Engineering Energy-Aware Web Services toward Dynamically-Green Computing

Peter Bartalos and M. Brian Blake

Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, Indiana, USA
{peter.bartalos.1,m.brian.blake}@nd.edu

Abstract. With the emergence of commodity computing environments (i.e. clouds), information technology (IT) infrastructure providers are creating data centers in distributed geographical regions. Since geographic regions have different costs and demands on their local power grids, cloud computing infrastructures will require innovative management procedures to ensure energy-efficiency that spans multiple regions. Macro-level measurement of energy consumption that focuses on the individual servers does not have the dynamism to respond to situations where domain-specific software services are migrated to different data centers in varying regions. Next-generation models will have to understand the impact on power consumption for a particular software application or software service, at a micro-level. A challenge to this approach is to develop a prediction of energy conservation a priori. In this work, we discuss the challenges for measuring the power consumption of an individual web service. We discuss the challenges of determining the power consumption profile of a web service each time it is migrated to a new server and the training procedure of the power model. This potentially promotes creating a dynamically-green cloud infrastructure.

Keywords: Energy-awareness, web service, service-oriented software engineering, green web service.

1 Introduction

By applying clean software interfaces to human-based capabilities or legacy software systems [11], web services are modular, network accessible software applications that promote the open sharing of domain-specific capabilities across organizations. With the current priority on sustainability, attention must be placed on the energy-efficiency of IT assets in all organizations. While the large majority of work in energy-awareness concentrates on measuring and repurposing hardware resources, the aim of this paper is to understand how to allocate the specific software service that is most energy-efficient. This paper provides insight into the problem of determining the power consumed when processing a particular web service operation. This work leverages the fact that one web service can provide multiple operations while also potentially being replicated on multiple

G. Pallis et al. (Eds.): ICSOC 2011, LNCS 7221, pp. 87–96, 2012.

servers. More specifically with respect to energy usage, the *same* operation is accessible at multiple places; however, invoking the *same* operation on *different* servers might result in varying degrees of power consumption. It is also important to realize, that the *same* operation invoked on the *same* server, but at a *different* time, also may result in a *different* power consumption. In other words, the power consumption changes over time and across regions. This is mainly caused by the fact that invoking the same operation during distinct states of the server, mainly characterized by the utilization of its hardware resources, does not result in performing exactly the same computation. This means that the power consumed when processing a particular web service operation request is not invariant according to time and requires *dynamic* determination.

Several factors must be considered when determining the power consumed by web services. The power estimation model must provide power estimations relative to distinct possible states of the server. Thus, if the state of the server at the time the request is to be processed is known, we can provide accurate consumption estimation for a particular web service on that server. Since the future state of the server is unpredictable, this approach assumes that the server state changes negligibly in the short period of time that it takes to execute a typical web service. We anticipate that our approach will be effective for providing *dynamic* power estimation information to *environmentally-aware* web service management systems that dynamically discover and compose web services across clouds [1,2,12]. This approach will promote effect decision-making when selecting the particular instance of the service (service running on a concrete server) that consumes the *least energy* of multiple redundant options. This approach that we call *green web services* is an inevitable step towards the realization of sustainable cloud environments.

Our work addresses the following research questions:

- *What are the challenges of evaluating the power consumption of web services?*
- *Which hardware conditions, as measure by system performance monitors or counters, should be used and how should the data be monitored and collected?*
- *How is a model for a specific web service achieved? How can a model be devised through a training process?*

2 Power Modeling Background and Related Work

State-of-the-art studies have demonstrated that relatively precise computer power consumption estimations can be derived from the readily-available hardware performance system measurement tools. In this context, the challenge is to: 1) choose the appropriate counters that most effectively capture the state of the computer, 2) define the mathematical model calculating the estimated power from the computer state, and 3) define the power model training process.

There are currently several power estimation models in related literature [3,5,6,10,9,4,8]. Through these models, an estimation of the power consumption

of the server, under various workloads, based on readily available system measurement tools can be created. Consistently across the related work, the power estimation models leverage system measurement tools or counters that monitor the hardware conditions on the machines for which they reside. These counters measure conditions such as the *usage of different CPU instructions*, *various memory access operations*, and *distinct hard drive access*. The power estimation model is represented by a function of these attributes. This function is usually a linear combination of the attributes, $X_1, \ldots, X_n$, with an additional constant representing the idle power consumption: $P = P_{idle} + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n$, i.e. the model is represented by the coefficients, β, of the attributes with additional constant for the idle power. The values of these coefficients are determined during a training process by reggression.

