Computer Science – Research and Development manuscript No.
(will be inserted by the editor)

Monitoring Energy Consumption With SIOX

Autonomous Monitoring Triggered by Abnormal Energy Consumption

Julian M. Kunkel · Alvaro Aguilera · Nathanael Hübbe · Marc Wiedemann · Michaela Zimmer

Abstract In the face of the growing complexity of HPC systems, their growing energy costs, and the increasing difficulty to run applications efficiently, a number of monitoring tools have been developed during the last years. SIOX is one such endeavor, with a uniquely holistic approach: Not only does it aim to record a certain kind of data, but to make all relevant data available for analysis and optimization. Among other sources, this encompasses data from hardware energy counters and trace data from different hardware/software layers. However, not all data that can be recorded should be recorded. As such, SIOX needs good heuristics to determine when and what data needs to be collected, and the energy consumption can provide an important signal about when the system is in a state that deserves closer attention. In this paper, we show that SIOX can use Likwid to collect and report the energy consumption of applications, and present how this data can be visualized using SIOX's web-interface. Furthermore, we outline how SIOX can use this information to intelligently adjust the amount of data it collects, allowing it to reduce the monitoring overhead while still providing complete information about critical situations.

Julian M. Kunkel
DKRZ GmbH
Hamburg, Germany
E-mail: kunkel@dkrz.de

Alvaro Aguilera
ZIH, TU Dresden
Dresden, Germany
E-mail: alvaro.aguilera@tu-dresden.de

Hübbe—Wiedemann—Zimmer
University of Hamburg
Hamburg, Germany
E-mail:
{huebbe, wiedemann, fzimmer}@informatik.uni-hamburg.de

Keywords Parallel I/O, Energy Consumption, Monitoring, Analysis, Optimization

1 Introduction

Analysis of performance and energy consumption of parallel applications is a challenging task. In principle, two different points of view can be distinguished: From the user's perspective, the goal is to analyze an application in order to improve its performance or energy footprint. From the perspective of a data center, gaining an overview of the performance and power characteristics of the applications is key to understanding usage and guiding future investments in machines and support staff to optimize the applications.

Analysis of parallel I/O is one of the most complex topics in HPC, as it involves different and potentially heterogeneous hardware components. Moreover, to potentially improve the performance of slow backend storage, these components utilize a variety of individual optimizations that lead to a complex interplay between them. The Scalable I/O for Extreme Performance (SIOX) project provides a versatile environment for monitoring I/O activities and learning from the gained information [13]. The ultimate goal of SIOX is to automatically suggest and apply performance optimizations, as well as to assist in locating and diagnosing performance problems.

The contributions of this paper are:

1) We introduce a Likwid-based plug-in that enables SIOX to track both the application's and system-wide energy consumption.
2) We describe SIOX's front-end tool for the visualization, correlation and analysis of application and system behavior.

3) We present strategies to restrict monitoring to phases of abnormal energy consumption.

This paper is structured as follows: Section 2 sketches the state of the art in I/O performance analysis. The modular architecture and implementation of SIOX is introduced in Section 3. Section 4 describes tools to analyze and visualize instrumented applications. Section 5 introduces the hierarchical control approach to intelligently monitor, and to react upon relevant activities. Evaluation procedures are discussed in Section 6. Finally, ongoing and future work is discussed in Section 7.

2 Related Work

There are several key aspects relevant to this paper: Monitoring tools to capture energy consumption, analysis and characterization of I/O access patterns, intelligent monitoring and anomaly detection with autonomous reaction to abnormal behavior.

Various solutions to measuring power and energy consumption in HPC nodes have been developed. Approaches to measure power consumption usually use power meters or microcontroller-based meters, measuring the current and voltage of external AC outside the platform or internal DC power lines using the Hall effect, therewith, capturing the power profile of a full node [1]. Recent processors have built-in energy consumption measuring capabilities. Starting with the Sandy Bridge processors, for example, Intel provides the RAPL (Running Average Power Limit) interface [17]. However, these internal performance meters often cannot match the precision of external power meters [6]. The correlation between the RAPL interface measurement and the reference measurement depends on the workload type. Libraries such as *pmlib* [1] and light-weight tools such as Likwid [21] or interfaces such as PAPI [22] can use the RAPL interface to query the energy consumption. Likwid controls and samples the performance counters of the microprocessor with low overhead, extracting information like memory bandwidth, and power and energy consumption.

