An Automatic Performance Data Handling Tool for Multi-Resolution Analysis in HPC Environments

Xavi Abellan[#1], Jorge Naranjo[#2], David Vicente[#3], Christian Simarro[#4]

[#]*Operations Department, Barcelona Supercomputing Center*
Jordi Girona 31, Barcelona SPAIN

[1]xavi.abellan@bsc.es

[2]jorge.naranjo@bsc.es

[3]david.vicente@bsc.es

[4]christian.simarro@bsc.es

***Abstract*— Performance analysis is an issue that is always taken into account in Computer Science, but specially in High Performance Computing (HPC) environments where it is a critical aspect. In this paper, we present APE, the Automatic Performance Engine. APE is a batch-system-integrated tool with web-based visualization capabilities based on the ideas of PerfMiner. It automatically handles performance data from the main components involved in an application runtime to boost multi-resolution analysis, from application to cluster-wide level, helping to characterize the system behavior.**

In this paper we describe the development and integration of the tool at Barcelona Supercomputing Center (BSC). We highlight its portability and present some examples with the kind of analysis that can be performed and some statistical data of the resource usage which is specially useful for HPC technical staff and managers.

I. Introduction

Supercomputing has been experiencing big leaps in the last years. This fast pace is an evidence that worldwide policy makers agree that it is one of the keys to leadership [1], [2], [3]. As a consequence, the supercomputing race has led us to reach the first Petascale systems and start preparing for the yet to come Exascale computers. Such big systems are expensive to build and maintain, specially in terms of energy consumption. That is why it is important to optimize the applications that are run in the HPC facilities so they can make an efficient use of the resources. Additionally, although we are already in the Petascale era in terms of hardware performance capabilities, software performance still needs to catch up with the new hardware architectures. This is where performance analysis tools arise as a basic element in HPC environments. There are several tools out there to do performance analysis, but the truth is that these tools are less used than one could expect due to its relevance. We believe the reason for that is the lack of a comprehensive, portable and easy to use tool that one can easily integrate in supercomputing systems to gather, store and visualize relevant information at all system levels.

In this paper we present APE, a tool we developed at BSC and integrated into MareNostrum [4] which is made up of 2560 IBM JS21 nodes and 42 servers p615. All the computing nodes are connected to a 2Gb/s Myrinet network [6] and sharing a parallel filesystem (GPFS). We took the PerfMiner's schema [5] as the starting point and developed a tool to be fused into the batch system to automatically collect and store performance information at various levels. It is completed with a web-based visualization interface where it is possible to perform multi-resolution analysis, from application to cluster-wide level, helping to characterize the system behavior. Using this tool a User Support specialist or an Application Consultant can detect and identify possible bottle necks and spot some performance flaws in an application. HPC technicians and managers can use this information as well, during procurements or to drive system upgrades.

One of the key points of APE is its portability. We have integrated it into SLURM but due to its modular design it can be also deployed into other supercomputers with different batch systems with little modifications.

The rest of the paper is organized as follows. In section II we describe the work done, starting with the reference of PerfMiner, pointing out the lacks we found and how we redesigned its schema to fit in a supercomputer such as MareNostrum. In section III we present the results through some examples and the different levels of analysis that can be performed. In section IV we summarize the recent work in the field and finally conclusions are given in section V.

II. Description of Work

In this section we present our design, development and deployment of Ape in the Barcelona Supercomputing Center, and specifically in MareNostrum. We start with the description of the main goals to be achieved, discussing pros and cons of the solution and then we explore the details of the resulting product.

A. Starting point

As mentioned before, there is a need to know if the applications that run in a cluster are making a good use of the resources, and if the global usage of it is what it was thought to be. There are many tools to measure performance, but most of them tend to be focused on a single experiment and require the user intervention. We wanted a tool capable of obtaining performance information in an automatic way and transparently to the user. This tool should monitor all kinds of

applications running on the cluster, and producing negligible overhead.

We found that PerfMiner was the only tool conceived following these concepts. It allows to collect performance data with no modifications to the applications or the user workflow, being able to store all this information in a database for later analysis.

It is not focused on a single run of an application, but in the whole cluster performance. It lacks the level of detail for a single application that other tools offer, but it is able to provide a general idea of the performance at several levels, ranging from thread, process, node, job and reaching cluster resolution.

However, the existing solution was only implemented in smaller clusters with a very specific configuration which made the portability to larger systems difficult. We then decided to produce a new tool, following the mentioned guidelines: transparency, automation, global approach, and easier scalability and portability.

