A Framework for Region-based Instrumentation of Energy Consumption of Software Programs

Simon Ostermann, Thomas S. Eiter, Vlad Nae, and Radu Prodan
Institute of Computer Science, University of Innsbruck, Austria

***Abstract*—Energy efficiency has become a key issue in computer science related research and development over the last years. While most approaches focus either on hardware or on software, we propose a solution incorporating both hardware and software enabling the measurement the energy consumption of code segments executed on physical machines. Our novel approach to energy measurement and instrumentation allows for both state-of-the-art offline analysis, and innovative online measurements associated with the code being executed. We present our modular architecture which shields the users from in-depth knowledge of their energy measurement hardware, and allows the development of code for measurement and instrumentation independent of each instruments' proprietary interface. Finally, we propose an efficient method for increasing the accuracy of measurements for low sampling rate measurement devices.**

I. Introduction

Energy efficiency has become a key issue in computer science research and development over the last years. Driving forces behind this are the ever growing energy consumption of computers worldwide, the trend towards mobile devices, and the Green IT initiative, aiming to save ecological resources during the lifetime of a product. According to [1], the global energy consumption of data centres has more than doubled between 2000 and 2005. This sharp rise has slowed down to approximately 56% between 2005 and 2010, partly due to the technical improvements and partly due to the financial crisis starting from 2008. As illustrated in Figure 1, the data centres were responsible for about 0.5% (75 billion kWh) of the world's energy consumption year 2000, increasing up to 1.0% (150 billion kWh) in 2005, and reaching between 1.1% and 1.5% (200 to 275 billion kWh) in 2010. [1, p. 6]. Another reason for the increasing importance of energy is the trend towards mobile devices such as smartphones and tablets that conquered the market over the last years. For example, in Germany, U.K. and France there were more customers using mobile devices than personal computers in 2011 [2, p. 2].

Regarding energy consumption in computer science, Green IT has become a major initiative whose major goal is to save ecological resources during lifetime of a hardware or software product (or a combination of both [3]), covering production, active runtime, recycling and biodegradability. Focusing on energy, the aim is to consume as little energy as possible during the product life-cycle. To curb the energy consumption and improve energy efficiency, many techniques and technologies have been invented in the last decade, most of them focusing either on hardware or on software. Besides the hardware, the software application plays a key role concerning energy consumption, as it defines which hardware components are used, when they are used, and to what degree. Thus, the total energy consumption of a code execution on a specific machine depends on both the machine and on the code characteristics. To achieve the best possible energy efficiency, one of the promising approaches is to design self-tuning codes that receive online feedback about their energy consumption and allow for fast and precise parameter adjustments for improving their energy efficiency. However, energy measurements in the field of data centres are coarse grained (e.g. power-rail level, rack level), difficult to collect (e.g. proprietary interfaces, sometimes not available to the users), and are typically not associated to the codes being executed. Moreover, if energy profiling data is available and can be associated to the code, it is only available post-mortem (i.e. after the code has been executed) and not online during its execution.

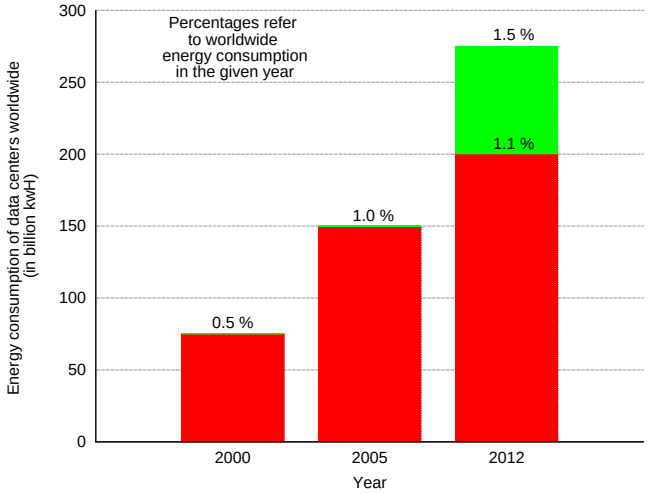

Fig. 1. Energy consumption of data centres worldwide, based on [1, p. 13].

