are freely distributed. The currently supported Relational Database Management Systems (DBMS) are PostgreSQL, MySQL, Oracle and DB2. The database component must be able to handle both large-scale performance profiles, consisting of many events and threads of execution, as well as many profiles from multiple performance experiments. Our tests with large profile data (101 events on 16K processors) showed the framework adequately handled the mass of data.

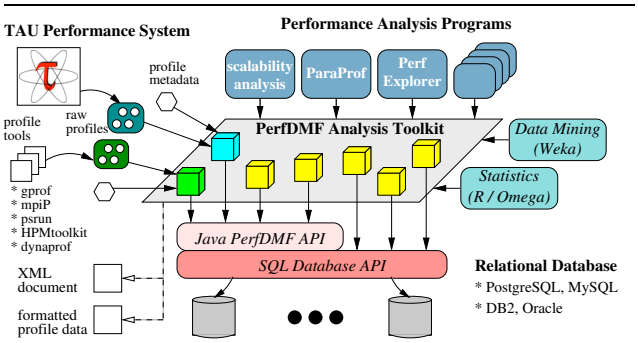

Figure 1. TAU PerfDMF Architecture

To facilitate performance analysis development, the PerfDMF architecture includes a well-documented data management API to abstract query and analysis operation into a more programmatic, non-SQL, form. This layer is intended to complement the SQL interface, which is directly accessible by analysis tools, with dynamic data management and higher-level query functions. It is anticipated that many analysis programs will utilize this API for implementation. Access to the SQL interface is provided using the Java Database Connectivity (JDBC) API. Because all supported databases are accessed through a common interface, the tool programmer does not need to worry about vendor-specific SQL syntax.

The last component, the profile analysis toolkit, is an extensible suite of common base analysis routines that can be reused across performance analysis programs. The intention also is to provide a common programming environment in which performance analysis developers can contribute toolkit modules and packages. As will be shown later in Section 5, analysis routines are a useful abstraction for developing profile analysis applications.

3.2. Profile Database Schema

A relational database schema is used to organize the performance data. The top level table, `APPLICATION`,

stores the data relevant to an application, such as name, version, description, etc. The EXPERIMENT table contains a foreign key reference to the APPLICATION table, and stores all data relevant to an experiment, such as the system information, compiler information, and configuration information. The TRIAL table contains a foreign key reference to the EXPERIMENT table, and contains information relevant to a trial, such as the date/time, problem definition, node count, contexts per node, and max threads per context. PerfDMF provides a flexible schema for these three tables. The schema requires that the id, name and foreign key reference columns exist in each of these tables, but additional columns may be added to (or removed from) the tables without requiring changes to the Java source code. This ability is provided by the getMetaData() call in JDBC, and provides flexible access to the columns in the database. The schema is designed such that if capturing such data as compiler names and versions, operating system attributes, etc. is important for analysis, then those columns can be added to the database. In addition, the analysis team is free to organize the performance attribute data in any way they like - the compiler information can be stored in the APPLICATION, EXPERIMENTor TRIAL table, or not at all. These features are important for the reusability of PerfDMF.

Some profiling tools, including TAU, collect more than one metric when executing an experiment trial. These metrics can include measurements such as CPU time, data cache misses and floating point operations, as well as derived metrics such as floating point operations per second. Because there can be more than one metric per trial, the schema includes a METRIC table, which stores the name of the metric and a foreign key reference to the trial table. Because some analysis tools also generate derived data, derived metrics can be saved with the profile data in the database using the PerfDMF API.

Performance profile instrumentation normally organizes interval data from a profile run according to functions, or as blocks of code given a "function name". Profiling tools can also organize interval data in smaller logical blocks, such as loops, basic blocks or even individual lines of code. The top level interval data table within a trial is the INTERVAL_EVENT table. The INTERVAL_EVENT table contains the name of the event, an event group (i.e. computation, communication, etc.), and a foreign key reference to the TRIAL table, indicating the trial to which it belongs. The INTERVAL_LOCATION_TABLE contains the cumulative data for each event, node, context, thread, metric combination. The data captured includes inclusive time, inclusive percentage, exclusive time, exclusive percentage, inclusive time per call, number of calls and number of subroutines. For some profiling tools, the value of one or more of these fields may be undefined. The INTERVAL_TOTAL_SUMMARY and INTERVAL_MEAN_SUMMARY tables contain the INTERVAL_LOCATION_TABLE total and mean values, respectively, across all nodes, contexts and threads.