B. Architecture

The new tool consists of five different parts or layers:

- Extraction of performance data
- Integration with the system
- Collection and processing of the information
- Storage
- Visualization

Each one of these parts are present in the different stages of the processing of the performance data gathering. Figure 1 shows the workflow of the tool from the moment that a job is submitted to the batch system until the performance results can be visualized and analyzed.

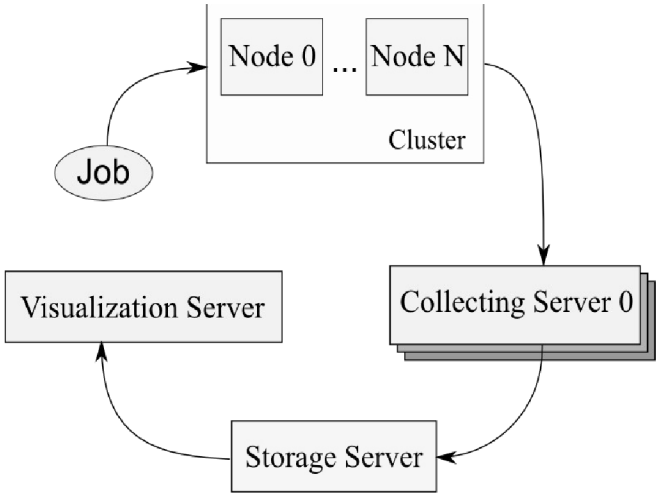

Fig. 1 Workflow of the tool

The process starts when a user submits a job to the cluster. It is important to remark at this point that the user does not have to make any change to the job or the applications to be run. As the job starts, the integration of the extraction engine with the batch system takes care of loading the proper environment and starting the gathering of the raw performance data. Note that only what is run in batch mode is considered. Interactive executions use to be short tests, compilation of programs and management tasks that are not relevant to the overall performance of the cluster.

Once the job finishes, the obtained data is sent out in a raw format to a collecting server. The mission of this element is to gather the information sent from the cluster, process and structure it. Then, the information is packed and flushed to the permanent storage server in a database. This approach separates completely the generation, processing and storage of the performance data, key point of portability and maintainability.

When the information is stored, it can be queried and visualized through the visualization server, which provides a web application that may be accessed from any client using a web browser. Besides, more tools and applications can be developed over the data server for specific of general purposes.

In the following sections we go through each of the layers detailing their main features.

C. Extraction of Performance Data

This is one of the most critical parts of the tool because the applications' runtime must not be modified. They must run fine, producing the expected results in the expected time, with virtually no overhead. Additionally, information from hardware counters (via PAPI [7]), usage of system resources and general statistics of the run must be obtained.

At this point we agreed with the PerfMiner's solution and decided to use Papiex [8]. This tool is able to generate a global report of the entire execution of an application at thread level, supporting MPI and OpenMP programming models.

One of the key features of Papiex is that it does not need to do any change on the monitored program at compile time because it uses the method of library interposition through the LD_PRELOAD mechanism. Papiex is able to intercept the creation and destruction calls of processes and threads to gather the required information. It also intercepts I/O and MPI calls to produce general stats of their usage.

It is fully configurable at runtime via environment variables, which makes it specially useful for automatic and transparent integration into the execution environment. In addition, as it is works over PAPI, we can be sure this tool is portable to any architecture where it is available.

The output is generated in files that are easy to parse. They are structured in a hierarchical fashion depending on the process number and thread number, if it is the case. Nothing is written to the standard out or standard error and thus, no interference is noticed from the user's point of view.

Taking the original free version of the tool, we have implemented some new features to improve the integration into the tool, making it more efficient. With that in mind, new information was added to the output to make each thread easier to contextualize, such as the MPI rank and the SLURM Step identifier in parallel runs. This step identifier corresponds to a single parallel run within a job with more than one.

But the main concern was the storage of the raw files with the performance data once the application has finished. In parallel executions, the directory where the files are created must be visible and accessible from all the processes. This means that only a shared file system can be used for this purpose.

Our experience tells us that, in the case of GPFS, the parallel file system in use at MareNostrum, the parallel writing of a great number of files in the same directory leads to a performance decay.

Our solution to minimize the stress of the parallel file system was to use the local disks in each node. The drawback of this strategy is that the performance information is scattered among all nodes, and thus it needs to be gathered in some way.

We extended Papiex functionality to be able to do the gathering at the end of the execution through MPI, using the interconnection network instead of the standard one. Finally, the gathered results would be stored in the local disk of the master process.