To address these issues we propose in this paper an energy instrumentation and measurement framework which allows implementation and easy deployment of self-tuning codes in data centres. Our framework allows for both offline and online (i.e. during execution) analyses by providing detailed energy consumption data for software programs with a region-level granularity. Our modular architecture shields the users from needing in-depth knowledge of their energy measurement hardware, and allows the development of measurement and instrumentation code independent of the instrument's proprietary interface. Finally, we propose an efficient method for increasing the accuracy of measurements for low sampling rate measurement devices (e.g. 0.5 – 10 Hertz).

The next section describes the framework architecture,

 DOI: https://doi.org/10.1109/IECON.2013.6699897

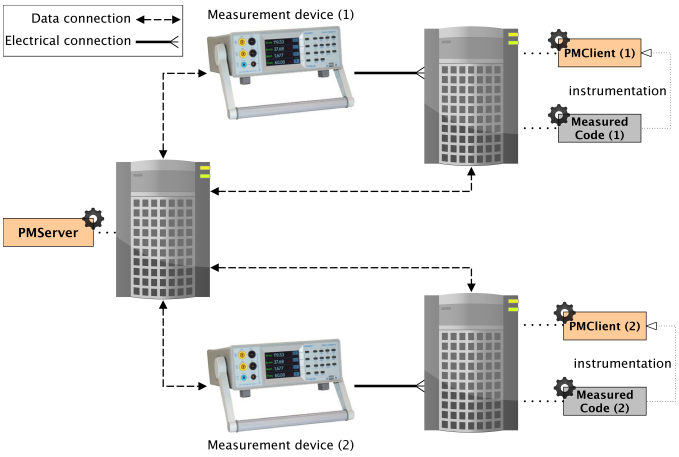

Fig. 2. Framework architecture with one server managing two sessions, each comprising one measurement device, one client, and the application code.

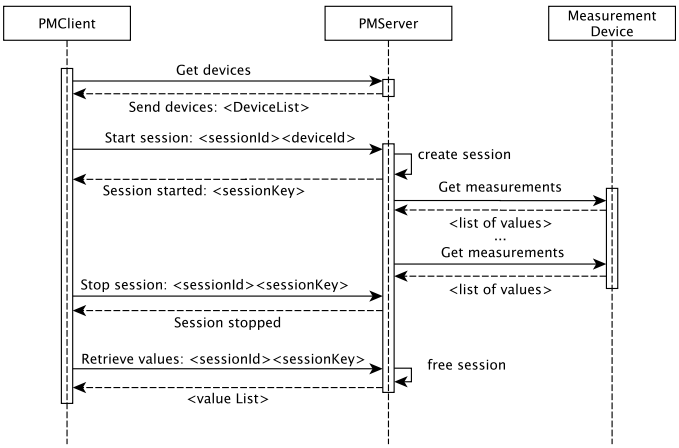

Fig. 3. Interaction protocol between `PMClient`, `PMServer`, and measurement device during a `PMSession` life-cycle.

followed by the methodology for energy consumption computation in Section III. Section IV presents techniques for improving the accuracy of energy measurements. Section V discusses the related work and Section VI concludes the paper.

II. Architecture

We propose an energy measurement framework based on a client-server architecture which allows for easy extensions to different programming languages, as only the client part has to be ported. The architecture allows the server to be run on a different physical machine and the deployment of lightweight clients with minimum energy overhead intrusion. A diagram of the proposed architecture with a server and two measurement sessions is presented in Figure 2. A server program (`PMServer`) is continuously running on a service machine, communicating and managing all measurement devices such as Voltech PM1000+. The other machines electrically connected to the measurement devices run the code to be measured, which uses the `PMClient` to retrieve runtime online measurements from the `PMServer`. In the following we present the role of each component and the interaction protocols.