Related Work. The authors of [9] present an approach for run-time modeling and estimation of operating system power consumption. They focus on instantaneous estimation of the power consumed when executing OS-intensive workloads (file management, compilation, database operations). One of the main results of the study is the observation of a high correlation between the IPC and power. In [4], the authors present a power estimation model for the Intel XScale®PXA255 processors. The approach exploits the insight into the internal architecture and operation of the processor. It achieves accuracy within an average of 4% error using five processor performance counters. The authors of [8] focus on power consumption estimation of virtual machines running on the same physical machine. Their solution is implemented by their customized tool, *Joulemeter*. It uses a similar power model as the other mentioned approaches. However, their motivation to use performance counters to predict power consumption is different. Direct measurement of the power is possible only for a physical device. If multiple virtual machines are running on one physical machine, it is not possible to measure the power consumption of the virtual machines separately. Since the performance counters can be monitored separately for each virtual machine, they attempt to segregate the power consumption. The power estimation of Joulemeter achieves accuracy, with errors within 0.4W - 2.4W.

Our Approach. Our experiments were performed using a model represented as a linear combination of the following attributes: number of received/sent packets, instructions executed, CPU cycles (the counter increments when the core clock signal is not halted, i.e. it is varying according to the changing load), *IPC* (i.e. instruction divided by cycles), percentage of *non-idle CPU* time, and *last level cache misses*. The values of the weights (i.e. beta vector) are acquired by performing a *training stage*, during which, samples of the measured attribute values and the resulting power are collected. The power value is retrieved using a physical power meter that is connected to the server. After the samples are collected, a regression method based on least squares is used to evaluate the power weights. Note that the power meter is only required during the training stage. This paper presents an insight to the specific problems related to the power estimation of web services. We evaluate different training processes through experiments

and show which approach results in a model estimating the instantaneous power (measured in Watts) during web service workloads the best. Note that in the case of web service workloads, the power model can only be used to estimate the total power. To estimate the power consumption (measured in Watt hours) of one particular request contributing to the workload, the total power must be split and the time dimension must be considered. Our preliminary approach performs this according to the number of actually processed requests and the execution time.

3 Web Service Power Estimation Challenges

Isolating the Specific Web Service. To determine the power consumption of a specific web service, the process should clearly segregate the impact of a particular request. In general operations, web servers continuously process multiple parallel web service requests. Each particular request might correspond to the execution of distinct web service operation or operations (when considering a composition service). Web services exploit different hardware resources, such as CPU, memory, hard drives. These resources all represent some part of the total power consumed as a result of the web service processing. Since the resources are *shared* and exploited in *parallel* by multiple requests, it is not feasible to determine the portion of the consumed power related to each of the requests. As such, a physical power meter cannot be used to measure the power consumed by one particular request. The power meter, generally, can only measure the overall power consumption of the server.

Web Service Executions Are Inherently Short in Time Duration. Since several web services execute in relatively short time, e.g. in milliseconds, the general power meters do not effectively capture accurate measurements at this scale. Furthermore, synchronizing the measurement with the processing of the request by the server is a challenge. Thus, isolating the power consumption of web service requests requires a custom model informed by the SOA paradigm. This model must estimate the power consumed when processing a particular web service operation request on a concrete server at a given moment.

Power Consumption Depends on the Actual Server State. Our preliminary experiments showed strong dependence between the actual server state and the power consumed to process a web service request. Significant differences are observed even when the same web service operation is executed with the same inputs. The most influencing factor, describing the server state, is the CPU load. Our results show that the nominal power consumption of one request is much higher when the server is under-utilized. As the utilization rises, the differences are lower. In our experiments, the nominal power consumption, while the server was under-utilized, was in average 3 times higher than the consumption when the utilization was high. However, at very high utilization rates, the consumption rised again. Thus, the same results when measuring the power consumption are only anticipated if the same circumstances are guaranteed.

Relationship between Input Data and Service Computation Is Unpredictable. The knowledge of the inputs in advance is limited and in most of the practical scenarios this information is not available. In general we can expect a relation between the size, and the structure of the inputs and the complexity of the computation. Considering the (de-)serialization of the I/O into/from messages, large messages and more complex structures are more computationally demanding. The dependence at the execution phase does not necessarily have to hold. However, for many types of web services there is an obvious dependence. For example, in the case of a web service sorting numbers, there is a defined relation between the size of the input sequence and the amount of required computation affecting the power consumption. As a result, for some web services, an effective approach is to determine the power estimation by referencing a value calculated when the service was invoked using a representative set of inputs. This value might be defined as the average, or maximum - similar to the response time determined when evaluating the QoS characteristics of the service.