In addition to the regular instrumented profile data, data from atomic events can be captured in profiles. In TAU, for instance, users can define atomic events at code locations to collect data which varies for each instrumentation call, such as the current application size in memory, or the size of an MPI communication. The ATOMIC_EVENT table stores the atomic counter information, such as the name and group name for the counter. The ATOMIC_LOCATION_PROFILE table contains a foreign key reference to the ATOMIC_EVENT table, as well as the sample count, maximum value, minimum value, mean value and standard deviation for each ATOMIC_EVENT, node, context, thread combination.

4. Implementation

Our goals in developing PerfDMF are primarily integration, reusability, and portability. We also wanted an implementation based on robust and open software and protocols. We have decided to use Java, JDBC, XML and ANSI SQL, for portability, standard DBMS connectivity and profile data exchange. There are four main components of PerfDMF, including profile input/output, profile database access, profile management, and the analysis toolkit. All four are self-contained modules, but they share common profile data objects and API. This section discusses the PerfDMF implementation from the perspective of analysis code development. Particular attention is paid to the performance database and the management component.

For historical reasons, there are two methods of data access in PerfDMF. The first method provides an overall data management toolkit, including all of the file parsers and database access for both querying and storing data. The other method provides access for just querying and storing data to the database directly. The nature of the analysis application will determine which method to use. For example, if an analysis application will only be a database client application, and the application developer wants to selectively query the data without having to load entire (possibly large) trials, then the database-only interface should be used. If the analysis application needs to support profile data directly from profiling tools in the form of flat files, and/or doesn't need database support, then the first method should be used. The selection of one method

does not preclude the use of the other, and the two are not mutually exclusive.

The two methods logically organize the profile data in the same way. Based on TAU's generalized performance data representation [27], PerfDMF structures its data in a *node*, *context*, and *thread* manner. Each thread then keeps track of a varying number of performance *events*, which associate singleton or aggregate data to named performance elements such as functions, loops or other blocks of code. In addition, for each node, context, thread, event, metric combination, there is an *event profile* object which stores the performance data for that particular combination. This event mapping approach allows an efficient and flexible method of performance data representation. Wrapped around this representation is PerfDMF's API for profile query and management. This API is implemented entirely in Java, and thus provides a completely portable and consistent method of accessing data.

The profile input component is responsible for obtaining performance data from a wide variety of sources, and converting it to PerfDMF's internal representation. It does so by creating a profile `DataSession` object specific to the profile format being imported. The `DataSession` object forms the core abstract object by which interactions with data sources take place. For example, the `GprofDataSession` provides an interface to parse gprof data. Some profiling tools output multiple files, one for each process or thread of execution. In those cases, PerfDMF provides support for parsing a directory of files, or a subset of files in a directory that start with a particular prefix or end with a particular suffix. The profile input component manages the details of parsing the output from the supported profiling tools. There is also support for parsing and managing TAU user-defined events, as mentioned in Section 3.

PerfDMF database access is provided through the use of interface functions that simplify the connection to the database. When building a client, the application developer need not concern herself with the details of database connectivity or with constructing SQL queries if she does not need or want to. It is relatively easy to get a list of `APPLICATION` rows from the database (returned as Java objects), and find an instance of interest. Iterating through the objects is similar to iterating through the the tuples of a SQL query, but with a simpler interface. The profile database component is provided by the `PerfDMFSession` extension of the `DataSession` class. Once the session has been initialized, a call to `getApplicationList()` will return a list of `Application` objects, from which the desired application is selected and set as a filter for subsequent queries. The code is similar for listing and selecting `Experiment`, `Trial`, `IntervalEvent` and `AtomicEvent` objects. Once an object is selected, all further query operations are filtered based on that particular context. For example, if a particular `Trial` has been selected, then any `IntervalEvent` objects that are queried are only those from that particular trial. Alternatively, an application could load an entire performance profile from the database or import from a raw profile dataset into a `DataSession` object (as was mentioned earlier with the gprof example), and then apply selections with the PerfDMF API, setting node, context, and thread parameters. Saving data to the database is also easy, in that the `Application`, `Experiment` and `Trial` objects all have `Save()` methods, which will save the object and all of its related object references to the database. The `Trial` object also has support for adding new, possibly derived, metrics to an existing trial in the database.