A. Measurement server

The `PMServer` is responsible for managing the direct communication with the measurement devices, hiding the different access methods and data representations of different device types and allowing for easy client access to the measurement results via a predefined interface. The server runs on a dedicated machine (i.e. not the machine being instrumented), records power and/or energy measurements from different device types, and makes them available to the clients. To request data from the measurement devices, the server implements the communication protocols provided and supported by the respective device types as decoupled modules. To maintain the light coupling of components, the communication and data exchange with the client is based on a separate communication protocol built over TCP/IP, and independent of the proprietary device (see Section II-D). For concurrent handling of multiple measurement devices, multiple clients, and multiple requests, we employ encapsulated measurement sessions. When a new measurement session is initiated, the requested measurement device is reserved and a session key used for further session-related requests and authentication by the session owner is generated. When a session is ended and its measurement results are retrieved by the session owner, the session is deleted and the measurement device is freed.

B. Measurement client

The `PMClient` offers a unified code instrumentation interface, allowing to start and stop measurements on specific devices connected to the server, and to collect and process the corresponding data via simple function calls. The client is the central piece in our architecture which allows the online collection of measurement data with minimal disruption to the measured process. Therefore, it is purposely designed as lightweight code, offloading all instrumentation data post-processing to the server to minimise its influence on the measurements. The client is implemented as a library which can be utilised by the code being instrumented. For increased flexibility, the client library provides multiple interfaces in different programming languages (e.g. C++, C, Java). The client communicates with the server using the communication protocol introduced in Section II-D.

C. Measurement session

The `PMSession` associates the power/energy measurements to a specific code region. A `PMSession` has a life-cycle determined by its associated code region and comprises a sequence of procedures designed to initiate, terminate and collect measurement data from the device. A typical `PMSession` is presented in Figure 3. The first step any instrumented code performs in its initialization phase consists of collecting information about the available measurement devices (the `get devices` call) and select the relevant one. Then the relevant regions of code are instrumented for energy consumption encompassed in a measurement session (line 5) and surrounded by `start` and `stop` session calls (lines 7 and 11). During the `PMSession`'s life-cycle, the `PMServer` continuously collects the measurement data from the measurement device, which it aggregates and delivers to the `PMClient` after the end of the session (line 15).

Listing 1. Code instrumentation example using the C++ interface.

```
1 #include <CPPInterface.h> // C++ Interface
2
3 int main(int argc, char ** argv){
4   // Defining parameters for the session
5   pmCreateNewSession("sessionId", "10.0.0.1", 5025);
6
7   pmStartSession(0);  // Start session on device 0
8
9   /* Insert HERE the code to measure */
10
11  pmStopSession();
12
13  pmRetrieveResults();
14
15  /* Use the desired measurements */
16  double consumption = pmGetEnergyConsumption();
17
18  pmDeleteSession();
19  ...
20 }
```

D. Client-server communication protocol

The communication protocol for data exchange between server and client is built upon TCP/IP for platform and language independence. The basic format of a client request is <HEAD>:<TAIL>, where HEAD contains a code defining the action to be performed on the server, and TAIL contains the corresponding parameters separated by a semicolon. The same message format is used for server answers, where HEAD indicates if the action succeeded or an error occurred, and TAIL is used for further action-related data.

E. Instrumentation interface

The instrumentation interface offers the methods needed to discover the available measurement devices managed by a server, to start, stop, suspend, and resume a measurement session, and to collect, access, and process the measurement data. The interface is designed for simplicity of use that hides many background operations from the user (e.g. server communication, result post-processing). An example of a code instrumentation using the C++ interface is given in Listing 1. The intialisation phase consists of creating the measurement session with a certain ID and indicating the connection details for the PMServer (line 5). The measurement begins with the pmStartSession call using the measurement device with the specified ID (line 7). The code to be instrumented should be placed immediately after the session start call. The measurements are stopped with the pmStopSession call (line 11), then the values are retrieved from the server (line 13). Finally, the instrumentation data is available locally and it can be used, as exemplified in line 16, and eventually the session can be deleted and the data released (line 18).

III. Energy consumption computation

Some power measurement devices only provide the basic instantaneous power consumption readings. Thus, in order to provide energy measurements even for these legacy devices, our framework needs to record the power consumption during the PMSession and eventually compute the energy consumption.