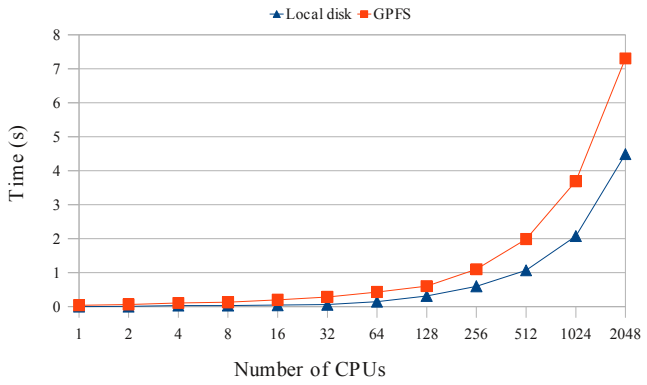

Fig. 2 Comparison between GPFS and local disk for the generation of the performance data files from Papiex

Figure 2 shows the time it takes to write in parallel different number of typical Papiex output files directly to GPFS or using the local disk approach. The use of local disk is proven to be faster than the shared file system, and the difference between both methods increases with a larger number of processes.

Of course, this approach is implemented as a new feature, not as a substitution of the standard behavior of the tool. It can be configured in runtime just the same way as the other options of the program can be set.

Since the monitoring of applications will be enabled by default, we also wanted to be sure that the overhead introduced by this tool would be negligible as it is supposed. Some benchmark executions are shown in table I. For the test, we used the well-known Linpack and NAS Parallel Benchmark. We compare the execution times of the standard execution and the monitored execution, computing an average of 10 runs of each kind. As we can see, there is no significant differences between them. The little variations in the execution time can be attributed to the usual noise of the system.

TABLE I
BENCHMARK COMPARISON

Benchmark	CPUs	Usual Time (s)	Monitored Time (s)
Linpack	128	3725.92	3713.77
NPB BT C	64	179.91	180,68
NPB CG C	64	35.08	36,01
NPB EP C	64	15.60	13,70
NPB FT C	64	36.58	36.11
NPB IS C	64	9.20	9.05
NPB LU C	64	49.10	49.77
NPB MG C	64	10.49	11.17
NPB SP C	64	134.04	135.17

D. Integration with the system

Papiex and other similar tools always offer a helping utility to properly load the right environment and execute the application to be monitored. In our case, however, the applications and the jobs should be run without any modification. In exchange, we have developed a plug-in for SLURM [9], the resource manager used at MareNostrum and many other clusters, to carry out this task.

The plug-in takes advantage of the SPANK API provided by SLURM. It allows an easy integration with the resource manager as an optional add-on that can be installed without changing the resource manager itself. The API offers a number of hooks that will be automatically called in the key moments of the job life cycle, to be implemented in the plug-in with the desired functionality.

When the job starts, the plug-in is automatically started. It loads the settings from a configuration file, where parameters like the information to be obtained or where to send it once finished. Several levels of information detail are defined, with different parameters and hardware counters to be monitored. They can be defined hierarchically, where the lower levels would contain less information and upper levels would be more complete, or independent, where we would find different sets of information with no relation to each other.

A default level for all jobs can be set, as well as a different one for specific users or groups. This allows to customize the monitoring of applications for a certain user or group if needed. These levels should include the most common information to be able to use it in global stats as well as specific analysis of a job or application. Users are also allowed to modify their default level, setting an environment variable with the desired level, or even to disable the monitoring for tracing or debugging purposes.

Before starting to run the actual job, the proper environment is loaded, including LD_PRELOAD of monitoring libraries and Papiex options. Then, every started process or thread within the job is monitored.

At the job ending, the plug-in sends all the information from the files generated by Papiex out of the cluster to one of the collecting servers configured. After that, it takes care of the clean-up of the temporary performance files that were generated in the job.

Note that this plug-in is totally independent of the information generated and what is required to do with it later. The post-processing of the files is left for the next step of the

workflow, isolating the generation of the information, the integration with the system, the loading mechanism, and the processing and later storage. This approach separates from the cluster everything that is not specifically related to the job's execution.

E. Collection and process of the information

Several strategies for the collection and the processing of the obtained information in the cluster could be implemented for this tool. A passive strategy from the job point of view would be valid for small clusters, with a harvester that periodically searches for new information to process. That is PerfMiner's approach.

We opted for the active strategy, where the information is sent out once it is generated at the end of the job. This solution is more portable and flexible due to the fact that the collector does not need to be in the cluster or have access to any of its file systems.