PerfDMF also provides an analysis toolkit component. This utilizes API support for application, experiment, and trial access to broaden single-trial profile analysis to multiple experiment datasets. This particular component is only implemented minimally, as the type of analysis done will be is somewhat application specific. However, there are methods for doing rudimentary multi-trial analysis, including performance comparisons and speedup analysis.

5. Application

This section presents some applications of PerfDMF to existing performance tools. We shall consider three applications of PerfDMF: parallel profile analysis and viewing, experiment trial browsing and scalability analysis, and performance data mining. Performance tools for these applications have been developed to use the PerfDMF API. Our ParaProf profile analyzer is particularly enhanced by the ability to parse additional profile formats, and the ability to store data to a database. The other two applications primarily demonstrate the use of analysis interfaces with the database.

5.1. ParaProf

ParaProf[3] is TAU's main profile browser, and is a portable, extensible and scalable tool for parallel performance analysis. ParaProf provides a mature, reliable platform on which to graphically browse parallel performance profile data. It implements graphical displays of all performance analysis results in aggregate and single node/context/thread forms. ParaProf also

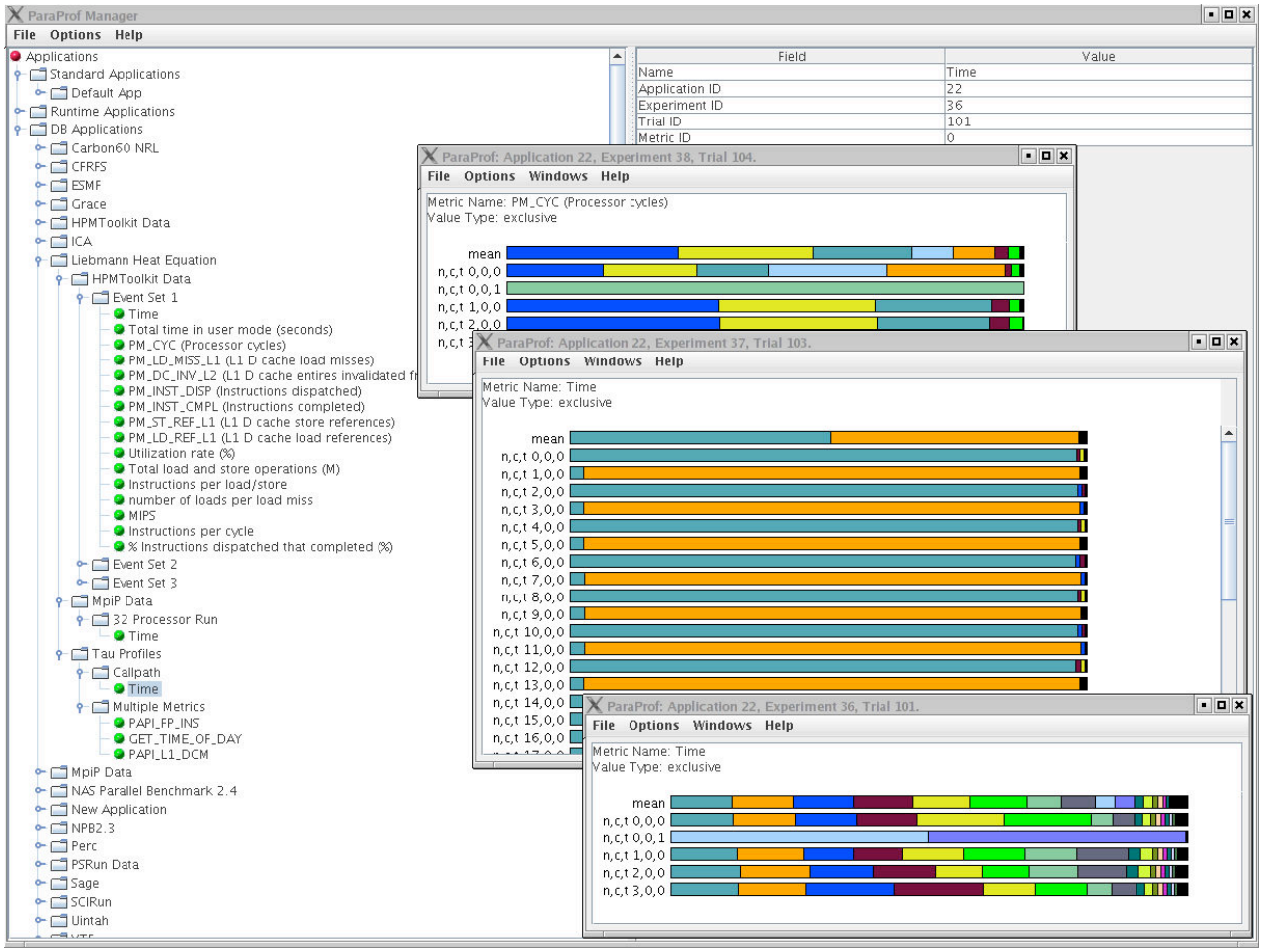

Figure 2. ParaProf with PerfDMF support accessing HPMToolkit, mpiP, and TAU data from a database archive. The top graph window shows the HPMToolkit data, the middle window is mpiP data, and the bottom window is TAU data. ParaProf can also be used to input data into the database.

provides the ability to compare the behavior of one instrumented event across all threads of execution, and offers summary text views of performance data, with various groupings and contextual highlighting. The initial release of ParaProf could only read TAU data from flat files, and though it could generate rudimentary derived data, it had limited methods by which that data could be saved for further analysis. With the addition of PerfDMF, ParaProf is now able to parse profile data from additional profile tools, and has database support for accessing archived profile data and saving derived metric data. ParaProf can also be used by an organization as the primary interface to the performance profile database, providing a graphical user interface which analysts can use to store and view performance profiles in a shared data repository.