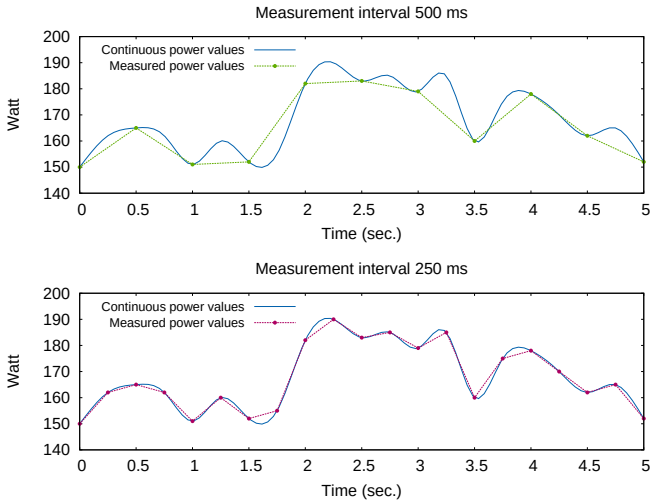

Fig. 4. Accuracy deviation using different measurement intervals.

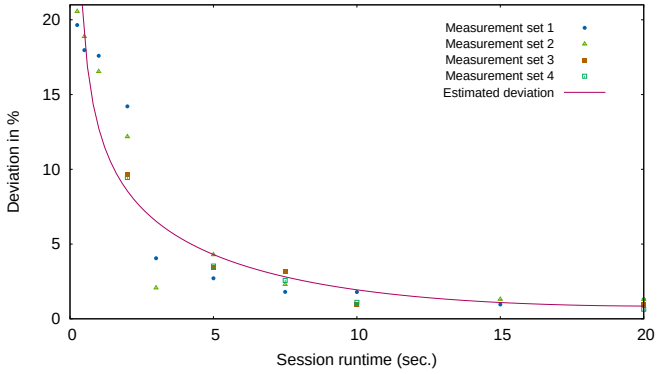

Fig. 5. Average deviation of calculated versus measured energy consumption.

To calculate the exact energy consumption:

$$E = \int_{t_0}^{t_n} P \cdot dt \tag{1}$$

between the start time t_0 and end time t_n of the PMSession, we would require continuous values for the power consumption. Since only a discrete amount of samples are available determined by the capabilities of the measurement device, we approximate the energy $\tilde{E}$ using the Riemann sum:

$$\tilde{E} = \frac{1}{2} \cdot \sum_{i=0}^{n-1} \frac{w_i + w_{i+1}}{t_{i+1} - t_i}, \tag{2}$$

where $[w_0, \ldots, w_n]$ represent the measured instantaneous power consumptions and $[t_0, \ldots, t_n]$ their associated timestamps on the server. The accuracy of the integral approximation improves by decreasing measurement interval, as illustrated in Figure 4. The measured code is a CPU intensive computation of Π to a certain precision. Some measurement devices such as the built-in integrator circuit of the Voltech PM1000+ offer native support for measuring energy consumption that can be used by our framework too, in addition to the integration method.

To assess the accuracy of the energy integration $\tilde{E}$, we compared it with the values measured by the hardware integrator of the Voltech PM1000+ power meter. We run the same

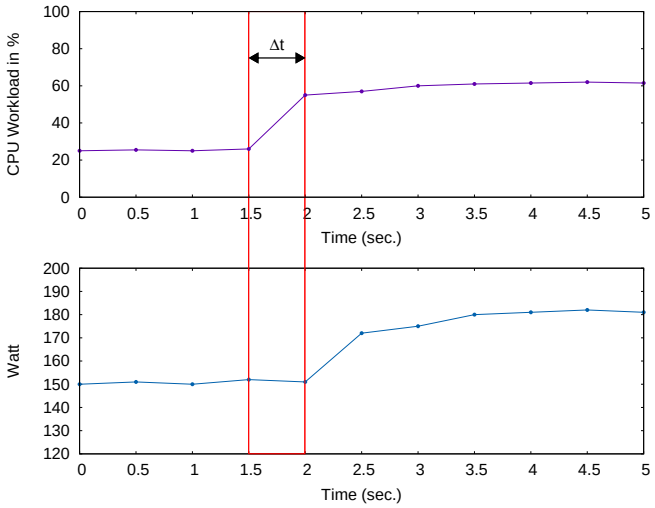

Fig. 6. Delay between workload and power consumption increase reflected at the outlet.

