Measurement for Energy Consumption of Applications

Abstract—**For IT systems, energy awareness can be improved in two ways, (i) in a static or (ii) in a dynamic way. The first way leads to building energy efficient hardware that runs fast and consumes only a few watts. The second way consists of reacting to instantaneous power consumption, and of taking decisions that will reduce this consumption.**

In order to take decisions, it is necessary to have a precise view of the power consumption of each element of an IT system. Nowadays, it is only possible to measure power at the outlet level, or - with larger efforts - a the hardware component level. It is still difficult to estimate the power consumption of single applications running on a computer.

This work aims at evaluating the power consumption of single applications using indirect measurements. We describe a methodology for predicting the power consumption of a PC depending on performance counters, and then use these counters to predict the power consumption of a single process.

Keywords-**energy efficiency; process power consumption; performance counters; linear regression;**

I. INTRODUCTION

Energy awareness can be improved in two ways for IT systems, in a static or in a dynamic way. The static approach is to design energy efficient hardware in the first place. The dynamic approach tries to utilize hardware in an optimal way by moving workload or setting hardware parameters which depend on the workload.

Those two approaches are complementary, and we mainly address the second one, as is done in autonomous systems. Such systems can react to several events, one of them being power consumption of hosts. Current hardware technology allows only to measure the power consumption of a whole node. As we manipulate Virtual Machines, we are interested in the power consumption of each application or VM inside the nodes. To address this problem it is necessary to create measurement software able to extract the power consumption of each single application.

Several techniques are currently used, but they either focus on a particular subsystem (such as processor [1], memory [2], GPU [3]) or are not precise enough (in [4] a comparison of several model leads to an average error of 10%). Generally some process related values are monitored, and based on those values we can then create a mathematical model linking those data to the power consumption of an application.

The contribution of this paper is twofold: first we investigate the questions whether the power consumption of a PC can be predicted by measuring OS performance variables. Second, we show how this information can be used to derive the power consumption of a single process.

II. METHODOLOGY

Our methodology focuses on standard off-the-shelf PC hardware and software running under the Linux OS. The global methodology is the following:

- We run several applications and synthetic benchmarks
- We log several measurements during the run (including power consumption)
- We use the logged values to derive a mathematical model linking measurements and power consumption

To achieve application power measurement, we use indirect measurements such as performance counters, process related information (using Linux `pidstat`) and host related information using (using `collectd`). The two first information types are related to a single process, and are to be the base of the produced model. Information based on `collectd` are machine-wide and thus not related to a particular process. But some measurements (such as network traffic) are currently difficult to obtain in a process-related way.

Variables measured using per process performance counters start with *perf_* and include for instance: *perf_task.clock.msecs* (CPU time per second), *perf_context.switches* (context switches/s), *perf_CPU.migrations* (process migrations/s), *perf_page.faults* (page faults/s), *perf_cycles* (CPU cycles/s), *perf_instructions* (instructions/s), *perf_cache.references* (references to the cache/s), and *perf_cache.misses* (cache misses/s).

Variables measured with `pidstat` start with *pid_* and include: *pid_usr* (Percentage of CPU used by the task while executing at the user level), *pid_system* (Percentage of CPU used by the task while executing at the system (kernel) level), *pid_guest* (Percentage of CPU spent by the task in virtual machine), *pid_CPU* (index of the CPU on which is

the process), *pid_minflt_s* (Total number of minor faults the task has made/s), *pid_majflt_s* (Total number of major faults the task has made/s), *pid_kB_rd_s* (Number of kilobytes the task has caused to be read from disk/s), *pid_kB_wr_s* (Number of kilobytes written/s), *pid_kB_ccwr_s* (Number of kilobytes whose writing to disk has been cancelled by the task), *pid_VSZ* (Virtual Size: The virtual memory usage of entire task in kilobytes), *pid_RSS* (Resident Set Size: The non-swapped physical memory used by the task in kilobytes), *pid_MEM* (Percentage of memory used (by resident memory)).

The system wide metrics from `collectd` start with *host_* and include: *host_df_df.root_used* (Instantaneous used octets on /root), *host_df_df.root_free* (Instantaneous free octets on /root), *host_memory_memory.cached_value* (Instantaneous cached memory), *host_interface_if_octets.eth1_rx* (Number of octets received during the last second), *host_interface_if_octets.eth1_tx* (Number of octets sent during the last second), *host_interface_if_packets.eth1_rx* (Number of packets received during the last second), *host_interface_if_packets.eth1_tx* (Number of packets sent during the last second), *host_processes_ps_state.blocked_value* (instantaneous number of processes in blocked state), *host_disk.sda_disk_time_write* (average time a write.operation took to complete), *host_cpu.X_cpu.system_value* (processes executing in kernel mode), *host_cpu.X_cpu.wait_value* (waiting for I/O to complete), *host_load_load_shortterm* (Instantaneous shortterm load, average over 1min), *host_load_load_midterm* (Instantaneous midterm load, average over 5min), *host_irq_irq.X_value* (Number of irqs during the last second), *host_disk.sda_disk_merged_read* (Number of reads that could be merged into other, already queued operations), *host_disk.sda_disk_merged_write* (Idem for write).