There are several tools which use these interfaces and relate energy or power metrics with application behavior. The Vampir tool-set [12] can be used to capture power consumption in Watts and visualize application traces together with energy statistics [6]. In *pmlib*, different energy states can be distinguished and measured using the Extrae capture tool and the visualization tool Paraver. Another energy monitoring tool named FEPA (Flexible Framework for Energy and Performance Analysis of Highly Parallel Applications) is currently under development [3]. The system uses simple locking mechanisms to ensure that only one module measures at any time.

Analysis and characterization of I/O access patterns can be performed using the measurement tool Darshan which generates reports on application read and write behavior [4]. It can be deployed in an HPC center to gather system-wide knowledge for all instrumented applications. Event-based systems are playing an increasingly important role in a broad range of application domains, including management, environmental monitoring, information dissemination, autonomic computing, collaborative working and learning [8].

Intelligent monitoring involves observing and guiding the behavior of a system toward a predefined objective. Generic functional requirements for the task of intelligent monitoring include the integration of perception, reasoning, and action as well as that of multiple reasoning activities, reasoning about complex, time-varying systems, and the coordination of multiple response modes [7]. Using ontologies permits the integration of information sources for intelligent monitoring; to be able to react to a behavior and activities of a complex system, a semantic reasoning framework can be used [18]. Approximate monitoring of complex dynamic systems using clustered subsystems helps to decrease the exponentially growing inference. Reasoning may be done only at the onset of interesting events, and extended through learning about the model parameters to be incorporated. A reactive answer set allows for reasoning on real-time dynamic systems running online [5].

Of the approaches interpreting system metrics, `Cluebox` by Sandeep et al. [19] points out the system counters most likely involved in the problem by principal feature analysis, ranking by decision trees and subsequent clustering. All anomaly detection systems always have the inherent problem of false positives. Both normal and abnormal conditions can occasionally result in the same observable characteristic. Building groups with those identifiers proven to be most effective for a certain issue, results in a more precise detection and reduction of false positives [11]. Identification and detection of nonlinear energy anomalies of system-related events can be performed using thresholds and a multivariate transformation based on multicasting information, monitoring tools and intensity matrices [16]. Usually, when using statistics to detect anomalies, a categorization is performed in advance to assemble heuristics in groups [9]. Finally, rather than indicating failures, anomalies can also expose abnormal situations or configurations [16]. In all these anomaly occasions, it is desirable to have an intelligent system that overcomes false positives and efficiently reacts to the anomalies' behavior using a classification system.

An early classification of I/O access patterns is presented in the work of Madhyastha and Reed [14] by means of feed-forward neural networks and hidden Markov models. Using this approach, the I/O patterns of higher level applications are inferred and looked up in a table to determine the file system policy to be set for the next accesses. Later approaches store their results in a database. Problem analysis benefits from past diagnostic efforts, possibly applying known corrective reactions to recurring problems. The reaction can also be based on classifications using I/O signatures [2], where a signature notation is introduced to represent the observed patterns. A toolkit generates these signatures automatically and feeds the information to a prefetching thread that reads the applications' signatures, adjusts them and reissues their calls at runtime for improved I/O. There are several energy saving schemes for application runs based on pattern classification and reorganization of active data parts. In this way, reduced energy consumption is achieved by decrementing the access time to active data, while increasing it for cold data [24].

SIOX [13] unites all the existing solutions and yet differs, because it is built with the idea in mind to automatically analyze system behavior and intelligently optimize I/O operations using a multitude of approaches. Among them are the injection of optimization hints, as well as the querying of optimal parameters to support decision making in existing libraries. Fusing the capabilities of existing tools into one framework and combining it with intelligent monitoring, we have taken a step forward integrating energy consumption awareness into SIOX's capabilities for abnormal behavior recognition and adaptive monitoring.

3 The SIOX Architecture

SIOX collects event-based information about the I/O activities taking place at the instrumented software and hardware components of a system, and samples statistics about the resource usage of the nodes where the SIOX daemons are running. Furthermore, SIOX combines online monitoring with offline learning. Monitoring data flows from the instrumented component into the daemon process, and from there into one of SIOX's transaction servers as detailed in Wiedemann et al. [23]. If relevant, the monitoring data is correlated and simplified at the transaction server, and finally sent to the data warehouse for long-term archiving. The recorded information will be analyzed offline to update a knowledge base holding optimized parameter suggestions for common or critical situations. During online operations, these parameters may be queried and used as predefined responses whenever such a situation occurs. The choice of responses to each situation is diverse, ranging from adjusting the monitoring level in the presence of anomalies, to automatically enforcing optimization techniques to achieve better performance, including alerting users and administrators with informative reports about problems and corrective actions. In Zimmer et al. [25], we discussed this workflow in more detail and sketched several modules for anomaly detection.