CPU intensive code (i.e. Π computation) with varying target precision resulting in proportionally longer run times, starting from 1 up to 20 seconds. Figure 5 illustrates the average deviation of the $\tilde{E}$ values as a percentage of the hardware-measured values, corresponding to different `PMSession` durations. The trend-line represents an estimation of the average deviation. We observed the highest average divergence of approximately 20% during our shortest test sessions lasting 250 milliseconds. Overall, we observed an average deviation of more than 5% only for sessions shorter than 5 seconds. For longer sessions, the average deviation decreases to approximately 1%.

IV. Improving the Accuracy of Energy Measurements

We present several methods for improving the accuracy of energy measurements. First, we present in Section IV-A our approach to minimise the inaccuracy introduced by the delay between the code execution and its reflected energy draw. Then, we propose in Section IV-B buffering methods for avoiding inaccuracies introduced by measurement truncation. Finally, we present in Section IV-C three methods for aggregating measurements collected with the buffering methods.

A. Workload – power consumption delay

Because of the long, multi-tier electronic circuitry between the mains outlet and the components involved in actual computation (e.g. the processor, the memory), it takes a certain amount of time from the moment when placing a workload on a machine (i.e. executing code) until this state change (i.e. idle-to-load) is reflected in the power consumption (see Figure 6). This delay is caused by multiple high capacitance capacitors placed on the power supply line for current stabilisation and noise filtering purposes. The actual delay Δt between the workload increase and the increase in energy consumption is machine dependent and has to be taken into account to increase the accuracy of our energy consumption measurements.

We determine the typical delay of a machine based on power consumption measurements in three steps: (1) a session is run for measuring the machine's characteristic power consumption in idle state; (2) a succession of intensive codes combined with idle periods are run on the machine and the power consumption is monitored; (3) the characteristic workload power consumption delay of the machine is computed for each execution and the results are aggregated.

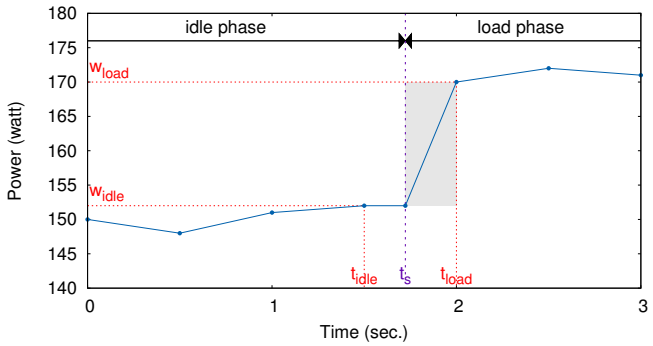

Fig. 7. Time and power values available for delay calculation.

TABLE I. Characteristic workload power consumption delays (d_m).

No.	CPUs and GPUs Type	CPUs and GPUs Count	Power Supplies	d_m [seconds]	w_{idle} [watt]
1.	CPU: AMD Opteron 6168 GPU: AMD Radeon HD 5870	2 1	1	0.485	245.074
2.	CPU: AMD Opteron 6168 GPU: Nvidia GeForce GTX 460	2 1	1	0.412	176.074
3.	CPU: Intel Xeon X5650	2	2	0.240	199.736
4.	CPU: Intel Xeon X5650 GPU: Nvidia GeForce GTX 480	2 2	2	0.349	371.171
5.	CPU: AMD Opteron 885	8	4	0.368	832.816
6.	CPU: Intel Xeon E7-4870	4	2	0.512	474.203
7.	CPU: AMD Opteron 8356	8	4	0.242	494.092
8.	CPU: AMD Opteron 8356	8	4	0.333	508.244
9.	CPU: AMP Opteron 885	4	2	0.372	423.764