A. Monitoring

The currently implemented framework uses two elements, a monitoring part and a synthetic workload part. The monitoring part is split between two computers in order to reduce the impact of measurements on the experiment. A first computer runs the application to be measured and carries out the process and machine related measurements. A second computer is linked to a watt meter and logs the power consumption. At start the two computer clocks are synchronized.

The current limit of the framework is that it cannot follow at the same time a process and its children. So it can currently only be used for applications that do not fork. As most benchmarks actually fork, we had to develop new ones to create the large datasets from which to create the models.

B. Synthetic workload

We created several benchmarks linked to synthetic workloads: Memory, CPU, network, disk and mixed. General behavior of each of them is always the same: They start at 100% of a particular resource and go down to 0% by step of 1%. Each step is 20s. Each second we measure around 200 values (using `pidstat`, `perfcounters`, `collectd` and a wattmeter). Thus one experiment produces around 2 MB of data. Starting at 100% and going down to 0% allows to reduce the impact of allocating resources as advised by SpecPower.

III. Data Analysis

The measured data consists of one response variable (power), and 165 explanatory variables. The task of model building is to explain the response as a function of the explanatory variables. In order to cope with higher order dependencies we also include the squared explanatory variables (marked by a "SQ" at the variable name end), yielding a total of 330 explanatory variables. We did not include interactions between the variables, since this would have lead to an explosion of the number of variables being uncomputable.

The task was now to identify a small number of variables that would explain each individual data set, and additionally a model that would explain all datasets in parallel. The desired models should be parsimonious (number of explanatory variables should be low) and, in relation to this number, the model quality should be as good as possible. Model quality is mainly measured by R^2, the squared correlation coefficient, which explains how much better our model is with respect to the simple data average. In more detail, R^2 measures how much of the total variance (when using the average of the data) is explained when exchanging the overall average with the respective model. Since $0 \leq R^2 \leq 1$, and a larger value is better, as an initial goal we wanted to at least achieve $R^2 \geq 0.9$, and consider a model to be "good" if $R^2 \geq 0.95$.

A. Model Quality

Model quality is also measured as average error σ, i.e., the square root of the sum of the squared differences between predicted and measured values. This σ, being the average prediction error, is also set in relation to the average power, yielding the average error in %. Here it must be noted that similar approaches usually report an average error of 5% or higher. Our task thus was to derived models that clearly result in much smaller average errors. However, error in % is of course very much influenced by the mean power consumption of the computers, i.e., higher average power consumption results in a lower error % without yielding a better model, and thus R^2 has more meaning that error in %.

Data was preprocessed by removing data items with at least one N/A value in any variable. The number of removed data lines however is quite small, in the order of below one per mill.

From data analysis we saw that there were only very few extremely influential points which can be identified using Cook's distance [5]. When removing them, the model quality quickly increases. We therefore also designed a simple outlier test removing those points which are at least 4σ away from the mean. This limit must be regarded as being extremely conservative, since usually either 2σ or 3σ are used. The result is a small number of outliers being removed, and we also state R^2 and σ for these cases, as well as the number of removed outliers.

Another test was done to see whether the residuals are normally distributed. However, since the dataset sizes are very high, and there are certain regularities left, using the Shapiro-Wilk test we cannot conclude that the residuals are indeed normally distributed. However, we plotted the data also on quantile-quantile-plots (Q-Q-Plots). On such a plot we put the quantiles of the theoretical normal distribution on the x-axis, and the empirically measured quantiles of the data on the y-axis. If the resulting plot is nearly linear (a line) then we can conclude that the data follows a normal distribution. The Q-Q-plots of our residuals indeed mainly follow a straight line, but show deviances at the tails, which explains why the Shapiro-Wilk test failed.

B. Additive Model Update

In our analysis we start with an empty model in which we only use the average to describe the response. We then use exhaustive search to find the best combination of N variables explaining the response. Since, when having K explanatory variables, the effort for doing this is

$$O(K(K-1)\cdots(K-N+1)),$$

this is not possible for larger values of both K and N due to an exponential runtime explosion.