Another noteworthy capability in SIOX is the possibility to correlate system-wide and application-local monitoring. SIOX was conceived to be very flexible and modular: Upon startup of either a process, component, or daemon, a configuration file is read containing the desired hierarchy of nested modules and plug-ins that are to be used. Several modules offer additional optimization interfaces that can be used by specialized plug-ins to detect anomalies in different ways, and to trigger corrective actions based on the observed activities and system state. Details of the existing modules are given in Kunkel et al. [13]. This paper extends the previous work by introducing concepts to monitor and analyze energy consumption at socket level.

3.1 Monitoring Energy-Efficiency

Figure 1 shows a configuration of modules involved in the measurement and evaluation of energy metrics. The interplay of the modules is explained in brief, starting from the capturing of energy metrics. A more detailed explanation of SIOX's basic modules is given in [13].

StatisticsCollector: This module queries the registered provider plug-ins to retrieve the performance counters every 100 ms and notifies the multiplexer component inside the daemon about the new performance data. The statistics multiplexer, in turn, notifies another set of plug-ins about the availability of new data. If desired, the flow of statistical data can be sent to a database server using a database writer plug-in. Finally, the statistics health anomaly detection plug-in checks the statistics for soundness and flags anomalies.

Likwid: This statistics provider plug-in uses Likwid to retrieve performance counters, including energy counters. To this end, Likwid has been patched to support a light-weight C interface which allows the setup of different counter groups and retrieval of performance values. According to the Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3A [10], the Sandy Bridge processor supports power, energy and time counters with increments of granularity 0.125 W, 15.3 mJ and 976 μs respectively[1]. Since the wraparound of the counters takes significantly longer than the 100 ms polling interval, any overflow is detected correctly.

Reasoner: The reasoner component aggregates reports from different sources and triggers anomaly handlers in case of abnormal behavior. Reasoners are deployed at process, node and system level and form a hierarchy, exchanging state information amongst themselves. This is further explained in Section 5.

Reporters: Upon process termination, reporters receive statistical information from all SIOX modules; this does not only include debugging data, but also performance-relevant information, such as elapsed CPU time and amount of data transferred. The console reporter simply outputs the retrieved information and it's not aware of parallel applications. If an application uses an MPI instrumentation, an MPI reporter module is available that aggregates the statistics over all participating MPI processes and presents one joint report for all of them.

Monitoring activities: The SIOX low-level API forwards observed activities to a layer-specific activity multiplexer. Similar to its statistics counterpart, this multiplexer can dispatch the activity to multiple plugins. Activity forwarders connect the activity multiplexers across layers or processes. At node level, a database writer module may store the activities in a database server; other back-ends are available to record activities into different file formats as well.

3.2 SIOX-induced Overhead

One of the main concerns regarding online tracing applications like SIOX is the performance overhead they introduce in the system. This is specially true when we are trying to determine the energy consumption of the applications monitored without disturbing their energy pattern. SIOX addresses this problem by minimizing its overhead as much as possible: Firstly, the tracing and statistic collection levels are dynamically adapted as needed in order preventing unnecessary logging. Secondly, the daemon is free from heavy computation code with most of its decision-making plug-ins based on simple action tables. Finally, SIOX is configurable and thus allows users and administrators to select the appropriate plug-ins that are beneficial for the application. Empiric experiences have shown that the only significant overhead introduced by SIOX takes place during the initialization of the instrumented applications, where a delay of about 2.5 seconds per instrumented layer was observed. This has been caused by the prototypical implementation for the database topology component and is subject to current optimization efforts. These observations are formally documented in [13].

[1] These values have been queried from the model-specific registers (MSR) and verified.