4 Practical Problems When Building Power Estimation Models

To build the power model, data must be collected using distinct software and/or hardware components. To train the model correctly, the data collection must be synchronized. Misalignment of data samples introduces errors in the power estimation, so a precise sampling with respect to their alignment is a challenge. The following types of data must be collected: 1) hardware performance counters, 2) web service execution statistics, and 3) power measurements. There are several software tools and devices that provide access and facilitate the measurement of the data required to build the power model for the server.

Monitoring System/Hardware Conditions. Measuring system performance is possible due to built-in registries holding data related to specific events related to the hardware, e.g. number of instructions, cycles, and last level cache misses in CPU. The number of these registries on a CPU, i.e. counters incrementing when the event occurs, is limited. Each particular processor model provides a set of events, which might be measured by the programmable counter. Using these, it is possible to develop various on-demand monitors. On some processors, the length of the registries storing performance data is too low. This causes frequent overflows which then must be handled at higher application level. Some performance statistics, such as network and hard drive traffic, are monitored at the operating system (OS) level. Thus, OS specific tools must be used to access the statistics. In our experiments, we used the *Dell PowerEdge SC 1430* server, with two *Intel Xeon 5130* dual core processors, running *Ubuntu 10.04*. The network traffic and the percentage of the non-idle CPU time were collected using a *Libstatgrab library*, written in C. The CPU instructions, cycles, and last level cache misses were measured by *Intel Performance Monitor Counter*, written in

C++. These applications showed high reliability during the experiments. Other useful tools include:

- *Libstatgrab library*
  `https://launchpad.net/ubuntu/+source/libstatgrab/0.17-0ubuntu1`
- *Microsoft Windows Performance monitor*
  `http://technet.microsoft.com/en-us/library/cc749249.aspx`
- *Windows Management Instrumentation* `http://msdn.microsoft.com/en-us/library/windows/desktop/aa394582%28v=vs.85%29.aspx`
- *V-Tune Amplifier XE*
  `http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/`
- *Microsoft Visual studio: Premium and Ultimate version.*
  `http://www.microsoft.com/visualstudio/en-us`

Web Service Execution Statistics. Web service related statistics, such as the number of requests and the corresponding response time, are natively supported by different web servers. This also means that the access to these statistics is platform dependent. In Java environment the *servlet filters* facilitate the collection of data. A filter can be implemented such that it triggers custom code each time the server receives a request and when it sends back the response to the client. On ASP.NET platform, *intercepting filters* are equivalent. In our work, the Java servlet filters were used.

Power Measurements. Measuring the power consumed by a computer usually requires an external physical power meter. Although, hardware-level measurement of the computer power consumption is available on some platforms, e.g. [7], these measurements generally do not occur a priori. There are several attainable power meters that provide an interface to communicate with computers. The communication is usually based on USB or Internet but is also accompanied by a noticeable delay. Accessing the data from a device requires custom applications. The device itself and the related applications generally do not provide the data at defined moments. Thus, synchronization with other measurements performed on a computer is limited. In our work, we use the *Watts UP .net* power meter connected via USB. To access its data, we enhance the linux application available at (`https://www.wattsupmeters.com`), written in C.

Synchronizing All Measurements. It is clear from the previous discussion that collecting all the necessary data requires multiple applications running concurrently. The synchronization of these diverse applications, running in parallel, presents a big technical challenge. In our work, a central Java application manages the overall process by invoking the underlying applications in parallel. The inter-application control is performed using *Java Native Interface* in the case of PCM, and using the `Runtime.exec()` method for the rest of the application. Since communication with the Watts UP device is accompanied by a non-negligible delay, the central application synchronizes all the other applications according to a signal received from the application handling the Watts UP device. The central application collects a defined number of samples, each of them

taken during a specified time period. Ultimately, the applications are synchronized only at the beginning of the data collection to limit the inter-application communication. Our experiments show, that the applications remain sufficiently synchronized even if the specified time period is 100 seconds. The misalignment at the end of the collection is few milliseconds.

5 Training the Power Model

Our approach to generate a power model, while addressing the specific requirements of web service workloads, is based on a controlled simulation of web service requests. The process recreates a variety of circumstances under which the server is monitored and performance measurements are captured. We developed a set of 6 different synthetic web services, each having 3 - 4 operations (20 in total), which are invoked during the simulation. Each web service tends to utilize distinct hardware resources, i.e. CPU, memory, hard drive, or a varied combination.