Figure 2 shows an example of the enhanced ParaProf using the PerfDMF API to interface with the database. On the left side of the application window is a tree view of the applications, experiments and trials which have been loaded into the database. Three trials shown, all from the same application, have been loaded into the database using the PerfDMF API, and are expanded in the tree. The three trials come from three different profiling tools, specifically HPMToolkit, mpiP and TAU. Additional application profile data is loaded into the database, mostly from TAU data files. This figure is not intended to show comparative analysis between trials, but rather the use of PerfDMF to parse various profile formats and store them in a database archive. This archive could be made available in one physical location for all analysts within an organization. Given

PerfDMF's design, it would be a simple matter to implement access authorization to enforce different policies for performance data security and sharing.

5.2. Trial Browser and Speedup Analyzer

One application we developed to test the PerfDMF API was a trial browser and speedup analyzer. The trial browser exercises a broad subset of the functionality available in the API, and the speedup analyzer demonstrates the need for common analysis capabilities. We applied this tool to study the scalability of the EVH1 benchmark [21]. Given performance data from experiments with varying numbers of processors, the tool automatically calculates the minimum, mean and maximum values for the speedup every profiled routine. The application has access to this data through the PerfDMF API, including requesting standard SQL aggregate operations such as minimum, maximum, mean, standard deviation and others. The fact that the database provides the data in such an organized fashion leaves the application programmer free to develop analysis and visualization code, rather than worry about data management.