In order to implement it, we have designed a client-server architecture, where the client is the plug-in that sends the information, and the server is the collector that is constantly waiting to incoming transmissions. This design also considers the possibility of having multiple servers for high availability or load-balancing.

The server listens to a certain port for incoming calls from the cluster. It will receive the raw information and will process it and keep it structured in memory in a buffer with a certain number of slots, one for the information of each job.

It is possible to filter the received information and discard useless information of certain processes tracked by name or by execution time. The decision is left to the administrator of the cluster, who knows if there are auxiliary system processes that are always run but do not add valuable information to the execution of the applications, but will be spending a considerable amount of storage space in later stages.

Note that the information processed from the cluster will not be permanently stored in any file or database immediately. The server is configured to perform a flush of all the received jobs at regular intervals of time. When the time comes, the server packs all the information, builds the necessary database statements and flushes all data in a single connection, instead of doing one connection for each job. Using this strategy we reduce and optimize the interactions with the storage servers.

The flush operation might not finish instantly, and other petitions from the cluster might arrive. The server must be ready to store the information that will be sent and must not overwrite or miss any stored job. For that reason, we applied a double buffer technique to the server. The server holds two different buffers, one is active and the other is empty and waiting. When a flush is to be done, the server automatically swaps buffers. The empty one that was waiting goes ready to receive next petitions, while the other goes back to be flushed.

Thanks to the buffer swap, we guarantee that no new information conflicts with the existing one while a flush operation is being performed. It also avoids the incorrect flush of the information of an incoming transmission from the cluster.

The server runs as a daemon in the system, being executed manually or as a system service. Like any other service of the system, it can be configured using a Linux-like configuration file. In that file, the parameters of the server are defined, such as the port, the number of working threads or the filtering criteria.

A log system is also provided, with different levels of verbosity. This parameter, set in the configuration file, allows only the errors to be logged in production stage, as well as printing all the steps and commands executed for debugging purposes.

This server is designed and implemented to be portable and reusable. It is coded in C++, taking advantage of the object oriented paradigm. Thanks to this fact, it would not be difficult to adapt it to an eventual change of the extraction tool, which is currently Papiex, or the database system described in the following section.

F. Storage

All the performance data generated and collected in the previous stages should be permanently saved following a smart structure for later analysis. A relational SQL database is the perfect choice, because it is a good system in terms of simplicity and efficiency at storing the information and at its querying. It offers a simple and flexible interface to the stored data, very useful for any kind of analysis at different levels of detail.

This is the same conclusion reached in PerfMiner. As a matter of fact, the solution we show is inspired on the original PerfMiner database, taking the main guidelines of its design and implementing them in a different way. Our design is focused on the easy extensibility and the different levels of resolution of the information, as well as the efficiency in the database performance. A target hierarchy is defined, starting at the cluster level, and going through job, node and process until reaching the thread level. Each of them will only contain the information to identify the element and its basic metadata.

Since each job might have different measurements depending on the level of extraction, and more levels or modifications to the current ones might be made in the future, the measurements cannot be stored along its target in the same table. With that design, tables with a great number of columns would be created and containing a lot of null values for the metrics not measured in a certain job.

The solution is then map each metric to a different table containing all the measurements gathered for it. There will be no null values in this case, as only the measurements actually done for a job will be stored in their corresponding tables which will be linked to the execution. All metrics are related to a specific level in the hierarchy. For example, the IPC metric is measured by thread, but the distribution of the nodes in terms of the interconnection network for a job would be linked to the job level.

Another advantage of having this clear separation among the metrics and the target hierarchy arises at the time of building the queries for the analysis. Only the tables needed for it will be joint, and leaving the rest of the information out.

The database also contains metadata for the metrics stored to make easier the automation of the query building process.

A critical issue in this stage is also the huge amount of data that will need to be stored. Most of the metrics are thread-related, so it is easy to end up with tables containing millions of rows in the database. With such a volume of information, some queries to the database may become slow, and this is an issue that may affect the responsiveness of the analysis and visualization tools. In some cases, the responsive time is less important than the results, such as aggregated analysis for a large period. But in some others, such as a performance report or analysis of a certain job, the responsiveness of the database becomes more necessary.

For this reason, a cache strategy has been adopted, creating a new database with the same schema as the original. The collecting server will add the information to both databases to keep the data consistent between both databases, but the one acting as a cache will only hold information for a certain period. The older information is deleted from the cache when it reaches the limit set, but it still remains in the root copy of the database. Applications or analysis for specific jobs can benefit from the responsiveness of the cache database, as the users tend to ask for the performance of a recent job. For global analysis or a specific query for old information, the root database is still available.

