Experiment Management and Analysis

Joachim Worringen <joachim@ccrl-nece.de>
NEC Europe Ltd., C&C Research Laboratories St.Augustin

Abstract: *This document describes 'perfbase', a system for database-supported management of performance measurement results. We explain the motivation behind this project and discuss related work. Then, we present the design concept and workflow of perfbase, give an application example and describe the available commands of the system used to perform the actions of the workflow. Finally, the implementation strategy is explained and the current status of the implementation is given.*

1 Motivation

Each time that a performance-critical part of a software system is changed, it is necessary to re-evaluate the delivered performance to rate the effect of the change. For this purpose, benchmarks are run, and the output of the benchmarks is analyzed or transformed into a presentable form like a chart or a table. The same is true for testing correctness of a software. This can be considered a special case of a performance test with only a single result value, namely the number of errors that occurred.

The MPI development at CCRL is a good example for this. The MPI library consists of many independent parts (subsystems) which are constantly changed for different reasons:

- deliver better performance on the same platform
- change the behavior of the software, i.e. to adopt to given resource usage restrictions
- adopt the software to a new hardware platform or to avoid problems with a platform already supported
- implement new subsystems that perform new tasks

Another example is the research at CCRL for pricing of stock options which requires a large amount of parameterized simulation runs. The results of these runs, which depend on halve a dozen of parameters, need to be stored for further evaluation.

1.1 Problems

The way that the described performance evaluation and result analysis is done now bears a number of problems:

- Translating the benchmark output into the presentable form is often a tedious, manual task as the data needs to be extracted and transferred between different software tools.
- Because of this complexity, the performance measurements are often limited in the range of the applicable test parameters, and in the number of samples taken for a certain set of test parameters. This leads to results of limited usefulness due to the unknown statistical variance in the results. This is especially true for application areas with a significant variation of results like testing of I/O performance, testing performance on a non-dedicated system or running simulations which include error estimation.
- It is complex and error-prone to manage all results in a (big) number of files (of a certain type, usually text). It is not easy to discover which dimensions of the parameter space have not yet been measured precisely enough, or which one may need a closer look due to irregular results.

- Comparing a certain measurement with the same measurement performed some time ago is difficult. This applies partly because the old data is not longer available, can not be found, is stored in a different format or didn't cover all required parameter sets.
- It is very difficult for others to access, understand and use the data of one's measurements as the system used for categorizing the result files is typically neither documented nor consistent.

2 Related Work

Research for related work in this area resulted in a relatively small number of academic projects and commercial products.

2.1 Academic Projects

ZOO: *Desktop Experiment Management Environment* (http://www.cs.wisc.edu/ZOO): Similar, but more complex ("scientific") concept than perfbase with a higher level of abstraction concerning a "process", data and so on. The project is dead since 2000 at latest.

Paradyn (http://www.paradyn.org): In the scope of the well-known Paradyn project for dynamic instrumentation and performance measurement of parallel applications, two papers on experiment management have been published 1997 and 1999, respectively, and a Ph.D. thesis (Karen L. Karavanic) dealt with this topic. However, no active (sub)project for this area exists, and it is not clear if the software components described in the papers have ever reached a mature state.

ZENTURIO (http://www.par.univie.ac.at/project/zenturio), part of the ASKALON project (http://www.par.univie.ac.at/project/askalon): Using a directive-based language named ZEN to extract data from applications, ZENTURIO is a web-based (thus "grid-enabled") system to run and control experiments, and visualize the performance data. Due to this Grid-orientation combined with the required source-level instrumentation, the whole system is complex to set up and run. The state of the software is unknown as it is not publicly available; the project seems to be active as the most recent publication is from 2004.

PPerfDB (http://www.cs.pdx.edu/~karavan/pperfdb.html): PPerfDB is designed to compare performance metrics of different executions of large-scale parallel programs with different optimization parameters or on different machines/sites. This makes it similar to perfbase, but the technical approach is much more complex. To gather the performance data, it uses different techniques for tracing or (dynamic) probing of the application like *Paradyn*. As it seems, this project has recently been renamed (and redesigned) to *PPerfGrid*, to indicate that it is now using Grid technology for communication of the components involved. The project seems to develop along master theses (from 2001 up to 2004); the actual status of the software is unclear as it can not be downloaded.