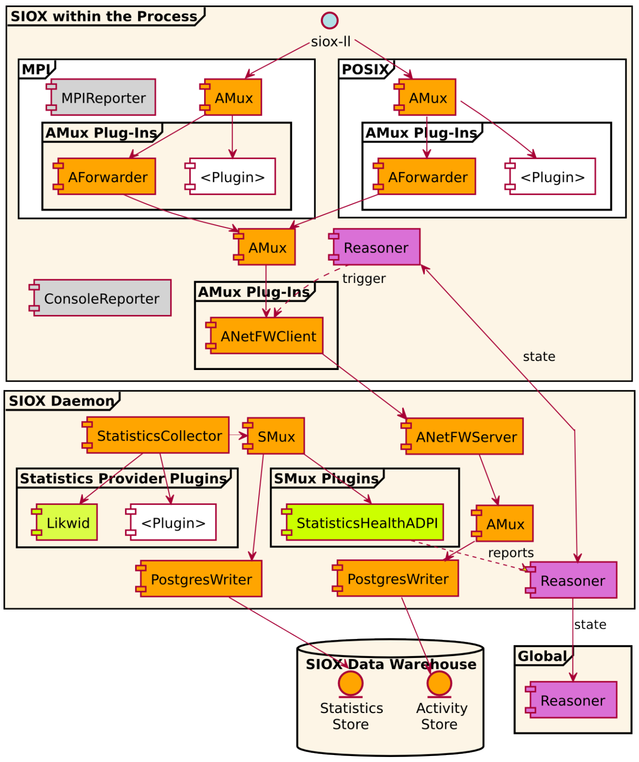

Fig. 1: SIOX configuration for analyzing energy consumption: modules within a process, local daemon, and their interactions.

4 Analyzing Energy Consumption

SIOX offers two methods of accessing collected statistics like, in this case, the energy consumption: First, the sampled flow of statistical information can be sent to a database server and kept there for later analysis; SIOX comes with a convenient web interface to visualize the information stored in this database. Second, at program termination, the reporters provide a short summary of performance characteristics and consumed resources. The former method is considered to work at *system-level* as it focuses on the system-wide analysis. The latter method works at *user-level*, since it is centric to application runs. Remember, statistics are sampled by the SIOX daemons at a dynamically assigned sampling rate appropriate to the current logging level.

As mentioned in Section 3, the set of available statistics is determined by the corresponding plug-ins specified in the configuration file of the SIOX daemon. Typical examples of statistics provided are memory consumption, CPU utilization, network load, different I/O

```
ID       Command
[3666]: ./parabench -e -D pattern0.pbl
[3828]: ./parabench -e -D pattern1.pbl
[3979]: ./parabench -e -D pattern2.pbl
[4156]: ./parabench -e -D pattern3.pbl

ID    Duration  Socket/RAPL
3666  377 s     23605 J
3828  133 s      8279 J
3979  119 s      7453 J
4156  120 s      7541 J
```

Fig. 2: Comparison of the cumulative energy counters for four different runs of Parabench.

statistics like number of blocks read and written, and energy consumption. Moreover, given the flexibility of SIOX's architecture, new statistical modules can be implemented without much effort. System utilization statistics can be divided into two classes: cumulative and consecutive statistics. Summation over consecutive statistics, like the system utilization metrics, is rarely useful. On the other hand, cumulative statistics like energy consumption and CPU time provide proper means of aggregation, and we consider this fact when presenting the information to the users.

4.1 System-Level Analysis

Armed with SIOX's web interface, the user may not only list and analyze the I/O activities his application run has created, but also produce plot diagrams of the statistics collected during the time frame of one or more application executions. Moreover, the knowledge gained can be used by administrators to analyze the system behavior. This permits a quick inspection and comparison of the effects that different parametrizations or algorithm choices have had on the system. The interface leverages the SQL-API of SIOX's transaction server to extract correlated information, so that it can be displayed without additional preprocessing. The information available to the user includes detailed statistics on energy consumption.

Consecutive statistics are shown as plot diagrams over time, like those in Figure 4. Since more than one logged execution can be selected for display, this offers an easy way of qualitative comparison. Cumulative statistics, e.g. the energy consumption in Joules, number of bytes read or written, etc., are additionally aggregated in a summary at the beginning of the page, similar to Figure 2.

4.2 User-Level Analysis

When an application terminates, the reporters are triggered writing information about the application's behavior to the console. Amongst others, the Reasoner manages statistics about the CPU time and energy consumed during the program run, providing direct feedback to the user. Listing 1 shows an example excerpt. The CPU time consumed is taken from "Likwid's Runtime (RDTSC)" counter, which indicates the time the CPU was busy processing instructions. Additional statistics cover the amount of data accessed from a local block storage and the data transferred across the network. Finally, the overall time span during which statistics have been updated is indicated in Line 5.