5.3. PerfExplorer

The PerfExplorer application is a data mining application for doing parallel performance analysis on very large profile datasets. Because current visualization tools are incapable of displaying thousands of data points with hundreds of dimensions in a meaningful way to a user, statistical analysis methods are used to perform cluster analysis on the data, and then do summarization of the clusters. Additional functionality is currently being added to PerfExplorer to perform additional data mining operations on the data.

PerfExplorer is designed as a client-server system. The client makes requests to an analysis server back end, which is integrated with a performance database, using PerfDMF. Using the PerfExplorer client, the analyst selects a particular trial of interest, sets analysis parameters, and then requests data mining operations on the parallel dataset. Using the PerfDMF API, the analysis server selects the data of interest, gets the relevant profile data and hands it off to an analysis application, R[23]. When R is done with the analysis, the results are saved to the database, using the PerfDMF API. When the analysis is complete, the user can browse the results using the PerfExplorer client. The browse requests are also processed by the PerfExplorer server, using the PerfDMF API.

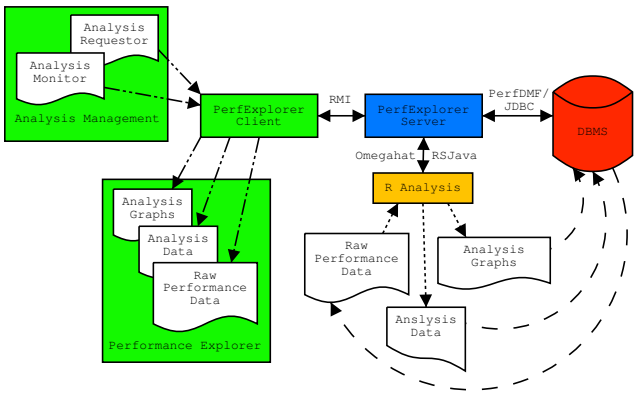

Figure 3. The PerfExplorer high-level design. PerfDMF provides an interface between the DBMS and the PerfExplorer server application.

Figure 3 shows how PerfDMF is integrated into the PerfExplorer application. Because PerfDMF provides database support, the PerfExplorer application developers are left to concern themselves with the user interface and the analysis portions of the application. Because PerfDMF is flexible and extensible, the PerfExplorer developers were able to extend the PerfDMF database API to support saving and retrieving analysis results.

The datasets that the PerfExplorer application have analyzed to date include the ASCI Purple Benchmark[15] applications sPPM, SMG2000 and SPhot and the Miranda[5] application in production at Lawrence Livermore National Laboratory (LLNL). The benchmark applications were run in a number of configurations on large parallel machines available at LLNL. The profile data was generated by a number of profiling tools, including TAU, gprof and in the case of sPPM, self-instrumented data for which a custom parser was written. Up to 1024 threads of execution were available on the resources at LLNL, and up to 7 PAPI[4] hardware counters were collected at a time. PerfDMF was instrumental in organizing the performance data, and providing easy access to the collected data using the PerfDMF API. Analysis results from Ahn and Vetter[1] were reproduced with PerfExplorer, showing interesting floating point operation behavior in the sPPM application. Because targeted data mining analysis of parallel performance data is now available in a reproducable, portable application, more interesting results could be found in other applications by other analysts.

The Miranda application data was provided by

LLNL, in the form of TAU profile data from test runs on Bluegene/L[14], currently in development. When completed, Bluegene/L will have over 64K processors. The test data we were provided was from runs of 8K and 16K processors. Over one hundred events were instrumented, and only one metric was available, wall clock time. The 16K processor run consisted of over 1.6 million data points, and the PerfDMF API was able to handle the data without problems.

6. Related Work

The PerfDMF project inherits from a rich background of research work in the fields of parallel performance benchmarking, performance analysis tools and environments, and performance experiment management systems. The need to manage performance data is a basic requirement of performance benchmarking activities, but is often accomplished in ad hoc ways. Both the Graphical Benchmark Information Service (GBIS) [8] and the more general Performance Database server (PDS) system [13] demonstrate the utility of a high-level access to a performance experiment repository that allows for meaningful queries without user-level knowledge of performance data storage details. With similar goals, PerfDMF intends to provide a common DMS substrate as a robust part of the framework for the development of performance analysis tools.