CUBE (http://icl.cs.utk.edu/kojak): CUBE is a multi-experiment performance analysis tool. It applies a specific *performance algebra* to the trace information that maps the information from the trace data into the three dimensions *performance metric, call path* and *process (or thread)*. This algebra is an extension of the model used in Paradyn and PPerfDB (see above). Data can be explored along these dimensions in different ways. To compare multiple experiments, it is useful to see the performance difference between two experiments. Alternatively, the mean values of multiple experiments can be displayed. The trace data can currently be imported via the TAU system (http://www.cs.uoregon.edu/research/paracomp/tau/tautools). This project is actively maintained in the context of the KOJAK project, and the source code is freely available.

PerfDMF (http://www.cs.uoregon.edu/~khuck): PerfDMF is a project of the *Performance Research Laboratory* at the University of Oregon. It aims to provide a framework for performance data management. PerfDMF addresses objectives of performance tool integration, interoperation, and reuse by providing common data storage and analysis infrastructure for

parallel performance profiles. PerfDMF includes a relational database to store profile data, an abstract profile query and analysis programming interface, and a toolkit of commonly used utilities for building and extending performance analysis tools. The type of data to be managed is profiling data gathered from different profiling tools or libraries like TAU (of which PerfDMF is a sub-project), dynaprof, mpiP, or SvPablo. This requires the handling of very large amounts of data.

2.2 Commercial Products

MDL *Discovery Experiment Management* (http://www.mdli.co.uk/products/experiment): A commercial product line, consisting of numerous components. It is targeted at the bio- and chemistry-market.

Sona-Systems (http://www.sona-systems.com) provides a web-based system to manage psychological experiments. It is used mainly at US universities.

Xception, a commercial software validation suite, has a component named Xtract (http://www.xception.org/products/index.php?target=xtract) which helps the users to analyze the gathered data. It performs predefined queries to the result database and visualizes it in charts.

2.3 Assessment of Existing Approaches

Only few commercial solutions in this area seem to exist, and the solutions evaluated are focused on special applications (biology, chemistry, psychology, software testing), targeted at Windows as a platform and put much focus on the GUI. This is very different from the environment and problem description given above. Next to this, these solutions are, of course, not free to use.

The academic projects all stem from the area of high-performance computing. However, ZOO seems to be a dead project, with no available software. Similarly, the part of the Paradyn project that dealt with experiment management obviously was no longer maintained after the responsible person left some years ago and focused on PPerfDB.

The ZENTURIO/ASKALON project is actively maintained and includes components that are to fulfill a similar purpose. However, it relies on a complex infrastructure of source-to-source compilers, web services etc. Such an environment is difficult, if not impossible to set up on the SUPER-UX platform, or on an “isolated” site as i.e. the Fuchu plant where the user has neither a fast access nor any rights to install software or services.

PPerfDB has evolved into PPerfGrid, and the current Grid-based implementation shares the complexity of the ZENTURIO project to set up the infrastructure. The potentially large amount of data that is gathered by tracing the complete applications can make analyzing the derived information to find a specific problem very complex. Nevertheless, some of the components, like the *Xquery* query language, should be analyzed for the perfbase implementation.

CUBE provides a specialized way to look at performance data gathered by tracing an application run, while PerfDMF is more a framework to store, manage, access and analyze such data in a SQL database.

The research conducted on the web did not show any more active products or projects that are active and at least partly suited to our requirements. It can be concluded that none of the projects and products evaluated is suited to match our requirements. They all focus on exploring trace files of applications for performance analysis and tuning of individual regions of the application code. This requires the handling of (very) large amount of data which has to be moved from one site to another,

3 Proposed Solution

To solve the problems mentioned in section 1.1, a higher automation of the test procedure needs to be established, and the data gathered this way has to be managed consistently and persistently. The solution proposed and to be implemented is described in this section.

3.1 Executing Benchmarks

Executing benchmarks on (remote) systems in an "automatic" way is required. Automating this task in a generic way for different environments is a highly complex task (which is attacked in the context of grid computing).