For each workload measurement session, the following values are available in the last step (see Figure 7): (1) maximum characteristic power consumption jitter in the machine idle state j_m; (2) session start time t_s; (3) time t_{idle} and power w_{idle} of the last measurement before t_s; (4) time t_{load} and power w_{load} of first measurement after t_s for which the condition $w_{load} - w_{idle} > j_m$ holds. Since for time t_s no power measurement is available (except for the extreme case where $t_s = t_{idle}$), we use w_{idle} as reference as it is the temporally nearest power measurement to t_s. To determine the delay value of a session, we compute first the power increase gradient F:

$$F = \frac{w_{load} - w_{idle}}{t_{load} - t_s}. \quad (3)$$

We compute F for all code executions and extract the *characteristic machine delay*:

$$d_m = t_s - t_{idle} \quad (4)$$

from the maximum gradient. Using this method, we measured the characteristic machine delay of multiple machines as presented in Table I. We observe for the 9 studied machines delays between 240 milliseconds and 512 milliseconds and no correlation of d_m to the number of redundant power supplies or the machines' peak power consumptions.

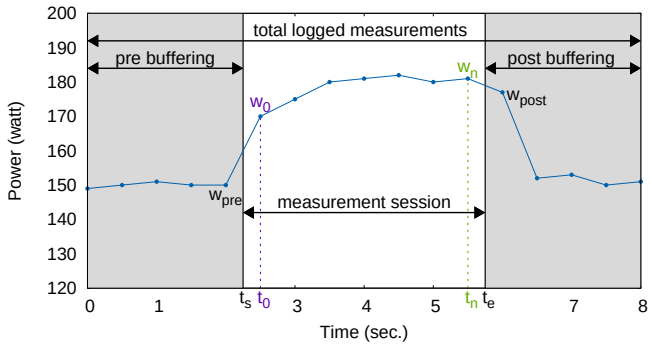

Fig. 8. Logged values for a measurement session with pre- and post-buffering.

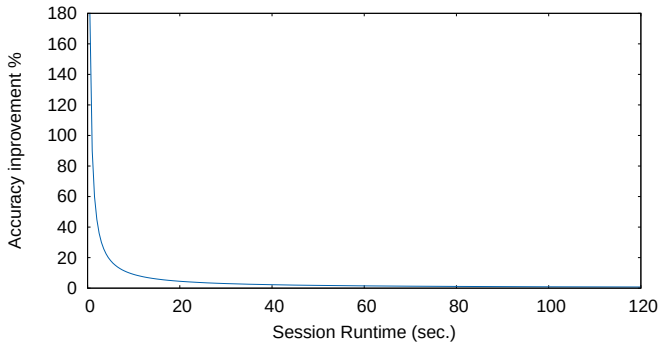

Fig. 9. Accuracy improvements using different session runtimes.

B. Measurement truncation

If a measurement session is shorter than the typical measurement interval of a device, we obtain at most one measurement within this session from which we cannot calculate the overall energy consumption. For this reason, we log on the server side a certain additional amount of measurements before the start and after the end of the session called pre- and post-buffering, as shown in Figure 8. This method also improves the accuracy of short sessions with multiple measurements, as the additional pre- and post-buffering measurements allow us to better interpolate the energy consumption E_t, as follows:

$$E_t = \frac{w_0 + w_{pre}}{2} \cdot (t_0 - t_s) + \frac{w_n + w_{post}}{2} \cdot (t_e - t_n) + E_m, \quad (5)$$

where t_s is the session start time, t_e is its end time, E_m us the session energy consumption calculated from measurements w_0 to w_n captured at times t_0 to t_n, with $i \in [0, n] : t_s \leq t_i \leq t_e$, and w_{pre}, w_{post} is the pre- and post-buffering values captured at times t_{pre} and t_{post}. Similarly to the delay, no measured values are typically available for the session start t_s and end t_e times unless $t_s = t_0$ or $t_e = t_n$. Hence, we use for interpolation the pre- and post-buffering values w_{pre} and w_{post}, as those are the temporally closest measurements available before t_0 and after t_n.