Listing 1: Example output reported by the Reasoner.

```
1 CONSUMED_CPU_SECONDS = 2.285407
2 CONSUMED_ENERGY_JOULE = 46.924286
3 ACCESSED_IO_BYTES = 23068672
4 TRANSFERRED_NETWORK_BYTES = 6336953
5 OBSERVED_RUNTIME_MS = 2600
```

5 Intelligent Monitoring

In this section, we detail and expand upon the concepts introduced in [25].

To stem the tide of logging information, SIOX employs a tree-shaped hierarchy of "Reasoner" modules, acting as arbiters and, if need be, inhibitors.

First, activities and statistics are collected at the respective multiplexer and further distributed to specialized "anomaly detection plug-ins" (ADPIs) for further investigation. Should any of these deem its input anomalous, it will generate an issue report describing the problem and the values concerned.

Dedicated reasoners exist at process, node and system level, and will regularly exchange status information with their direct neighbors in the hierarchy. At the same interval, each one will poll all its assigned ADPIs for issues witnessed during the last cycle. All are united into a differentiated status report, taking into account local as well as neighboring components' general conditions (see Listing 2). Should any reasoner detect anomalous behavior at this stage, it will trigger another set of specialized plug-ins, each custom-tailored to provide an appropriate response. Also, it will preserve the abnormal state for one more cycle to capture the effect of the abnormally for further inspection. This strategy also guarantees that no actvities are lost, if consecutive cycles are flagged abnormal. Possible actions include archiving all or selected log data from a given time window, automatic adjustment of system parameters, and a message to a human administrator containing detailed information about the problem and its preliminary analysis. It should be noted that in the context of SIOX, the term anomalous behavior may describe unusually good as well as exceptionally bad performance/-consumption: Reproducing the former is as desirable as avoiding the latter during future operations.

By this device, SIOX is able to differentiate bottlenecks due to high system load from those engendered by system faults or misconfigurations.

Anomalies may cover the whole range of system conditions detectable through metrics available to SIOX. As these, too, may be extended via dedicated plug-ins, there is virtually no limit to the auto-diagnostic possibilities of a system running SIOX. At this time, the data sources available include the RAPL interface and several statistics offered by `/proc`.

As statistics vary, so do anomalies detected through them. For this paper, we will list but a few prominent examples for ADPIs possible and already implemented:

- As any system operation will consume energy, a reported value of 0 J will clearly signify a sensor problem.
- For most metrics available, values within the top or bottom 5 % of the possible range mark anomalous behavior worthy of further investigation by the reasoner. The results of a sample implementation are shown in Table 3, under "StatisticsADPI" (S).
- For modern CPUs, power consumption is correlated to the instruction cycles executed. FLOPS and other operation types are available via hardware counters, allowing for fairly accurate estimates of energy used. Even for CPUs for which this data is unavailable, the ratio of energy consumed to CPU time spent will be a constant for any given clock speed, and can easily be estimated from observations. Any major deviation from the estimate may hint at irregularities (see Listing 3). Results for this scheme are shown in Table 3, under "EnergyEfficiencyADPI" (E).
- For long-running applications (e. g., climate models), any major discrepancy between recent consumption and average behavior over the last hours or days will most likely indicate a problem.

Listing 2: Pseudocode for simple reasoning rules. Any k... variables are program parameters.

```
query_current(var nBadAnomalies, var
     ↪ nGoodAnomalies, var nOtherAnomalies)
if (nBadAnomalies + nGoodAnomalies +
     ↪ nOtherAnomalies > 0)
   if (nBadAnomalies > kThreshold &&
        ↪ nBadAnomalies > kRatio *
        ↪ nGoodAnomalies)
      currentState = ABNORMAL_BAD
   else if (nGoodAnomalies > kThreshold &&
        ↪ nGoodAnomalies > kRatio *
        ↪ nBadAnomalies)
      currentState = ABNORMAL_GOOD
   else
      currentState = ABNORMAL_OTHER
...
if (currentState != GOOD && currentState !=
     ↪ ABNORMAL_GOOD)
   for (c in categories)
      if (utilization[c] > kMaxLoad)
         raise_issue("Overloaded", c)
```