If a manual adoption to a given environment is done, this actually is a solved problem either by using an adapted set of scripts, possibly in conjunction a batch system. However, this task can be supported by providing the parameter sets to run the tests with and by comparing the set of results by performed executions with the required set of results as specified by the experiment definition. This matching should be supported by the proposed solution.

3.2 Data Management

It is naturally to use a database to manage all data gathered. Different relational SQL databases (relational database management systems, RDBMS) are freely available, like MySQL (http://www.mysql.com) and PostgreSQL (http://www.postgresql.org). Because PostgreSQL is more powerful (concerning e.g. the supported data types) and more compliant to the SQL ANSI standard, and has a more liberal license, this RDBMS is chosen for the implementation of perfbase.

3.3 Input Data Interface

We propose a solution that uses plain ASCII text files for input data. There are virtually no formatting requirements to the input files. Reasons for this decision are:

- Text output, which can easily be captured in a file, is generated from arbitrary benchmarks and applications. One of the design goals is that for most cases, no modification of the benchmark is required to gather the relevant data.
- If a benchmark, library or application does not generate the required data as text output, the necessary "instrumentation" (print statements) can easily be performed in *any* environment, without setting up any tools, services or libraries, and regardless of the programming language or compiler used.
- Text files, or even output printed to a console, can easily be transferred between remote systems (even by cut & paste, if necessary).

3.4 Output Data Interfaces

Perfbase generates output data based on the gathered input data and the user's query. Internally, the data is stored as a multi-dimensional matrix. For further processing, it has to be written into a file, according to the specification in the user's query. Depending on the user's intentions, different formats for the output data should be supported:

3.4.1 raw-binary

The data is written to a file in binary IEEE representation. In this case, the endian format needs to be specified as well as the mapping of the matrix dimensions into the one-dimensional file.

3.4.2 raw-text

The data is written to a file as numbers in ASCII-representation. This allows writing out the data in two dimensions (rows and columns of the text file). Again, the mapping of the (two-dimensional) matrix slices into the files needs to be specified.

3.4.3 Self-describing Data Formats

Writing out raw data makes it impossible to connect the data back to from where it has been created and what it actually represents. Self-describing data-formats such as NetCDF and HDF-5 avoid such problems and are thus a suitable output data format.

3.4.4 Plot Files

A typical usage of the output data will be visualization, using a plotting program like *gnuplot* or *Grace*. Therefore, it is desirable to provide input files for such plot back ends which can generate a plot. The data itself can be contained in the plot file, or be provided in an external *raw-text* file.

3.4.5 Direct Plotting

Next, to provide the data to be plotted by a program in a file, it is also possible to provide it through a "direct" connection between *perfbase* and the plot program, i.e. though a pipe. This allows it to interactively visualize the result of a query without additional user interaction.

3.4.6 LaTeX table

The data may also be provided as a table in LaTeX format, to be directly included in a LaTeX document.

3.4.7 XML

Writing XML formatted data allows generic post-processing tools (like Excel 2003) to natively read the data files without the need to specify import rules (as required for text files).

4 Workflow with perfbase

It is important to recognize how a tool (or set of tools) can efficiently be used by a standard user. This includes setting up the environment (this, if too complex, may hinder many users to actually use a software) and running jobs on a daily basis.

4.1 Generalized Workflow

Working with *perfbase* means typically:

Access to a database server supported by *perfbase* is needed. This can either be a database that someone else is running for many users, or a database that is running on the user's local computer for his personal use with *perfbase* only.

Design an experiment which means choosing or writing a benchmark and specify the input parameter space, and the output values to be observed.

Set up a new database for the specified new experiment (run a specific *perfbase* command).

Run a series of tests. This typically is a script-/batch-controlled long/full range of tests, with storing all results/benchmark output in (a single) text file(s). But it actually doesn't matter how the input text files are generated.

Process the input text files to store all data in the database which has to be set up before. This can be repeated with any additional data you gather.

Perform queries to retrieve specified parts of the data, or derived data, from the database in a certain format (potentially including visualization).