To further reduce complexity we actually chose a mixture approach between additive and subtractive, and first start with a full model containing all variables. Then we remove all those variables that do not have any influence onto the result, by demanding that the p-value (of a t-test testing whether the coefficient is different from zero) of a variable should be larger than 0.5. This reduced K, thus speeding up the following additive approach significantly. This reduction of course is not possible if a full model cannot be created due to a lack of data. In these cases, we had to run the search for $K = 330$, which resulted in a drastically increased runtime. Still, combinations of more than $N = 4$ hardly make sense since they demand run times of days or months. However, there is seldom demand for doing so, since quite often only little can be gained by using more than three variables.

A further possibility is to sequentially add the next best L variables to the model. This means that we can start with an empty model, and then continuously find the best combination of L variables to add to the model. This approach still increases exponentially with L, but only linearly with N. We therefore also state the resulting models for $L = 1$.

IV. Analysis Results

The data consists of the six datasets "Burn CPU", "Mem Loop", "Network", "Tar Kernel", "Disk Read" and "Disk Write". The sizes of these sets are quite large with the exception of "Tar Kernel". The datasets do differ significantly with respect to their power consumption. The empirical cumulative distribution functions (ECDFs) are shown in Figure 1. An ECDF is the integral of the empirically measured density of a dataset. Data is concentrated at those values where the ECDF rises steeply. The datasets influenced by CPU and memory show a rather broad spectrum of power values. Those making heavily use of I/O like "Network" and "Disk X" show a rather narrow spectrum. "Tar Kernel", making use of both CPU and I/O is also narrow, but with a significant shift to the right.

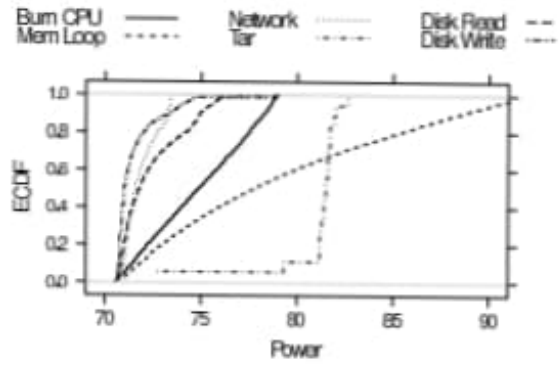

Figure 1. Empirical CDFs (ECDFs) of the respective power consumptions.

In the following we show results of the regression analysis, which also includes plots of the respective residuals. In these plots the x-axis is as important as the y-axis, since it shows the range of the power measurements. Residual sizes must also be related to these ranges.

A. Burn CPU

The dataset "Burn CPU" consists of 2074 complete data records (without N/A values). Both datasets "Burn CPU" and "Mem Loop" are similar in the sense that power can be explained by using one variable only. Table I shows the best variables explaining the power, with outliers taken into account, and when removing outliers. It must be noted that it is not clear why e.g. host_df_df.root_free_SQ yields such good results. There are some other host related variables also yielding high correlations. However, for instance the variables perf_instructions and perf_cycles do make sense

in a pure CPU stress test and also well explain the power consumption.

Variable	R^2	σ	%	#Out
host_df_df.root_free_SQ	0.992	0.216	0.289	0
	0.996	0.146	0.195	12
host_df_df.root_used	0.992	0.216	0.289	0
	0.996	0.146	0.195	12
perf_instructions	0.991	0.232	0.310	0
	0.996	0.143	0.191	12
perf_cycles	0.991	0.233	0.310	0
	0.997	0.144	0.192	12

Table I
BURN CPU.

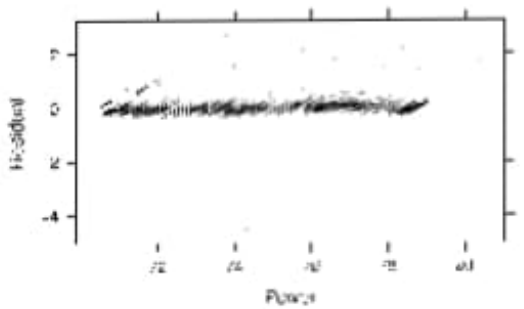

Figure 2. Burn CPU residuals for variable host_df_df.root_free_SQ.

Figure 2 shows the model residuals when using a model with only the variable host_df_df.root_free_SQ as single explanatory variable. There are only a few outliers, nevertheless influencing σ significantly.

B. Memory Loop

The dataset "Mem Loop" consists of 2017 complete data records (without N/A values). Table II shows the best variables explaining the power.