For a better understanding of the Equation 5 and its use, we give a short example measuring the idle energy consumption of an imaginary machine for which the measurement device measures a constant power of 200 Watt every 500 milliseconds. Our measurement session runs for 1.400 milliseconds starting at time $t_s = 50$ and ending at time $t_e = 1450$. We have one pre-buffered measurement w_0 at time $t_0 = 0$, one post-buffered measurement at time $t_0 = 1500$, and two measurements at times $t_0 = 500$ and $t_1 = 1000$. The measured energy consumption E_m within the session is:

$$E_m = P_m \cdot t_m = 200W \cdot (1000ms - 500ms) = 200W \cdot 500ms = 100\,Ws. \quad (6)$$

By applying Equation 5 for calculating the total energy consumption E_t, we obtain:

$$E_t = 200W \cdot (500ms - 50ms) + 200W \cdot (1450ms - 1000ms) + 100\,Ws = 90\,Ws + 90\,Ws + 100\,Ws = 280\,Ws. \quad (7)$$

By comparing the measured and the interpolated energy consumption in this example, we get an absolute difference of 180 Ws, which means that the accuracy improvement using the interpolation mechanism is 180%. However, this mechanism is mainly beneficial for short measurement sessions. Figure 9 shows the accuracy improvements using different session runtimes sharing the same setup as in this example.

Pre- and post-buffering further allows us to meet the workload-energy delay problem described in the previous section, and use it to offer different measurement selection functionality to the user as described in the next section.

C. Measurement aggregation functions

Based on the additional measurements available through pre- and post-buffering, we offer three different (measurement) aggregation functions which determine the range of measurements retrieved from the server.

a) Skewing: aggregation function takes into account the characteristic delay d_m of the machine, as described in Section IV-A. This delay is added to session start t_s and end t_e times, so that the new session start t_{sn} and session end t_{en} correspond to: $t_{sn} = t_s + d_m$ and $t_{en} = t_e + d_m$. All measurements in the $[t_{sn}; t_{en}]$ time interval are sent to the client, including the interpolated values for t_{sn} and t_{en}.

b) Graph: aggregation function detects the increasing and decreasing flanks of a measurement session based on the measured power consumption values. All measurements between the start of the first increasing flank and the end of the last decreasing flank are sent to the client. This aggregation function can be employed only if the energy consumption increases at the beginning of a session, and decreases at the end, therefore it is useful for the precise instrumentation of a single power-intensive code region.

c) Basic: aggregation function is a special case of skewing with the machine-specific delay $d_m = 0$. All measurements between the session start and end are sent to the client without considering the workload energy delay. If necessary, the values for the start and end time are interpolated. This represents the safe approach in case the characteristic delay of the machine is not known (i.e. skewing cannot be applied) and the user instruments multiple consecutive power-intensive code regions (i.e. graph cannot be applied).

Figure 10 illustrates the different value ranges returned when applying the different aggregation functions for measuring a power-intensive test code which computes Π to a

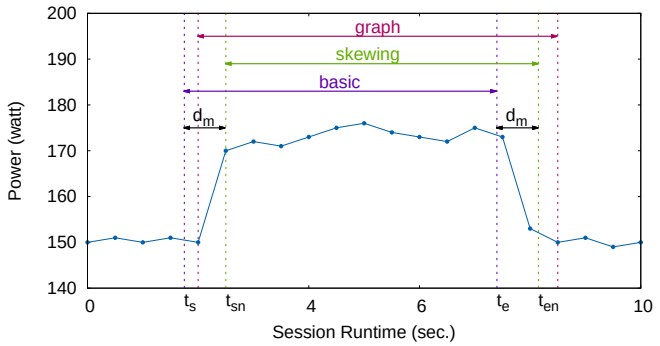

Fig. 10. Retrieved measurement values for the three aggregation functions.

certain precision. The total execution time of the test code is approximately 6 seconds.

V. Related work

We introduce in this section the related work organised in two research fields: energy efficiency and power measurement.

A. Energy efficiency

Several source code level transformations for C/C++ code and their impact on energy consumption during sequential execution are analysed in [4]. The results show that fairly all code transformations which reduce execution time also reduce energy consumption. Power and energy consumption are captured using a model based on the number of clock cycles, cache misses, memory access and further values. The impact of source-to-source code transformations on energy consumption models is studied in [5]. Based on this, a methodology for estimating the potential benefits of applying a code transformation is introduced. This model considers the three major consumers in a machine: processor, system bus, and main memory. The energy estimates are based on three execution parameters: number of assembly instructions, number of instructions and number of data cache misses. These works based on estimation models are complementary to our work, as our measurement and instrumentation framework could be used to test the accuracy of the models on different architectures, especially for code regions .