At this point, the user typically iterates by running additional tests. It is also possible to modify the experiment by adding or removing parameter and result values.

4.2 Case Study

A case sturdy of a performance evaluation of an MPI implementation will illustrate the required kind of workflow.

We want to examine the performance of an MPI message protocol for a new interconnection network. In a first approach, we define the parameter values 'S_{msg}' (message size), 'S_{buf}' (size of internal buffer) and 'M' (transfer mode, which can be PIO or DMA). The single result value is 'L' (message latency). We set up the *experiment* (the test to generate the data we need) accordingly by supplying the experiment description file to the perfbase command setup.

We then run the benchmark 'B' we chose to use for this experiment. The message size is a command line parameter of 'B', while the buffer size and the transfer mode have to be specified as an environment variable. From the output of 'B' and the script that is used to run the benchmark, we derive the input description that defines how *perfbase* will extract the parameter and result values from this output to be stored in the database.

After running a series of benchmarks with different values of the parameters, we use the *input* command to import the data from the output files into the database. We want to know how much confidence we can have into the existing data and use the *query* command together with a suitable query description to provide the related statistical data like variance etc. From this data, we see that the error is too high, and we need to check the run-time environment for possible perturbation by other users, and run more tests.

Once enough data has been gathered to provide statistically safe information, we specify a query which will provide input data for gnuplot, plotting two charts (for DMA and PIO transfer modes) of L over S_{msg}, using S_{buf} as a plot parameter. Perfbase will also add error bars to the plotted values to show the confidence interval for each data point.

Some weeks later, we install a new software driver for the interconnect. We would like to know if and how this affects our message latencies. We want to use the existing perfbase experiment to perform this comparison. However, we need to extent the experiment definition by adding another parameter value, which is V_{drv} (the 'software version' of the interconnect driver). We can add this parameter to the existing experiment using the *setup* command. Additionally, we set the default value to the old version number of the driver, which will be applied for all data entries that are already stored in the experiment. We run a number of experiments within the new testing environment and import that data into *perfbase*. As the driver version can not be found in the output files of the benchmark (and we don't want to add it manually), we provide the value for this parameter explicitly for all runs when importing the data. We then can formulate a query that generates a plot with the absolute or relative *difference* of the latencies between the two available versions of the driver.

From these charts, we notice a performance drop for certain message sizes with DMA transfer. Eventually, we fix this problem with a new version of our protocol. To evaluate and document these changes, we introduce a new parameter V_{prot} (the 'protocol version' of our library) like we did with V_{erve} above, and re-evaluate the complete system. We can do similar changes when running the benchmark on different hardware, and so on.

After having delivered the fixed version of the library, we get user reports that claim on performance decreases of their application 'A' with the new version of the library. As we are able to run this application on our own, we want to conduct experiments with it to find the relation between our changes in the protocol, the measured latency 'L' and the applications performance.

Therefore, we define a new parameter value 'N_{procs}' ('number of processes') and a new result value 'T_A' (execution time of the application). We set N_{procs} to '2' for all existing runs, but leave T_A undefined. We define a new input description based on the output we get from running A. After having run application A, we import the data into the same database. Existing entries with the same parameter values, but an empty result value T_A should be updated with the new result value T_A. We then can analyze the correlation between the message latency and the application performance on the one hand and the parameter values on the other hand.

4.3 Detailed Work-flow

The work-flow for the application of this solution is described in more detail in the following subsections.

4.3.1 Experiment Description

Create an "experiment description" by defining the test parameters (*parameter*) and the resulting performance value(s) (*results*). This experiment will be referenced via a unique name (*experiment label*) which is supplied by the user.

All information that describes an experiment is stored in an XML formatted file. See the document *Perfbase Input File Syntax Specification*.

4.3.2 Database Setup

Based on the experiment description, create a new database in the with the (personal) performance database server. The experiment description will be stored in the database.