Listing 3: Pseudocode for sample ADPI algorithm.

```
query_current(var cpuConsumed)
query_current(var energyConsumed)
currentEfficiency = cpuConsumed / energyConsumed
nValues = nValues + 1
update_distribution_estimate(currentEfficiency)
query_estimated_distribution(var mean, var stddev)
if (nValues > nStabilizationLimit)
   if (currentEfficiency > mean + stddev)
      flag_anomaly(ABNORMAL_GOOD)
   if (currentEfficiency < mean - stddev)
      flag_anomaly(ABNORMAL_BAD)
```

6 Evaluation

6.1 Applications

Parabench is a programmable benchmark that can mimic different temporal and spatial access patterns [15]. In this experiment, we analyze the four levels of access in MPI [20] using a strided access pattern. The four levels of access are described in Table 1 and defined by two orthogonal aspects: collective vs. independent I/O, and contiguous vs. non-contiguous I/O. In the spatial access pattern, each process accesses 6400 blocks with a size of 100 KB, which accumulate in a shared file of 5 GByte.

Level 0:	non-collective, contiguous
Level 1:	collective, contiguous
Level 2:	non-collective, non-contiguous
Level 3:	collective, non-contiguous

Table 1: Parabench's access levels.

ICON is a climate model developed by the Max Planck Institute for Meteorology and the German National Weather Service. ICON is under active development with one of its main features being that it uses an icosahedral grid to discretize the earth surface. This allows the model to avoid the pole problem inherent to all longitude/latitude based grids. Output is written along a space filling curve that traverses the grid and fuses the two surface dimensions of the earth into one grid coordinate. Height and time are used as two additional dimensions. By default, all data is written into one common file. This file contains a two- or three-dimensional array for each physical variable that is recorded during

```
define pattern {"pattern0", 2, 6400, (100 * 1024), 0};
time["MPI-IO test"] {
        time["pwrite-lvl0"] pwrite("output.dat", "pattern0");
        barrier;
        clearcache;
        time["pread-lvl0"] pread("output.dat", "pattern0", "world");
        barrier;
        pdelete("output.dat");
        barrier;
}
```

Fig. 3: Parabench's test definition file for level 0 access.

the run. From time to time, the output file is closed and a new one is started to avoid intractably large files. In addition to the model output, a restart file is written at fixed intervals.

6.2 Analysis of Energy Consumption and Behavior

The Parabench tests were conducted using the local storage of the Sandy system hosted at German Climate Research Center (DKRZ) and 8 Open MPI processes. The test definition file for the access level 0 is shown in Figure 3. The test definitions for levels 1 to 3 are almost identical, only differing in the pattern number. The `clearcache` instruction after the write barrier clears the system cache in the traditional way by setting `/proc/sys/vm/drop_caches` to the value 3.

Table 2 shows a summary of the runtime as well as energy consumption of the process with rank 0 for all four access levels. Figure 4 shows the energy consumption for every access level as obtained by Likwid through the RAPL interface. All four access levels showed a similar energy consumption pattern. The test using a non-collective, contiguous pattern required around three times the runtime (and thus also the energy) required by the other levels since it doesn't profit from MPI I/O optimizations like collective operations and data sieving.

Level	Runtime [s]	Energy [J]	Activities
0	377	23605	21335
1	133	8279	17390
2	119	7453	11704
3	120	7541	2826

Table 2: Parabench runtime and energy consumption per process.

6.3 Intelligent Monitoring

Two of the anomaly detection plug-ins we developed were evaluated by running ICON with either none of them, the StatisticsADPI, the EnergyEfficiencyADPI

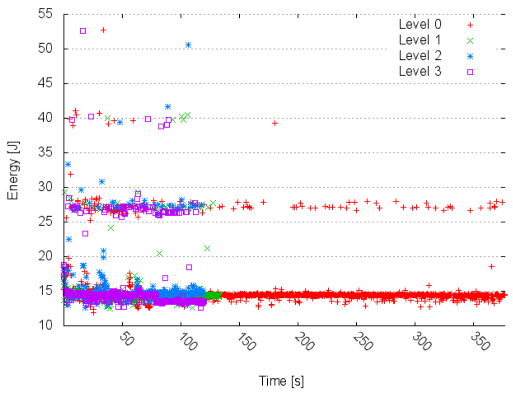

Fig. 4: Energy consumption of the Parabench test for the different access levels.

or both together. In the last configuration, an abnormal state is triggered if any of the plug-ins detects an abnormal state. Table 3 shows the runtime, the percentage in which abnormal phases are detected and the number of activities stored. It can be observed that the number of stored activities decreases from 100% to between 7 % and 18 % when the ADPIs are used. The StatisticsADPI fires rarely, while the EnergyEfficiencyADPI rates a state abnormal twice as often. In contrast to the activities stored, the number of phases which are classified to behave abnormally varies between 0.2 % and 88 %.