Directly relevant to PerfDMF are the projects that utilize a performance database as a component of a performance analysis system, particularly for multi-experiment performance analysis. The SIEVE (Spreadsheet-base Interactive Event Visualization Environment) system [25] showed the benefit of a simple table-based structuring of performance data coupled with a programmable analysis engine. More sophisticated performance data models, such as found in Paradyn [18] and CUBE [26], allow a richer analysis algebra to be applied to multi-experiment performance information. Both SIEVE and CUBE would naturally extend to implementation on top of a performance data management system such as PerfDMF.

Similar to PerfDMF, the HPCToolkit [16] targets profile-based performance analysis. It is able to merge data from multiple performance experiments in a database that is correlated with the program source and hyperlinked for analysis and viewing with the HPCView [17] tool. Performance data manipulated by HPCView can come from any source, as long as the profile data can be translated or saved directly to a standard, profile-like input format. To date, the principal sources of input data for HPCView have been sample-based hardware performance counter profiles. In addition to measured performance metrics, HPCView allows the user to define expressions to compute derived metrics as functions of the measured data and of previously-computed derived metrics. In contrast, PerfDMF can work with true parallel profiles from large-scale parallel executions and provides a programmatic interface for building analysis packages, such as those to compute common derived metrics. However, the two systems are complementary in many ways and we will investigate support for importing HPCToolkit profile data into PerfDMF.

The Prophesy system [28] successfully applies a performance database to manage multi-dimensional performance information for parallel analysis and modeling. The database is a core component of the system, implemented using relational DBMS technology and storing detailed information from the Prophesy measurement system and performance modeling processes. The Prophesy architecture has enabled it to be applied to both parallel and grid applications. PerfDMF follows in the spirit of Prophesy, with a specific focus on the performance DMS as a robust, open, and retargetable framework, and on analysis programming. PerfDMF supports the import of profile data from multiple sources and Prophesy's database schema suggests it could be a source of performance data as well. Designed as a framework, PerfDMF would enable open access to the archived performance data and provide a programming interface for building multiple analysis components. This could allow Prophesy's modeling algorithms to be captured as part of a broader analysis library. In this way, several performance tools could benefit from the advanced modeling analysis Prophesy provides.

The PPerfDB project [11] comes closest to sharing the broader objectives as PerfDMF. PPerfDB is developing methods for diagnosing the performance of large-scale applications using data from multiple executions over an application's lifetime. It supports the import of performance data produced from multiple sources and allow performance results to be exchanged and compared across geographically disperse sites. Performance information is related through hierarchical property, resource, and event mappings that enable PPerfDB to support powerful comparison and analysis operations. The PPerfXchange component enables distributed PPerfDB-enabled performance repositories to interoperate (see PPerfGrid [10]). We view PPerfDB and PerfDMF as complementary, rather then competing, systems. The PPerfDB architecture provides the opportunity for the two systems to co-exist in a performance analysis environment. PerfDMF might be

used, for instance, to store high-volume TAU performance results for an application suite, while supporting PPerfXchange-compatible interfaces that tie the performance data to a global PPerfDB system. We hope to work with PPerfDB developers on PPerfDB-PerfDMF integration in the coming months.

In [12], multi-experiment performance data is managed to encompass executions from all stages of the lifespan of an application. Here all experiment information is gathered in a *program space* which can be explored with a simple naming mechanisms to answer performance questions that span multiple program instances. With this interface, it is possible to automatically describe differences between two runs of a program, both the structural differences (differences in program source code and the resources used at runtime), and the performance variation (how were the resources used and how did this change from one run to the next). As this work demonstrates, the ability to easily access performance data history and comparatively process the data has high payoff for automating performance diagnosis. Our objectives with PerfDMF to implement necessary performance data management and analysis programming infrastructure are consistent with these aims of automated performance regression analysis and diagnosis. With a extensible programming layer for query and analysis, it is conceivable that higher-level abstract interfaces, such a program space, can be implemented using PerfDMF to offer more sophisticated diagnosis support.