Qasem et al. present in [6] a strategy based on static compiler analysis and a corresponding auto-tuning framework, allowing to automatically transform a code for parallel execution on multicore architectures and to exploit data locality to improve energy efficiency without sacrificing performance. They estimate the codes' energy efficiency using the HPC-Toolkit [7] and perform the validation using WattsPro power meters. In contrast, our approach is not based on modelling requiring complex validation, but offers direct online energy consumption measurements.

B. Energy measurement

SoftWatt [8] is a power consumption simulator that estimates the components' power characteristics during code execution. In contrast, our solution directly measures the power and energy characteristics on system, subsystem or component level, depending on the available measurement devices.

PowerPack [9] is a framework for measuring energy consumption of devices including disks, network interface cards, processors, memories, and total systems, and correlate it to application functions. PowerPack uses timestamps to synchronise multiple data streams from different measurement devices and create post-mortem power consumption profiles. While PowerPack is especially tailored towards high performance clusters, our solution aims at general applicability. We use code instrumentation with single machine timestamps (the measurement server's), thus eliminating synchronisation processes which may introduce inaccuracies, especially for short time intervals. Moreover, our approach is not restricted to post-mortem analysis but can supply the measurement data at runtime, this being a viable option for self-tuning systems.

VI. Conclusions

We proposed in this paper an energy instrumentation and online measurement framework targeting the deployment of self-tuning codes in data centres. Our framework allows for both offline and online analyses by providing detailed energy consumption data for software programs with a region-level granularity. Our modular architecture shields the users from needing in-depth knowledge of their energy measurement hardware, and allows the development of measurement and instrumentation code independent of the instrument's proprietary interface. We also proposed an efficient method for increasing the accuracy of measurements for low sampling rate measurement devices (e.g. 0.5 to 10 Hertz sampling frequency). Using Voltech PM1000+ measurement devices. we showed an increase in accuracy of approximately 20% for measurements lasting 5 seconds and of approximately 9% for measurements lasting 10 seconds.

References

[1] J. Koomey, "Growth in data center electricity use 2005 to 2010." Analytics Press, Tech. Rep., August 2011. [Online]. Available: http://www.analyticspress.com/datacenters.html

[2] G. M. ads blog, "Mobile internet & smartphone adoption," January 2012, online available at: http://googlemobileads.blogspot.com/2012/01/new-research-global-surge-in-smartphone.htm, visited on 25th February 2013.

[3] S. Murugesan, "Harnessing green it: Principles and practices," *IT Professional*, vol. 10, no. 1, pp. 24–33, 2008.

[4] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto, "The impact of source code transformations on software power and energy consumption," *Journal of Circuits, Systems, and Computers*, vol. 11, no. 5, pp. 477–502, 2002.

[5] ——, "Analysis and modeling of energy reducing source code transformations," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 3, 2004, pp. 306–311 Vol.3.

[6] A. Qasem, M. Cade, and D. Tamir, "Improved energy efficiency for multithreaded kernels through model-based autotuning," in *Green Technologies Conference, 2012 IEEE*, 2012, pp. 1–6.

[7] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "Hpctoolkit: tools for performance analysis of optimized parallel programs http://hpctoolkit.org," *Concurr. Comput. : Pract. Exper.*, vol. 22, no. 6, pp. 685–701, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1002/cpe.v22:6

[8] S. Gurumurthi, A. Sivasubramaniam, M. Irwin, N. Vijaykrishnan, and M. Kandemir, "Using complete machine simulation for software power estimation: the softwatt approach," in *Eighth International Symposium on High-Performance Computer Architecture*, 2002, pp. 141–150.

[9] R. Ge, X. Feng, X. Song, H. Chang, D. Li, and K. Cameron, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 5, pp. 658 –671, May 2010.