The StatisticsADPI detects only a few abnormal phases and remains quite stable with its detection rate. Presumably, the EnergyEfficiencyADPI identifies many computational and communication phases with inconsistent energy behavior – since these do not generate I/O activities there is nothing to record there. Still, in both cases the number of activities is quite stable; this demonstrates that I/O patterns with interesting energy consumption are preserved while irrelevant patterns are discarded.

Already these first prototypical plug-ins show the benefit of the approach. While the intelligent monitoring is only a first step in the analysis chain, the number of activities can be reduced by a factor of 5 to 10. This will decrease the burden of the monitoring system and storage system and increase the scalability significantly – with the right configuration and plug-ins.

7 Summary and Conclusion

We have shown how information from hardware energy counters can be used to focus monitoring on system states that deserve attention. This way, we can provide the system administrators with detailed data

Table 3: Recorded activities for the ICON runs. S: StatisticsADPI only, E: EnergyEfficiencyADPI only, S&E: both.

Application	Runtime	**Abnormal phases**			Activities	**Activities stored**		
		S	E	S&E		S	E	S&E
Average	553 s	0.3 %	5.8 %	39.4 %	15,297	6.9 %	12.7 %	11.3 %
Min	552 s	0.2 %	4.6 %	3.0 %	15,297	6.7 %	6.5 %	6.5 %
Max	553 s	0.4 %	7.2 %	87.7 %	15,297	7.4 %	18.4 %	18.4 %

about critical situations while omitting less valuable data to reduce the overhead of data collection. Even though the monitoring strategy of SIOX proves effective at focusing monitoring efforts and thus reducing overhead, we are only beginning to take advantage of the intelligent monitoring and self optimization provided by this framework.

Opportunities for further research abound: Which algorithms are suited best to focusing our attention on the data most valuable? How can this information be best put to use, and what kind of self-optimizations does it facilitate? Which of these are the most efficient at reducing the amount of energy consumed by high-performance computing systems? And finally, what additional analyses and feedback should a system like SIOX provide to both users and system administrators to help them unlock their machine's full potential?

Acknowledgements We want to express our gratitude to the German Aerospace Center (DLR) as responsible project agency and to the Federal Ministry of Education and Research (BMBF) for the financial support under grant `01IH11008 A-C`.