G. Visualization

The database system previously described offers the opportunity to access performance information at many different levels of the recent executions as well as historical data. SQL provides a very flexible and powerful way to obtain useful results and aggregates from the stored information. For specific queries, it is possible to directly query the database engine to get concrete answers for complex questions.

However, this method is not suitable for the end-user for obvious reasons. They require an easy way to visualize the performance obtained from their executions, but at the same time enabling security policies to guarantee that they only have permissions to access their information.

We have developed a web application as a front-end to the stored information. The advantage of it is clear: users are able to do their analysis on the performance of their jobs from any computer with an Internet connection. There is no need to install any kind of software or log in to any machine.

The access to this application requires the users to provide their credentials. Depending on the roles assigned to each particular user, they will have privileges to do queries over the information generated by them, by their group, or by any user in the machine in the case of administrators.

The application is designed by modules, and each one of them implements a specific use case of the system. Thanks to that, it is easy to expand the features of the application by adding new modules. Of course, a module can be protected and only the users with the right privileges would have permission to use it.

The basic module consists of a report generator of the performance obtained from a specific job. Users can select the desired job and see a brief report of their application behavior once it is executed. Only the valid metrics for the job are displayed, in an average of the whole execution, including parallel ones. If the application is parallel, users are able to check the values of a certain metric in each one of the processes or threads of the execution.

At the end of the day, this report gives a global picture of the application's performance, showing the distribution of the execution time, including the time spent in communication or I/O. Used memory or processor related metrics are also shown, and thanks to the process / thread detail plots, imbalances in any metric can be detected.

III. RESULTS

Our integration provides a flexible, transparent, easy to use and maintainable tool that you can use to carry out several kinds of analysis at various levels while you are targeting different elements in a supercomputing environment. As an example, we will describe two kinds of analysis you can perform. It will illustrate how you can target different aspects of the supercomputer resources stack. We used Paraver [10] to validate the results. Obviously, Paraver is a very powerful and intrusive element that is rather used when you are doing a deepest code analysis. But as it is a well established and proved suite we use it to compare the results and corroborate the hypothesis aroused using our tool. Normally, when analyzing applications, you can use our tool as a high level non-intrusive tool to generate some hypothesis of the problem you might have. Then, if required, you can go deepest with Paraver to analyze the code in detail.

But also a global analysis can be done with our tool and in this case it cannot be corroborated with Paraver. This highlights its flexibility as one can obtain statistics from the overall system and outlook different parts of the HPC environment.

A. Application performance analysis. Validation with Paraver

In this section we have chosen Gromacs [11] version 4.5.3 as example program. Gromacs is a widely used Molecular Dynamics package primarily designed for biomolecular systems. Gromacs' performance has already been deeply analyzed at BSC. For this test we are using a 5,000 steps simulation of a Nucleosome molecule composed by 145,723 particles. The HPC machine for this validation is MareNostrum and there will be two runs of 64 MPI tasks. As each MareNostrum node has 4 cores, there will be 4 tasks per node. One run is having APE automatic extraction enabled. The second one is using the Extrae [10] mechanism which works with the PMPI feature available at MPI standard. It intercepts at execution time all the MPI calls using dynamic libraries and LD_PRELOAD. This instrumentation extracts all the information concerning the behavior of the application through the MPI calls. It stores the data into intermediate binary files during the execution. Once all the information is gathered and the program has finished, these files have to be merged to form a .prv file which can be read with Paraver.

Table II shows the overhead introduced (~8.1%) by the Extrae tool compared to the almost negligible one when executing with APE.

TABLE II
GROMACS EXECUTIONS WITH 64 PROCS AT MARENOSTRUM

	MareNostrum Gromacs 64 processors 5k steps	
	Total seconds	**Ns/day simulation**
Plain execution	154.539	5592
APE	155.260	5.566
Extrae	169.000	5.113

The comparison of the performance analysis is based on the application's cycles. They have been split into two main groups: MPI cycles and Useful cycles. We ignore I/O cycles bcause in Gromacs, they suppose less than 0.001% of the total execution. APE allows to obtain these counters from the database in a straightforward way. On the other hand, the cycle counter extraction process with Paraver becomes more tiresome. This run generated a 4.2Gb trace file which had to be filtered and cut until it became more manageable. Just then, this data could be extracted.