Variable	R^2	σ	%	#Out
perf_context.switches	0.998	0.268	0.339	0
	0.999	0.213	0.270	18
perf_cache.references	0.998	0.285	0.361	0
	0.999	0.196	0.248	25
perf_context.switches_SQ	0.977	0.919	1.163	0

Table II
MEMORY LOOP.

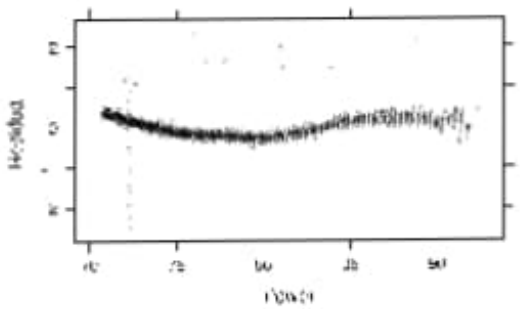

Figure 3. Loop memory residuals for variable perf_context.switches.

Figure 3 shows the model residuals when using a model with only the variable perf_context.switches as single explanatory variable. Again a few outliers are visible, which however do not significantly influence the results. Also, it is obvious that the residui do exhibit a deterministic component. In fact this deterministic dependence could be exploited by further refining the regression model, e.g., by adding a $\sin()$-like function. It turns out that this is not really necessary since the one-variable model is already good enough such that any model extension is superfluous.

C. Network

The dataset "Network" consists of 1937 complete data records (without N/A values). Table II shows the best variables explaining the power.

Variable	R^2	σ	%	#Out
host_interface_if_octets.eth1_tx	0.954	0.178	0.247	0
	0.983	0.106	0.147	13
host_interface_if_packets.eth1_rx	0.953	0.181	0.251	0
	0.980	0.117	0.163	13
host_interface_if_packets.eth1_tx	0.951	0.183	0.255	0
	0.981	0.114	0.159	13
perf_cycles +	0.967	0.152	0.211	0
host_interface_if_packets.eth1_tx	0.991	0.079	0.110	16
perf_instructions +	0.967	0.152	0.211	0
host_interface_if_packets.eth1_tx	0.991	0.079	0.110	16
perf_cycles +	0.967	0.152	0.211	0
host_interface_if_octets.eth1_tx	0.991	0.077	0.108	17

Table III
NETWORK.

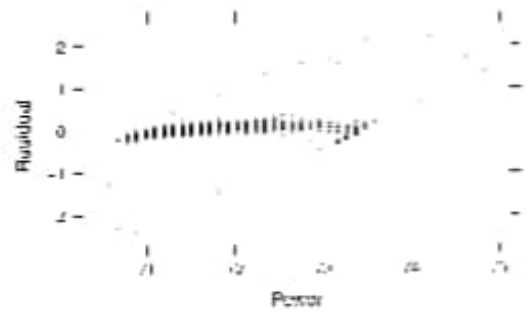

Figure 4. Network residuals for variable host_interface_if_octets.eth1_tx.

Figure 4 shows the model residuals when using a model with only the variable host_interface_if_octets.eth1_tx as single explanatory variable. This variable makes sense when keeping in mind that the workload mainly stresses the network card. Furthermore, Table III and Figure 4 show that although two variables are enough to explain the power consumption, there are a few outliers that do have a significant influence in the result. Removing only 13 out of 1937 data records increases R^2 to 0.98 for one, and 0.99 for two variables.

D. Tar Kernel

This dataset contains only 18 complete data records. Inasmuch it was not possible to reduce the number of

explanatory variables by using a full model first, and thus, searching for optimal variable combinations had to include all 330 explanatory variables (K=330). This was partially compensated by the fact that regression for a small dataset is much faster. Nevertheless, the resulting run times were much higher. Table IV and Figure 5 show the regression results. It can be seen that mainly host oriented variables perform best, e.g., number of blocked processes, disk writing load, but also the load averages.

Variable	R^2	σ	%
host_processes_ps_state. blocked_value	0.911	0.661	0.816
host_processes_ps_state. blocked_value_SQ	0.911	0.660	0.816
host_disk.sda_disk_time_write_SQ	0.911	0.661	0.817
host_cpu.2_cpu.wait_value_SQ + host_load_load_midterm_SQ	0.984	0.290	0.358
pid_kB_rd_s_SQ + host_load_load_midterm_SQ	0.983	0.295	0.364
host_cpu.2_cpu.wait_value_SQ + host_load_load_shortterm_SQ	0.983	0.300	0.370

Table IV
TAR KERNEL.