ZENTURIO [22] is another performance experiment management system that incorporates an experiment data repository at the core of its architecture. While ZENTURIO shares features of Prophesy, PPerfDB, and HPCToolkit, it is remarkable for its implementation as a set of services for experimentation and analysis, with a graphical portal for user interaction. In comparison to all of these efforts, PerfDMF is specifically advocating a performance DMS component and analysis programming interface that is flexible for a broad range of applications and is based on open, standard implementations and reusable, pluggable toolkits. We believe PerfDMF could address much of the data management functionality present in these tools.

7. Conclusion and Future Work

The PerfDMF project is developing performance data management infrastructure that we hope will be leveraged by the performance tool community for purposes of tool integration, interoperation, data sharing, and next-generation performance analysis. The utility of PerfDMF is demonstrated in our own TAU parallel performance system, which has fully integrated the framework in its ParaProf profile analysis tool. Additionally, we have developed a performance data mining prototype, PerfExplorer, that incorporates PerfDMF for performance storage and shows how packages such as R can be easily integrated. PerfDMF is being requested by an increasing number of TAU users for application performance analysis as well as systems benchmarking.

We believe the support in PerfDMF for importing common profile data formats and developing reusable and scriptable profile analysis functions will appeal to tools developers and users alike. We hope to work with the University of Tennessee to integrate the CUBE [26] algebra with PerfDMF to implement high-level comparative queries and analysis operations. TAU already supports translation of parallel profiles to CUBE format for presentation with the Expert [30] tool. We also are working with Portland State University (PSU) and LLNL to apply PerfDMF to performance benchmarking for large-scale systems procurement. This work will involve building translational interfaces between PerfDMF and the PPerfDB/PPerfXchange [11] tool suite. Additionally, we hope to began work soon with ORNL on a multi-platform code evaluation study of the PERC benchmarks [21] and PERC performance tools, with PerfDMF being used to store the performance results. These efforts will be important steps forward in linking sophisticated performance technology and tools through open interoperable interfaces and standard data management technology.

Longer term we envision the development of performance profile data and analysis servers that are PerfDMF-compliant and can be easily configured and deployed for specific performance management tasks. The PerfDMF technology will be equally valuable in the creation of shared performance repositories for performance benchmarking purposes as it will for efficiently tracking the performance history of a single application code. We will continue to enhance the features of PerfDMF to address these broader concerns.

8. Acknowledgments

The authors wish to acknowledge the contribution of Li Li, a Ph.D. graduate student in the Department of Computer and Information Science at the University of Oregon, to an earlier design of a parallel performance database that served as a conceptual framework for PerfDMF. This research is supported by the U.S. Department of Energy, Office of Science, under contracts DE-FG03-01ER25501 and DE-FG02-03ER25561.

References

[1] D.H. Ahn and J.S. Vetter., "Scalable Analysis Techniques for Microprocessor Performance Counter Metrics.", Proceedings of Supercomputing, 2002.

[2] APART, IST Working Group on Automatic Performance Analysis: Real Tools. See http://www.fz-juelich.de/.

[3] R. Bell, A. D. Malony and S. Shende, "ParaProf: A Portable, Extensible and Scalable Tool for Parallel Performance Profile Analysis," *Proc. Europar 2003 Conference*, LNCS 2790, Springer, Berlin, pp. 17–26, 2003.

[4] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A Portable Programming Interface for Performance Evaluation on Modern Processors," *International Journal of High Performance Computing Applications*, **14**(3):189–204, Fall 2000.

[5] W. Cabot, A. Cook and C. Crabb, "Large-Scale Simulations with Miranda on BlueGene BlueGene/L", Presentation from BlueGene/L Workshop, Reno, 14–15 October, 2003.

[6] L. DeRose, "The Hardware Performance Monitor Toolkit," *Euro-Par 2001*, 2001.

[7] T. Fahringer and C. Seragiotto, "Experience with Aksum: A Semi-Automatic Multi-Experiment Performance Analysis Tool for Parallel and Distributed Applications," *Workshop on Performance Analysis and Distributed Computing*, 2002.