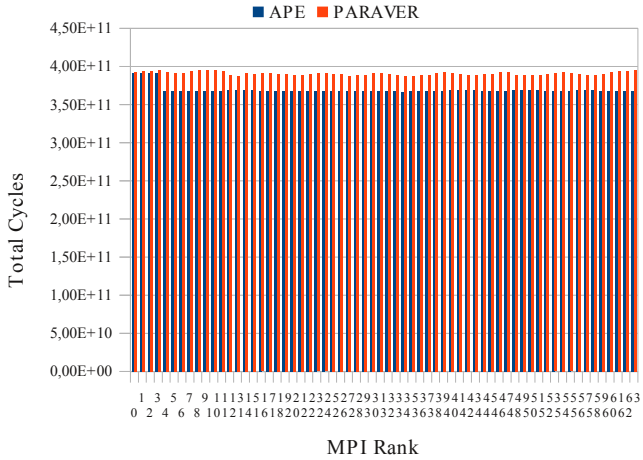

Fig. 3 Total cycles in two executions: one with APE environment and the other with Paraver extrae on.

Figure 3 shows the total cycles count of each execution where the different behaviour of each mechanism can be noted. APE extracts information about the whole execution whereas Extrae starts its process at the MPI_Init call and finishes at MPI_Finalize. All the cycles out of this region are not included. Figure 3 shows this behaviour, nevertheless it is undermined by Paraver instrumentation overhead. The first node (four processors) shows a different behaviour in the APE's case. The reason is that MareNostrum MPICH_MX [REF] implementation spends ten seconds on the master node to configure all the parallel environment of the run. Since this step is performed before the MPI_Init call, Extrae does not take it into account.

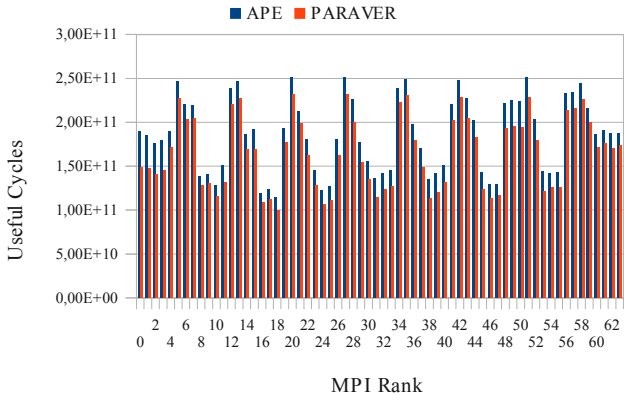

Fig. 4 Useful cycles

Figure 4 shows from the total amount of useful cycles, those which belong to computational time. In this case, the overhead of APE over Extrae is produced by the uncounted cycles before MIP_init and after MPI_finalize calls.

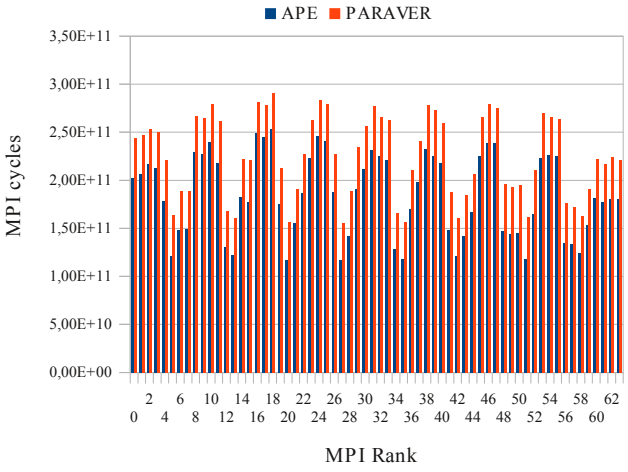

Fig. 5 MPI cycles

Putting together the figure 4 and 5, we observe an important load imbalance pattern, where some tasks use more Useful_cycles (work) than others. Similarly, the tasks with less work are the ones with more MPI_cycles, because they are waiting for the overloaded tasks. This work imbalance causes a performance penalty.

Once the analysis results are validated with Paraver, this section demonstrates that APE is also a powerful tool not only to perform a system general analysis but also to detect strenghts, weaknesses and behaviours of an specific program. Therefore, a user should use a fine-grain performance instrumentation tool to find the key of the problem.

B. Cluster-wide analysis

As mentioned in previous sections, this tool is not only useful to analyze specific executions of an application, but also to extract conclusions at higher levels of abstraction.

Several kinds of global analysis can be carried out with this tool, and it can be used by the administrators and managers of the cluster to know the real usage of the machine, understand the usage of the different resources and help future upgrades or acquisitions of new machines.