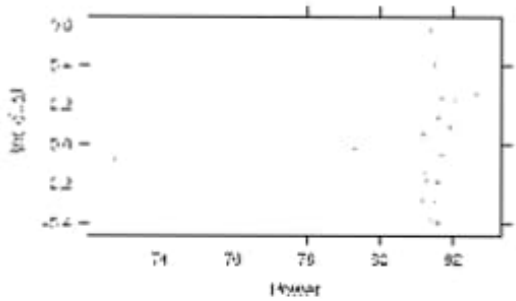

Figure 5. Tar kernel residuals for variables host_cpu.2_cpu.wait_value_SQ + host_load_load_midterm_SQ.

E. Disk Read

This dataset contains 2018 complete data records. As can be seen in Tables V and VI, and Figure 6 there are a few outliers that have a significant impact on the model accuracy. Removing them results in a sufficient fit though. We also computed models for the optimal triple of variables, which however did not improve the model quality significantly. We thus omitted them in Table V. Interestingly single variables do not include disk related variables, but when sequentially adding variables, pid_kB_rd_s_SQ turns up at place third, so it is definitely related to the power consumption.

Table VI shows that for six variables R^2 reaches up to 0.98 without outliers, more variables though do not improve the accuracy any more.

F. Disk Write

This dataset contains again 2018 non-empty data records. Similar to "Disk Read", this dataset also suffers from a

Variable	R^2	σ	%	#Out
perf_cache.misses	0.824	0.698	0.964	0
	0.956	0.337	0.465	18
perf_context. switches_SQ	0.791	0.761	1.05	0
	0.946	0.361	0.499	22
perf_cache. references_SQ	0.768	0.803	1.109	0
	0.963	0.298	0.412	23
perf_cache.references + pid_system_SQ	0.941	0.405	0.559	0
	0.972	0.277	0.382	12
perf_cycles + pid_system_SQ	0.939	0.410	0.566	0
	0.971	0.281	0.388	12
perf_cache. references_SQ + pid_system_SQ	0.939	0.412	0.569	0
	0.971	0.283	0.391	12

Table V
DISK READ.

Variable	R^2	σ	%	#Out
perf_cache.misses	0.824	0.698	0.964	0
	0.956	0.337	0.465	18
perf_cache.misses_SQ	0.909	0.502	0.694	0
	0.956	0.337	0.465	17
pid_kB_rd_s_SQ	0.919	0.474	0.655	0
	0.970	0.280	0.387	18
pid_system_SQ	0.937	0.417	0.576	0
	0.971	0.274	0.379	19
perf_cache.references_SQ	0.949	0.377	0.520	0
	0.979	0.236	0.326	14
perf_instructions_SQ	0.951	0.367	0.508	0
	0.980	0.231	0.320	14

Table VI
DISK READ, SEQUENTIALLY ADDING SINGLE VARIABLES.

handful of severe outliers. Figure 7 and Tables VII and VIII show the details. Here it can be seen that power for this workload can be best described without using disk related variables. In general the models perform a little worse for disk related workload (read and write), but when adding more variables the model quality quickly rises significantly.

G. All Datasets

The previous sections show that our approach yields good results for individual types of work load. In this subsection we derive models for a dataset containing all previously analyzed datasets. Since the datasets are quite heterogeneous it can be assumed that only 1,2 or 3 variables will not yield good models. However, Table IX shows that three variables already yield a good model. The main result however is shown in Tables X and XI, showing nine variables that have been added sequentially. As can be seen, even without outlier removal, these variables explain 99% of the total data variance, yielding them to be excellent predictors of the "All" dataset.

The sequential model shown in Table X also shows the coefficient of variable perf_cache.misses_SQ, which is in the order of magnitude of 10^{-14}. This is explained by the fact that the maximum of this variable (squared number of cache misses per second) is also in the order of magnitude of 10^{14}.

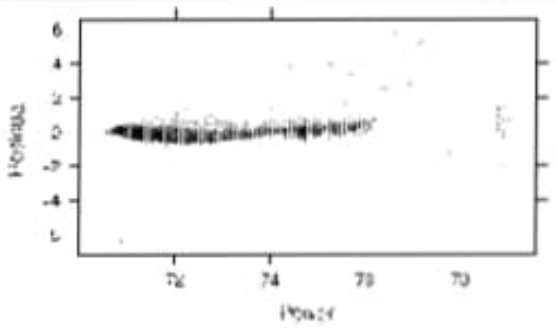

Figure 6. Disk read residuals for variables perf_cache.references + pid_system_SQ.