References

1. Barrachina, S., Barreda, M., Catalán, S., Dolz, M.F., Fabregat, G., Mayo, R., Quintana-Ortí, E.S.: An integrated framework for power-performance analysis of parallel scientific workloads. In: ENERGY 2013, The Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies, pp. 114–119 (2013)
2. Byna, S., Chen, Y., Sun, X.H., Thakur, R., Gropp, W.: Parallel I/O prefetching using MPI file caching and I/O signatures. In: Proceedings of the Conference on Supercomputing, SC '08, pp. 1–12. IEEE Press, Piscataway, NJ, USA (2008)
3. Carias, C.G., Hesse, W., Navarrete, C., Brehm, M., Treibig, J.: A flexible framework for energy and performance analysis. inSiDE Journal, Vol. 11, No. 2 (2013)
4. Carns, P.H., Harms, K., Allcock, W.E., Bacon, C., Lang, S., Latham, R., Ross, R.B.: Understanding and improving computational science storage access through continuous characterization. In: Proc. 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST) (2011)
5. Gebser, M., Grote, T., Kaminski, R., Schaub, T.: Reactive answer set programming. In: Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'11, pp. 54–66. Springer-Verlag, Berlin, Heidelberg (2011)
6. Hackenberg, D., Ilsche, T., Schone, R., Molka, D., Schmidt, M., Nagel, W.E.: Power measurement techniques on standard compute nodes: A quantitative comparison. 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) **0**, 194–204 (2013)
7. Hayes-Roth, B., Washington, R., Hewett, R., Hewett, M., Seiver, A.: Intelligent monitoring and control. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI, pp. 243–249. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1989)
8. Helmer, S., Poulovassilis, A., Xhafa, F.: Reasoning in Event-Based Distributed Systems. Springer (2013)
9. Himura, Y., Fukuda, K., Cho, K., Esaki, H.: An automatic and dynamic parameter tuning of a statistic-based anomaly detection algorithm. In: Proceedings of the 2009 IEEE International Conference on Communications, ICC'09, pp. 1003–1008. IEEE Press, Piscataway, NJ, USA (2009)
10. Intel Corporation: Intel 64 and IA-32 architectures software developer's manual. Volume 3a. `http://download.intel.com/design/processor/manuals/253668.pdf` (2011)
11. Kind, A., Stoecklin, M.P., Dimitropoulos, X.A.: Histogram-based traffic anomaly detection. IEEE Transactions on Network and Service Management **6**(2), 110–121 (2009)
12. Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M., Nagel, W.: The Vampir performance analysis tool-set. In: M. Resch, R. Keller, V. Himmler, B. Krammer, A. Schulz (eds.) Tools for High Performance Computing, pp. 139–155. Springer Berlin Heidelberg (2008)
13. Kunkel, J., Zimmer, M., Hübbe, N., Aguilera, A., Mickler, H., Wang, X., Chut, A., Bönisch, T., Lüttgau, J., Michel, R., Weging, J.: The SIOX architecture – coupling automatic monitoring and optimization of parallel I/O. In: Supercomputing, no. 8488 in Lecture Notes in Computer Science. Springer, Berlin, Heidelberg (2014 – to-appear)
14. Madhyastha, T., Reed, D.: Learning to classify parallel input/output access patterns. Parallel and Distributed Systems, IEEE Transactions on **13**(8), 802–813 (2002)
15. Mordvinova, O., Runz, D., Kunkel, J., Ludwig, T.: I/O performance evaluation with Parabench – programmable I/O benchmark. Procedia Computer Science pp. 2119–2128 (2010)
16. Ostrouchov, G., Naughton, T., Engelmann, C., Vallee, G., Scott, S.: Nonparametric multivariate anomaly analysis in support of hpc resilience. In: E-Science Workshops, 2009 5th IEEE International Conference, pp. 80–85 (2009)
17. Rotem, E., Naveh, A., Ananthakrishnan, A., Rajwan, D., Weissmann, E.: Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. IEEE Micro **32**(2), 20–27 (2012)
18. Sabri, L., Chibani, A., Amirat, Y., Zarri, G.p.: Semantic reasoning framework to supervise and manage contexts and objects in pervasive computing environments. In: Proceedings of the 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications, WAINA, pp. 47–52. IEEE Computer Society, Washington, DC, USA (2011)

19. Sandeep, S.R., Swapna, M., Niranjan, T., Susarla, S., Nandi, S.: CLUEBOX: a performance log analyzer for automated troubleshooting. In: Proceedings of the First USENIX conference on Analysis of system logs, WASL'08. USENIX Association, Berkeley, CA, USA (2008). URL `http://dl.acm.org/citation.cfm?id=1855886.1855887`
20. Thakur, R., Gropp, W., Lusk, E.: Optimizing noncontiguous accesses in MPI/IO. Parallel Computing **28**(1), 83 – 105 (2002)
21. Treibig, J., Hager, G., Wellein, G.: Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In: Parallel Processing Workshops (ICPPW), 2010 39th International Conference on, pp. 207–216. IEEE (2010)
22. Weaver, V., Johnson, M., Kasichayanula, K., Ralph, J., Luszczek, P., Terpstra, D., Moore, S.: Measuring energy and power with PAPI. In: Parallel Processing Workshops (ICPPW), 2012 41st International Conference on, pp. 262–268 (2012)
23. Wiedemann, M.C., Kunkel, J., Zimmer, M., Ludwig, T., Resch, M., Bönisch, T., Wang, X., Chut, A., Aguilera, A., Nagel, W., Kluge, M., Mickler, H.: Towards I/O analysis of HPC systems and a generic architecture to collect access patterns. Computer Science - Research and Development pp. 1–11 (2012)
24. Yin, Y., Li, J., He, J., Sun, X.H., Thakur, R.: Pattern-direct and layout-aware replication scheme for parallel I/O systems. In: Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, pp. 345–356 (2013)
25. Zimmer, M., Kunkel, J., Ludwig, T.: Towards self-optimization in HPC I/O. In: J.M. Kunkel, T. Ludwig, H.W. Meuer (eds.) Supercomputing, no. 7905 in Lecture Notes in Computer Science, pp. 422–434. Springer, Berlin, Heidelberg (2013)