In this sense, the first element that a global system analysis would include is a characterization of the mean job of the system. This gives an idea of the kind of jobs, in average, that are submitted to the machine. We would know the number of requested CPUs, its memory usage or CPU utilization. Distribution time profiles would also be included, with the time spent in communications or in I/O operations. Other average metrics of any kind might be included in this characterization, such as Mflops or IPC.

When computing averages on these metrics, some considerations must be made. The most usual mean (the arithmetic mean) may not reflect the real values of the machine, since all the samples used to compute it are treated equally. As a consequence, a very short execution would have the same weight than other that used the resources for longer. For this reason we also compute the weighted mean of the values considering the time they consumed, giving more importance to those which spent more time.

Another useful analysis would be the one based on the applications run in the cluster. It would be easy to detect what the most used applications are and how they perform. Across all executions of each one of the applications, we compute averages and maximums of useful metrics such as the ones mentioned before. In this case, we also offer, for the Mflops metric, a weighted mean using the total amount of floating point operations made by the application. With this new average, we are promoting those executions with more floating point operations.

It is also possible to focus the analysis on a specific application across several executions, taking into account different input files, users or number of processes spawned in case of a parallel application. A similar approach would be the analysis of the different parameters and metrics of performance that every user is obtaining in their executions. This would help to classify the users based on their use of the system and its resources.

These are examples of what can be extracted from this tool, which we consider to be useful. However, since the information is easily aggregated following different criteria, a vast number of analysis might be done.

IV. Related Work

Performance measurement of the applications has always been important in HPC environments. For this reason, a number of programs or tools devoted to this task has been developed along the years, with different approaches and targets.

Profilers such as gprof [13] shed some light on the time distribution in a program by user function, although they usually require to recompile the application. Another example of a simple profiling tool is mpiP [14], which generates statistics of the MPI usage and behavior of a parallel application.

Other tools go beyond, being able to offer richer metrics and useful information about the resource usage of a certain program. In this category we could find tools that provide general information about the execution, such as Papiex or the IBM High Performance Computing Toolkit (IHPCT) [15].

In addition, frameworks such as Scalasca [16], PerfSuite [17] or Tau [18], are able to extract and show graphically the performance of an application, with a great number of possibilities to analyze and visualize different aspects of the behavior of the application.

In that direction, trace generators and visualizers can also be found. Examples of them are Vampir [19] or Paraver. This kind of tools instrument the analyzed program and are capable of generating trace files of the whole execution and visualize the information they contain in multiple ways.

However, all this tools are experiment oriented, so they are not applicable in a global way for all the applications, which is APE's approach. PerfMiner is the closest tool to ours, as they share the same philosophy. Both are able to obtain and store information of all applications running in a cluster, while offering a powerful and flexible way to carry out an analysis of the collected data at multiple levels of abstraction.

Other integrated solutions exist, but they have slightly different approaches. An example is IPM [20], which is a tool, integrated in the system execution environment. It generates profile reports of an application with low overhead for the user, but it is not able to collect the results to do a global analysis at a higher level.

V. Conclusions and Future Work

This tool is the result of the effort of the BSC support team to deliver to users, support staff, HPC technicians and manager with a comprehensive tool that gather, store and visualize performance data at various levels. With that we mean that we can manage data from the different components of a supercomputing environment allowing a sort of multi-resolution analysis.

It has proven to be useful for users that have access to BSC supercomputing facilities. But overall it has also been useful for the HPC application consultants and the support staff within the Operations Department. We have been able to detect performance drops in applications or come up with hypothesis that are later confirmed with other tools like Paraver. Besides, we have been able to obtain important statistical data which can be taken into account in future procurements or to set up more efficient policies when establishing users limits or batch system limits.

We plan to continue developing the tool in order to enhance it and make it more complete and user friendly. In addition, we plan to include more low level metrics according to the feedback received. For example it could be useful to gather more information related to I/O associated to a running application.

One of the points we consider to be important to continue working on is the optimization of the database. The goal is to

speed up queries, scalability and foresee further growth in tables sizes due to the increment of jobs monitored and/or the addition of new metrics or features.

Another point on which we are already working, but is not enough mature yet to be included in this paper, is the development of an advanced analysis user interface. The database holds a lot of information and the user can extract a myriad of data but through complex queries that are not always easy to build for the average user. For this reason we want to extend the current basic analysis and build a new interface to make it easy for users to do some data mining taking advantage of modern web technologies. Despite this latter advanced user interface, there are some points that are suitable to be enhanced in the current interface to improve flexibility and usability. For example, one thing to add is the possibility to have multiple charts opened at the same time.