Variable	R^2	σ	%	#Out
perf_cache.misses	0.768	0.588	0.822	0
	0.925	0.297	0.415	19
perf_context.switches_SQ	0.635	0.738	1.032	0
	0.955	0.211	0.295	22
perf_context.switches	0.628	0.745	1.0414	0
	0.919	0.281	0.393	23
perf_cycles_SQ +	0.895	0.396	0.553	0
host_cpu.0_cpu. system_value_SQ	0.967	0.216	0.303	12
perf_instructions_SQ +	0.895	0.397	0.555	0
host_cpu.0_cpu. system_value_SQ	0.967	0.217	0.303	12
perf_context. switches_SQ +	0.893	0.400	0.560	0
host_cpu.0_cpu. system_value_SQ	0.970	0.202	0.283	14
perf_context.switches +	0.901	0.385	0.539	0
perf_instructions_SQ +	0.969	0.211	0.295	10
host_cpu. 0_cpu.system_value_SQ				
perf_context.switches +	0.901	0.385	0.539	0
perf_cycles_SQ +	0.969	0.211	0.296	10
host_cpu. 0_cpu.system_value_SQ				
perf_cache.misses +	0.900	0.386	0.539	0
perf_instructions_SQ +	0.970	0.203	0.283	16
host_cpu. 0_cpu.system_value_SQ				

Table VII
DISK WRITE.

Figure 8 shows the model residuals when using nine variables. Residuals are taken from the data sets in the order "Burn CPU", "Mem Loop", "Network", "Tar Kernel", "Disk Read" and "Disk Write". As can be seen residuals are not purely random, but also depend on the data set, but not to a high degree. This is further investigated in Section IV-H.

Figure 9 show how the models improve when adding more and more variables sequentially, i.e., the ECDFs of the residuals when using 1, 3, 5, 7, or 9 variables added sequentially. It can clearly be seen that the 9-variables model is superior and yields excellent results.

Finally, Figure 10 shows the quantiles of the residuals plotted against the theoretical quantiles of normal distributions. The more straight lines the curves are, the more

Variable	R^2	σ	%	#Out
perf_cache.misses	0.761	0.588	0.822	0
	0.925	0.297	0.415	19
perf_cache.misses_SQ	0.816	0.524	0.732	0
	0.919	0.308	0.430	20
host_cpu.	0.865	0.449	0.627	0
0_cpu.system_value_SQ	0.932	0.296	0.414	21
perf_context.switches_SQ	0.906	0.374	0.523	0
	0.973	0.196	0.274	13

Table VIII
DISK WRITE, SEQUENTIALLY ADDING SINGLE VARIABLES.

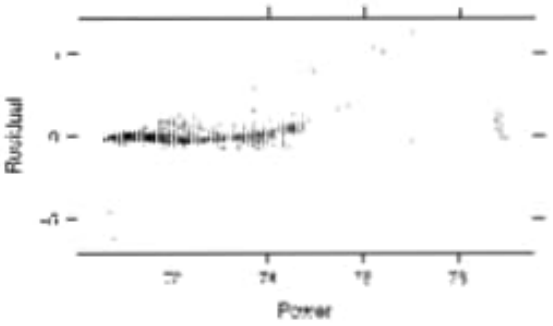

Figure 7. Disk write residuals for variables perf_context.switches + perf_instructions_SQ + host_cpu.0_cpu.system_value_SQ.

they follow a normal distribution. As can be seen, the model using the optimal triple (O3), as well as the model using 9 variables show a large region between -3.5 and 2.5 where they follow a normal distribution nicely. Outside this region, both models show differences in their tails caused by outliers.

H. Global Model Applied to all Datasets

Table XI shows how a model with 9 variable explains all data sets in total. The question now is how this global

Variable	R^2	σ	%	#Out
perf_cache.references_SQ	0.770	2.003	2.708	0
perf_cache.misses_SQ	0.767	2.017	2.728	0
	0.773	1.987	2.686	16
perf_cache.references	0.683	2.353	3.181	0
pid_usr +	0.920	1.181	1.597	0
perf_cache.references_SQ	0.937	1.046	1.414	57
perf_instructions_SQ +	0.917	1.207	1.632	0
perf_cache.references_SQ	0.933	1.080	1.460	57
perf_task.clock.msecs_SQ +	0.916	1.208	1.633	0
perf_cache.references_SQ	0.933	1.078	1.457	57
perf_cache.references +	0.963	0.807	1.091	0
perf_instructions_SQ +	0.976	0.646	0.874	62
pid_RSS_SQ				
perf_cache.references +	0.963	0.807	1.091	0
perf_instructions_SQ +	0.976	0.646	0.874	62
pid_MEM_SQ				
perf_cache.references +	0.963	0.807	1.091	0
perf_instructions_SQ +	0.976	0.646	0.874	62
pid_VSZ_SQ				

Table IX
ALL DATASETS.