The services used to test the model were created by wrapping operations available in a mathematical library *Apache commons library* `http://commons.apache.org/`. We implemented 11 distinct web service operations which varied in the nature of their computation. Their execution time varies from a few milliseconds up to 10s of milliseconds. We created two testsets using these web services. In the first case, the service workload was generated using the same operation over a defined period of time. Subsequently, another operation was executed in the same fashion. This environment emulates a server that executes a smaller, limited set of web services at a time. The second testset was created by randomly selecting the operation before every invocation. In both cases, we simulated multiple clients independently invoking the web services concurrently. While performing the workloads, we collected performance counter data to be used as inputs for the model generation.

By running different configurations of our test environment and varying the number of clients (running on the same network), we emulated web service workloads. In determining how to invoke the synthetic web services for training the power model, we tested two simulation strategies. In our first simulation strategy, the client randomly selects the requested operation before every invocation, i.e. the requested operation changes over time. In the case of multiple clients, this may result in parallel execution of different operations. In the second strategy, the client continuously invokes the same web service operation, specified by the training manager. The training manager starts the clients, collects the required number of samples, and terminates the clients. In the case of the second strategy, the training manager also specifies the specific web service operation thus cycling through all the operations.

In our experiments, we tested the power model using three public benchmarks: *CPUburn* utilizing the CPU - the *burnBX* command was used `http://linux.softpedia.com/get/System/Diagnostics/cpuburn-1407.shtml`, *Tiobench* for hard drive IO operations `http://linux.die.net/man/1/tiobench`, *MBW* testing the memory `http://manpages.ubuntu.com/manpages/lucid/man1/`

`mbw.1.html` . The power model (built using the benchmark workloads) achieved a precision within a 4.84% error. It was less accurate than the models built using the synthetic services-based workloads. We anticipate, that this difference in error was a result of the benchmark workloads not incorporating web service-specific computations, e.g. serialization of the I/O and the overall management of the request processing.

An important issue when training the power model is determining the number of samples to collect. In general, more samples result in a more accurate model. We collected 3200 samples, sampled every second, for both strategies. Tab.1 presents the mean relative error of the power model according to the number of samples. The table illustrates the error separately for all of the test web service operations, the average through all operations, and the error when the operations were selected randomly. The different columns represent what portion of the 3200 collected samples were used to train the model (i.e. 10%, or 100%), for the two strategies S_1 and S_2. The mean error of the model built from 320 samples while performing strategy S_1 is 3.86%. S_2 achieves better results. The error in this case is 2.49%. When using all the 3200 samples to train the model, S_1 achieves mean error 1.67%. S_2 results almost the same error, 1.66%. Considering our server, this error corresponds approximately to 3.3 Watts. The bottom line of Tab.1 shows that estimating the power while the service workload is made by random operations is more accurate. Considering the number of required samples, we can conclude that S_1 requires more samples to achieve favorable accuracy. When the number of samples is sufficiently high, it achieves similar results as S_2. In this case, S_1 showed to be more accurate for the randomly selected service workload. In the case that fewer samples are available for training, S_2 achieves better results. Fig.1 presents the error of the power model built using strategy S_2 with 3200 samples. Each point of the graph represents one pair of the measured values and its corresponding estimation retrieved from the model. Ideally, the points should lie on a line y = x. The points above the line are overestimations of the power and the points below the line represent underestimation.

Note that fewer samples do not necessarily cause higher error. Consider for example results for the *TestAddhoc* web service operations and strategy S_2. The error is lower in the case when fewer samples were used. This means that the model built using the selected 10% of the samples better characterizes the power consumption while executing *TestAddhoc*. However, for several other operations, this leads to lower accuracy, i.e. the model is less universal.

We also experimented with the training process executing locally on the server. The experiments demonstrated that the models built while the service workload was made by remote clients are more stable. It achieves similar results independently on the workload. The model, using our first strategy, built while executing the clients locally achieved low accuracy within 3.0% error. In the case of randomly selected services, the accuracy within 1.0% error, was better than in the case of remote clients. We can conclude that the models built with remote clients are more universal.