We have developed this tool with the idea that it could be ported to other environments and batch systems. So far only SLURM is supported but due to its modularity, we plan to make some developments to be able to integrate APE in other systems.

To conclude, we would like to remark that the results of this work showed us that it is feasible and worthwhile to develop an Automatic Performance Engine at multi-cluster level to centralize the monitoring of different aspects in a supercomputer infrastructure. Of course, this increases the complexity of data management and for this reason it was needed to provide an easy way to access and describe the desired queries. We want to contribute to help in the decision making and software consultancy for HPC professionals in supercomputer environments.

ACKNOWLEDGMENT

We thank all the staff of the Operations Department for their valuable comments and support to this project, specially to Jorge Rodriguez and Montserrat Gonzalez for their advice and ideas. We would also like to thank Phillip Mucci for his contribution to integrate PAPI and Papiex in MareNostrum and Daniel Ahlin for sharing his knowledge on PerfMiner.

REFERENCES

[1] S. E. Koonin. (2011) The US Department of Energy website (Office of Science). [Online]. Available: http://science.energy.gov/news/in-the-news/2011/03-28-11/

[2] (2011) PRACE website. [Online]. Available: http://www.prace-project.eu/about-prace

[3] M. Feldman. (2011) HPCWire. [Online]. Available: http://www.hpcwire.com/hpcwire/2011-04-26/top_chinese_supercomputers_point_the_way_to_aggressive_hpc_strategy.html

[4] (2010) MareNostrum architecture [Online]. Available: http://www.bsc.es/plantillaA.php?cat_id=200

[5] *P. J. Mucci, D. Ahlin, J. Danielsson, P. Ekman, and L. Malinowski, "*PerfMiner: Cluster-Wide Collection, Storage and Presentation of Application Level Hardware Performance Data", in *Proceedings of 2005 European Conference on Parallel Computers (Euro-Par), 2005, pages 124–133.*

[6] *Myrinet Express (MX): A High Performance, Low-Level, Message-Passing Interface for Myrinet*, Myricom, 2006.

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

[8] P. J. Mucci (2007) PapiEx - Command line/library utility to measure hardware performance counters with PAPI [Online]. Available: http://icl.cs.utk.edu/~mucci/papiex/ papiex.html

[9] M. Jette and M. Grondona, "SLURM: Simple Linux Utility for Resource Management", in *Proceedings of ClusterWorld Conference and Expo*, San Jose, California, June 2003.

[10] V. Pillet, J. Labarta, T. Cortés, and S. Girona. "Paraver, A Tool to Visualize and Analyze Parallel Code", in *Proceedings of WoTUG-18: Transputer and occam Developments*, pages 17–31. IOSPress, March 1995.

[11] B Hess, C Kutzner, D van der Spoel, and E Lindahl. "Gromacs 4: Algorithms forhighly efficient, load-balanced, and scalable molecular simulation". *J. Chem. Theory Comput.*, vol. 4 (3), pp 435–447, 2008.

[12] C. Simarro and D. Vicente, "Experiences on porting Gromacs to the hybrid parallel models MPI/OpenMP and MPI/SMPSs" in *The 17th meeting of ScicomP, the IBM HPC Systems Scientific Computing User Group*, 2011, Accepted for publication.

[13] *GNU gprof*, J Fenlason, R Stallman - GNU Free Software Foundation, 1998.

[14] (2010) mpiP: Lightweight, Scalable MPI Profiling [Online]. Available: http://mpip.sourceforge.net/

[15] D. Klepacki, The IBM High Performance Computing Toolkit, IBM, 2004.

[16] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker and B. Mohr, "The SCALASCA Performance Toolset Architecture", in *Proceedings of the International Workshop on Scalable Tools for High-End Computing (STHEC), Kos, Greece*, pages 51-65, June 2008.

[17] R. Kufrin, "PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux", in *6th International Conference on Linux Clusters: The HPC Revolution 2005*. Chapel Hill, NC. April 2005.

[18] S. Shende and AD Malony, "TAU: The TAU Parallel Performance System", *International Journal of High Performance Computing Applications*, vol. 20(2), pp. 287–331, 2006.

[19] W.E. Nagel, A. Arnold, M. Weber, H.C. Hoppe, and K. Solchenbach, "VAMPIR: Visualization and Analysis of MPI Resources", *Supercomputer*, vol. 12(1), pp. 69–80, 1996.

[20] (2009) IPM: Integrated Performance Monitoring [Online]. Available: http://ipm-hpc.sourceforge.net/