Variable	Coeff	Coeff Err
Intercept	7.056e+01	7.781e-03
perf_cache.misses_SQ	-8.195e-15	1.671e-15
pid_usr	8.401e-02	1.879e-04
host_irq_irq.26_value	6.882e-03	6.278e-05
pid_system_SQ	3.717e-03	3.237e-05
perf_context.switches	6.266e-04	1.107e-05
pid_kB_wr_s	1.737e-04	4.696e-06
host_disk.sda_disk_merged_write	-5.270e-04	1.930e-05
perf_cache.references	1.190e-06	1.357e-08
pid_VSZ	-1.601e-05	2.064e-07

Table X
ALL DATASETS, REGRESSION COEFFICIENTS AND ERRORS OF THE COEFFICIENTS. ALL COEFFICIENTS ARE SIGNIFICANTLY DIFFERENT FROM ZERO (P-VALUES<1.0E-7)

Variable	R^2	σ	%	#Out
perf_cache.misses_SQ	0.767	2.017	2.728	0
	0.773	1.987	2.686	16
pid_usr	0.916	1.209	1.635	0
	0.935	1.062	1.437	56
host_irq_irq.26_value	0.950	0.934	1.263	0
	0.971	0.706	0.955	63
pid_system_SQ	0.966	0.772	1.043	0
	0.977	0.634	0.857	48
perf_context.switches	0.976	0.651	0.881	0
	0.986	0.491	0.663	65
pid_kB_wr_s	0.980	0.593	0.801	0
	0.989	0.426	0.577	78
host_disk.	0.983	0.541	0.732	0
sda_disk_merged_write	0.989	0.430	0.581	92
perf_cache.references	0.985	0.508	0.687	0
	0.990	0.407	0.550	77
pid_VSZ	0.991	0.402	0.544	0
	0.995	0.282	0.382	90

Table XI
ALL DATASETS, SEQUENTIALLY ADDING SINGLE VARIABLES.

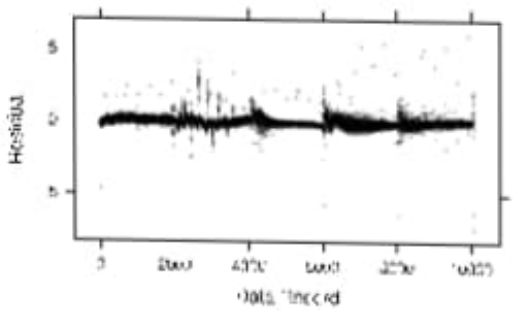

Figure 8. All datasets residuals for the sequential model using 9 variables shown in Table X.

model performs when applied to each individual data set. Table XII shows the respective regression results. As can be seen the global model performs outstandingly well, and explains the power consumption for every single dataset. The only data set having some deviances is "Disk Write", but when removing only 12 outliers, the situation dramatically changes, and 97% of the variance is explained, which is an excellent result.

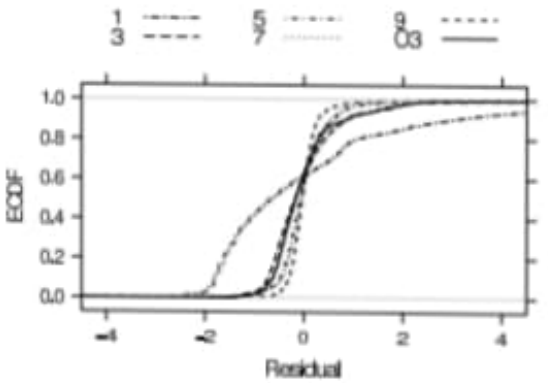

Figure 9. All datasets, ECDFs of the residuals, when using 1, 3, 5, 7, or 9 variables added sequentially. O3 denotes the optimal combination of 3 variables.

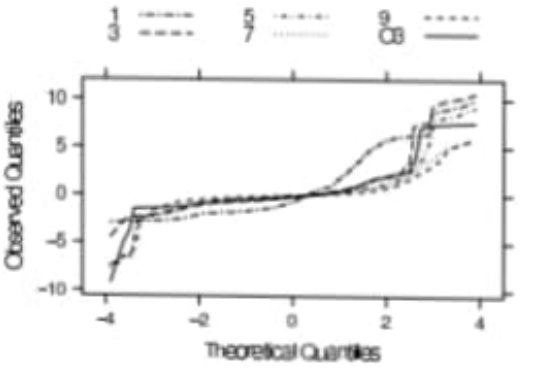

Figure 10. All datasets, Q-Q-plot of the residuals, when using 1, 3, 5, 7, or 9 variables added sequentially. O3 denotes the optimal combination of 3 variables.