4.3.3 Experiment Run

Perform the experiments and store the results in one or more *result files*. These files are normal text files that will be parsed for the required *parameters* and *result values*. A single execution of the experiment with a certain parameter set is called a *run.* It has to be noted that two different types of parameters exist:

- *Singular parameters*: A singular parameter is a parameter that does not change its value throughout a run. It is typically used to differentiate runs, i.e. separate runs which have been performed on platform A from those performed on platform B. A singular parameter is only stored once for each run.
- *Repeated parameter:* A repeated parameter is a parameter that changes throughout a run. It is stored with each set of data in the data series of a run. An example for a repeated parameter is the size of a message in an experiment which measures the communication bandwidth for message passing. Of course, a repeated parameter can also be used to differentiate runs by choosing only those which contain datasets with specific values of this parameter.

The type of parameter, singular or repeated, is specified in the experiment description.

4.3.4 Input Description

Create an *input description* to extract the required parameters and results from the result files. The input description is again an XML formatted file. Typically, it is sufficient to create a single input description for one experiment which then can be used for all runs.

Next to retrieving information from the content of the result files, it is also possible to retrieve information from the name and the timestamp of the result file: it is common behavior to code parameters of the experiment run into the filename, and the time of the experiment run is given by the timestamp of the result file.

4.3.5 Experiment Data Import

Together with the input description and the experiment information already contained in the database, it is now possible to import the data of the result files into the database via the 'input' command. The extracted results will be stored in the database and form a *run* which has a unique, automatically generated *run index.*

A single result file may contain an arbitrary number of runs, which are split into single runs based on information from the input description. On the other hand, it is possible to import data for a single run from multiple result files, each of which is parsed using a different input description. The data of each run is stored in a separate table of the database (for the repeated parameters and results), and in a row of a shared table (for the singular parameters and results).

It is not desired to import the same result file more than once as this would lead to a falsified statistical relevance of this data. Therefore, a checksum is generated for each file, and data import is refused for a new file with the same checksum of an already imported file. The user can override this check using a special option if he is sure that this file should be imported anyway.

For all runs the absolute names of the imported files is stored. Additionally, it is possible to store the raw result files inside the database only for reference (this means, without storing external files).

4.3.6 Experiment Update

It is very common that a parameter or result value needs to be added to an existing experiment. A typical example is the addition of a new option or technique to existing software, and it is desired to compare results using the "old" and the "new" technique.

This can be achieved with *perfbase* by defining a new parameter with a default value that will be assigned to all existing runs. Generally spoken, it is possible to update an experiment with a new experiment description to add, change or remove parameters or result values, or to change the experiment meta data. This operation is also performed via the 'setup' command. When adding a parameter, a default value needs to be specified which will apply to all existing runs to allow valid queries including this parameter. When adding a result, this result will be undefined for all existing runs.

4.3.7 Data Retrieval and Experiment Analysis

To make use of the data stored within the database, means are provided to access the database to retrieve information from it. These accesses may return meta information or information derived from the data of the experiments as specified via a query.

4.3.7.1 Meta Information

Various types of meta information on the data can be provided:

- General information on the database by listing the available experiments, each with the number of runs contained, date of last modification etc.
- General information on an experiment in the database by listing the parameters and results, the different runs (with date, run label, number of results etc.)
- specific information on an experiment:
- range of a parameter or result
- number of result values for a parameter set

4.3.7.2 Data Querying

Querying the experiment for data is done with the 'query' command. An XML formatted query description is used to determine which data is used to generate the output data, how this data is processed to retrieve derived information from it, and how the output data should be formatted.

The general concept for a query is to define one or more result values to generate data for using the `<output>` tag. Without explicit specification, the data sources used would be all runs of this experiment, with all possible parameters values representing a different dataset. Such a basic query will make sense only for the most simple experiments as it has to be specified what to do with the result value(s) of datasets that have identical content for all parameters. These datasets have to be related or combined in some way, like using the minimal, maximal or average value, or deriving other statistical information from them.

A relation or combination of multiple result values is specified via the `<operator>` tag. This tag can either specify global combinations, like averaging all result values from datasets with identical parameters, or specify relations between distinct data sources by referencing them via their names (see below) to calculate the difference between the result values of each data source. Operators can be used hierarchically, like first averaging two data sources, and then calculating the difference of these two averages.