[8] Graphical Benchmark Information Service, http://www.netlib.org/parkbench/gbis/html/.

[9] S. Graham, P. Kessler, and M. McKusick, "gprof: A Call Graph Execution Profiler," *SIGPLAN '82 Symposium on Compiler Construction*, pp. 120–126, June 1982.

[10] J. Hoffman, A. Byrd, K. Mohror, and K. Karavanic, "PPerfGrid: A Grid Services-based Tool for the Exchange of Heterogeneous Parallel Performance Data," HIPS-HPGC Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models, in conjunction with IPDPS 2005, April 2005, to appear.

[11] K. Karavanic, PPerfDB. http://www.cs.pdx.edu/∼karavan/research.htm.

[12] K. Karavanic and B. Miller, "A Framework for Multi-Execution Performance Tuning," in On-Line Monitoring Systems and Computer Tool Interoperability , Thomas Ludwig and Barton P. Miller, editors, Nova Science Publishers, New York, USA, 2003.

[13] B. LaRose, "The Development and Implementation of a Performance Database Server," M.S. thesis, University of Tennessee, Technical Report CS-93-195, August 1993.

[14] Lawrence Livermore National Laboratory (LLNL), "Bluegene/L", http://www.llnl.gov/asci/platforms/bluegenel/.

[15] Lawrence Livermore National Laboratory (LLNL), "The ASCI sPPM Benchmark Code", http://www.llnl.gov/asci/purple/benchmarks/.

[16] J. Mellor-Crummey, "HPCToolkit: Multi-platform tools for profile-based performance analysis," *5th International Workshop on Automatic Performance Analysis (APART)*, November 2003.

[17] J. Mellor-Crummey, R. Fowler, and G. Marin, "HPCView: A Tool for Top-down Analysis of Node Performance," *The Journal of Supercomputing*, **23**:81–104, 2002.

[18] B. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. Irvin, K. Karavanic, K. Kunchithapadam, and T. Newhall, "The Paradyn Parallel Performance Measurement Tool," *IEEE Computer*, **28**(11):37–46, November 1995.

[19] P. Mucci, "Dynaprof." http://www.cs.utk.edu/∼mucci/dynaprof/.

[20] National Center for Supercomputing Applications (NCSA), "PerfSuite", http://perfsuite.ncsa.uiuc.edu/, University of Illinois at Urbana-Champaign.

[21] National Energy Research Scientific Computing Center, "Performance Evaluation Research Center (PERC)", http://perc.nersc.gov/.

[22] R. Prodan and T. Fahringer, "On Using ZENTURIO for Performance and Parameter Studies on Cluster and Grid Architectures," *11th EuroMicro Conference on Parallel Distributed and Network-Based Processing (PDP 2003)*, February 2003.

[23] R-Project, "R", http://www.r-project.org/.

[24] D. Reed, L. DeRose, and Y. Zhang, "SvPablo: A Multi-Language Performance Analysis System," *10th International Conference on Performance Tools*, pp. 352–355, September 1998.

[25] S. Sarukkai and D. Gannon, "SIEVE: A Performance Debugging Environment for Parallel Programs," *Journal of Parallel and Distributed Computing*, **18**:147–168, 1993.

[26] F. Song, F. Wolf, N. Bhatia, J. Dongarra and S. Moore, "An Algebra for Cross-Experiment Performance Analysis," International conference on Parallel Processing (ICPP'04), pp. 63–72, August 2004.

[27] TAU (Tuning and Analysis Utilities). http://www.acl.lanl.gov/tau/.

[28] V. Taylor, X. Wu, and R. Stevens," Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications," ACM SIGMETRICS Performance Evaluation Review, **30**(4), pp. 13–18, March 2003.

[29] J. Vetter and C. Chambreau, "mpiP: Lightweight, Scalable MPI Profiling". http://www.llnl.gov/CASC/mpip/.

[30] F. Wolf and B. Mohr, "Automatic Performance Analysis of SMP Cluster Applications," Technical Report IB 2001-05, Research Centre Jülich, 2001.