V. Power Consumption of a Single Process

The above derived models describe the power consumption of a computer by a combination of system wide variables $Y_i, i = 1, \ldots, I$, as well as variables $X_{jl}, j = 1, \ldots, J$ describing individual process $p_l, l = 1, \ldots, L$. Let the power consumption of a computer with no load be denoted by P_0 (the intercept), and the respective coefficients of the regression model be called α_i for system wide variables, and β_j for per process variables. The linear model then relates

Data set	R^2	σ	%	#Out
"Burn CPU"	0.992	0.225	0.3	0
"Mem Loop"	0.999	0.144	0.24	0
"Network"	0.944	0.196	0.27	0
	0.974	0.134	0.19	17
"Tar Kernel"	0.965	0.556	0.69	0
"Disk Read"	0.946	0.389	0.54	0
	0.974	0.265	0.36	11
"Disk Write"	0.89	0.407	0.57	0
	0.965	0.221	0.31	12

Table XII
APPLYING THE GLOBAL MODEL FROM TABLE XI TO THE RESPECTIVE DATA SETS.

the power consumption P of a computer in the following way:

$$P = P_0 + \sum_{i=1}^{I} \alpha_i Y_i + \sum_{j=1}^{J} \beta_j \sum_{l=1}^{L} X_{jl}. \quad (1)$$

By setting

$$S_j = \sum_{l=1}^{L} X_{jl},$$

(1) can be written as

$$P = P_0 + \sum_{i=1}^{I} \alpha_i Y_i + \sum_{j=1}^{J} \beta_j S_j. \quad (2)$$

In order to derive the power caused by an individual process P_l, we assume that the relevant system wide variables are caused by the processes, which are themselves described by their individual per process variables. To verify this assumption we explained the system wide variables host_irq_irq.26_value and host_disk.sda_disk_merged_write by the per process variables. The resulting R^2 were 0.956 and 0.99, i.e., they can be almost entirely explained by per process variables. We therefore write

$$Y_i = \sum_{j=1}^{J} \gamma_{ij} S_j, \quad (3)$$

and adapt (1) to a new form

$$\begin{aligned} P &= P_0 + \sum_{i=1}^{I} \alpha_i \sum_{j=1}^{J} \gamma_{ij} S_j + \sum_{j=1}^{J} \beta_j S_j \\ &= P_0 + \sum_{j=1}^{J} S_j \left(\beta_j + \sum_{i=1}^{I} \alpha_i \gamma_{ij} \right) \\ &= P_0 + \sum_{j=1}^{J} \eta_j S_j \end{aligned} \quad (4)$$

by defining

$$\eta_j = \beta_j + \sum_{i=1}^{I} \alpha_i \gamma_{ij}.$$

Thus, we now describe the global power consumption by per process variables only. At this point we can derive the power P_l as caused by p_l to be

$$P_l = \sum_{j=1}^{J} \eta_j X_{jl}. \quad (5)$$

It can easily be seen that

$$P = P_0 + \sum_{l=1}^{L} P_l,$$

as it should be.

VI. Conclusion

We defined and implemented a measurement platform at the University of Vienna which we used to measure the dependence of the power consumption of a standard PC onto synthetic workload. By using multivariate regression we explain this power consumption by a subset of 165 process and system variables we measured during our experiments. We explore this dependence in depth and demonstrate that the obtained model quality is very good. We derive a global model for all data sets that also explains each single data set excellently. Using simple linear analysis we also define the power consumption of a single process.

Although we present a multitude of results, the main result of this paper is actually the methodology itself. The described regression results merely demonstrate that our approach does yield excellent prediction results.

Future works include choosing the best values to measure in order to reduce the impact of measurement on the system.

Acknowledgment

This work was done during a short term scientific mission (Cost action 0804) of Georges Da Costa (IRIT, Toulouse, France) to the University of Vienna (Austria) from 8, January to 12, February 2010, and was supported by the European Commission.

References

[1] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design*. ACM, 2001.

[2] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. J. Irwin, and A. Sivasubramaniam, "vec: virtual energy counters," in *PASTE '01: Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. ACM, 2001.

[3] X. Ma, M. Dong, L. Zhong, and Z. Deng, "Statistical power consumption analysis and modeling for gpu-based computing," in *SOSP Workshop on Power Aware Computing and Systems (HotPower '09)*, 2009.

[4] S. Rivoire, P. Ranganathan, and C. Kozyrakis, "A comparison of high-level full-system power models." in *HotPower*, F. Zhao, Ed. USENIX Association, 2008.

[5] M. J. Crawley, *Statistics: An Introduction using R*. Wiley, 2005, iSBN 0-470-02297-3. [Online]. Available: http://www.bio.ic.ac.uk/research/crawley/statistics/