It is also necessary to be able to shrink the parameter space to limit the selection of datasets of which result values are taken. Such a limited parameter space represents a data source, which can be named for further reference (i.e. from operators). This can be achieved with the `<source>` tag, which includes an arbitrary number of `<parameter>` tags which limit the range of values that a parameter of a dataset valid for this data source may have. Apart from using parameters to filter the data, it is also possible to specifically exclude or include runs from a data source via the `<run>` tag. Within this tag, runs to be excluded or included can be specified via their ID, the creator, the date of creation, synopsis or data file name.

4.3.7.3 Data mining ("intelligent" data analysis)

The experiment analysis as it can be done with the *query* command allows to extract parts of the stored data, or to generate derived data from the existing data. The interpretation of the data is left to the user. This means, the user has to know in advance what exactly he wants to see.

Data mining tries to have *perfbase* provide hints to the user on typical patterns in the data of an experiment via the *datamine* command. Examples of such patterns are high (or very low) variance or non-monotone development of result values.

Typical questions that could be answered by data mining are:

- Is there an equal (sufficient) number of result values for all parameter sets?
- Are there significant variances on a result value across a single parameter set?
- Is there a significant variance or a recognizable trend for a result value across multiple runs?

4.3.8 Database Management

The initialization of the database server is done via the *init* command. Starting up and shutting down the database server is done via the *start* and *stop* command, respectively.

To remove data for individual runs from an experiment, or to delete an entire experiment (including all its data) from the database, use the *delete* command.

General access rights etc. for the database are managed "externally" by the default means for this, which are either the command line control of the database or any sort of front-end like pgadmin. The same is true for dumping and restoring (or importing) databases between database servers.

5 Implementation

5.1 Language

'Python' was chosen as the implementation language or perfbase for several reasons:

- Simple database usage via DBI
- Good string/text processing functions
- Portable as it exists on all major platforms
- Stable and highly backwards-compatible
- Free to use also in commercial projects & environments
- No compilation, but fast enough for this application
- Good support for different types of software distribution (open or closed source)

5.2 Database Server

The database server chosen to be used with perfbase is *PostgreSQL* (www.postgresql.org) for different reasons:

- Implements large parts of SQL92 (more than e.g. MySQL)
- Offers convenient data types (like arrays, large variety of time/date types, etc.)
- Simple to use, i.e. because of smart input data processing
- Good performance also for concurrent write accesses
- Available for a large range of platforms, incl. Windows and Linux
- Free to use also in commercial projects & environments

5.3 Description Language

The description of the experiments, parser, queries etc. requires an extensible content description language. XML is a natural choice for this task. Reasons for this decision are:

- No need to re-invent the wheel by defining a proprietary language
- A large range of tools and libraries for XML does exist:
- Parsers to process XML (also for Python)
- Generators to create XML (also for Python)
- XML-capable editors
- XML verification tools
- Offers a simple and defined interface for possible third-party software (like GUI-based tools).
- Perfbase description files will not get that large that processing speed or size of the files will be an issue.

5.4 Status

The implementation of perfbase has started in October 2004. First results of a working prototype version of perfbase have been presented on a CCRLE internal presentation at the beginning of December. Since then, perfbase has undergone steady testing and improvement and is now also used by other members of the lab. These users stimulate further development by their feedback.

The commands init, start, stop, setup, input, query, delete and info are implemented completely, making perfbase a fully usable solution already in the current state. The check command has basic functionality for version update of an experiment and will be extended as required.

Next to the software itself, a number of real-world examples and a complete test suite for automated regression testing (currently, about 50 tests) have been implemented and included in the distribution. The examples can be used to study how perfbase provides a solution to specific problem scenarios. The tests can serve as a reference to see how a specific perfbase feature can be used.

The next command to be implemented is 'find'. The 'datamine' command is not yet specified on implementation level. The setup functionality for automatic system integration needs to be implemented to simplify the installation of perfbase. The user documentation needs steady improvement and will be written in DocBook format (currently, a basic set of documentation exists in Word format).

The software can be requested NEC-internally from its author Joachim Worringen <joachim@ccrl-nece.de>. It is not yet decided whether to make the software publicly available, and which license would be used for a public distribution.