Table 1. Power estimation model error

	S_1 **10%**	S_2 **10%**	S_1 **100%**	S_2 **100%**
TestSort	5.28	0.49	1.65	0.57
TestAddhoc	6.27	0.53	1.69	0.64
TestAxisAngle	6.21	0.47	1.59	0.56
TestCircleFit	1.95	5.67	3.48	4.09
TestExactIntegration	4.24	0.59	1.54	0.60
TestLongly	0.82	4.16	1.67	2.56
TestLonglySpearman	2.28	4.10	0.85	2.20
TestError	2.39	1.38	1.92	1.07
TestQRColumnPermut	5.18	1.58	2.87	2.53
Testwave	4.65	3.08	0.85	1.00
Testplane	3.99	3.84	0.53	1.77
Average	3.86	2.49	1.67	1.66
Random selection	2.95	1.5	1.25	1.3

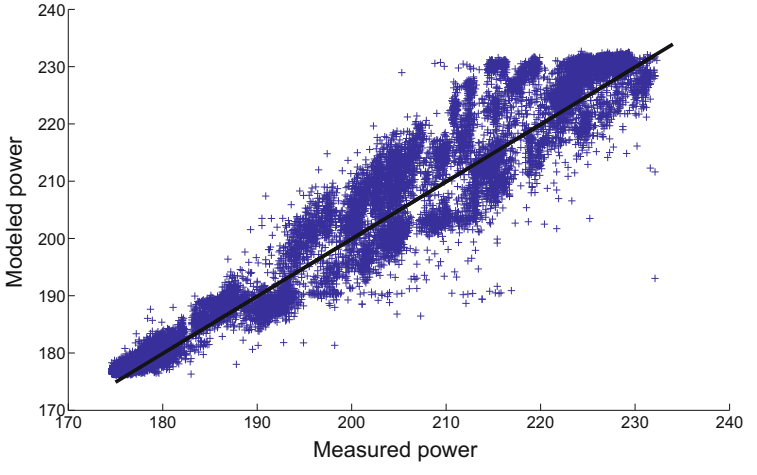

Fig. 1. Power estimation model error

6 Conclusions

This paper explores server-side power consumption of web services. We discuss the conceptual and practical problems when estimating the power required during web service workloads. The approach is based on a model that derives the power estimation from a variety of computer hardware conditions. We present an evaluative body of knowledge of the training process, which is a critical part of building the power model. We performed several experiments to investigate what aspects affect the ability of the model to estimate the power consumed by a particular web service. Our experiments demonstrated that the most accurate universal model is built when the workload during the training is made by remote clients. We considered two strategies to create service workloads. In first

strategy, one operation is selected to be executed for a defined period of time. In the second strategy, the operation is randomly selected before every invocation. Assuming that a sufficient number of samples are recorded during the training, there is no significant difference between the two strategies. The strategy executing one web service at a time showed more accurate results even when the model is built from fewer samples.

Acknowledgments. The authors would like to thank Mr. Chris Ketant of Rochester Institute of Technology for insights and his development of several of the synthetic web services. The authors would like recognize the interaction with Dr. Sekou Remy in comparing our regression approaches to relevant neural network approaches for training the greenness web services. This work benefited greatly from discussions with Tanya Salyers, Department of Mathematics, University of Notre Dame and Dr. Roman Dementiev from Intel. This project was partially supported by NSF Award Number 0512610.

References

1. Bartalos, P., Bielikova, M.: Qos aware semantic web service composition approach considering pre/postconditions. In: IEEE Int. Conf. on Web Services, pp. 345–352 (2010)
2. Bartalos, P., Bielikova, M.: Automatic dynamic web service composition: A survey and problem formalization. Computing and Informatics 30(4), 793–827 (2011)
3. Bircher, W., John, L.: Complete system power estimation: A trickle-down approach based on performance events. In: IEEE International Symposium on Performance Analysis of Systems Software, pp. 158–168 (April 2007)
4. Contreras, G., Martonosi, M.: Power prediction for intel XScaleR processors using performance monitoring unit events. In: Int. Symposium on Low Power Electronics and Design 2005, pp. 221–226. ACM (2005)
5. Economou, D., Rivoire, S., Kozyrakis, C.: Full-system power analysis and mod eling for server environments. In: Workshop on Modeling Benchmarking and Simulation (2006)
6. Fan, X., Dietrich Weber, W., Barroso, L.A.: Power provisioning for a warehouse-sized computer. In: International Symposium on Computer Architecture (2007)
7. Jenne, J., Nijhawan, V., Hormuth, R.: Dell energy smart architecture (desa) for 11g rack and tower servers (2009), `http://www.dell.com`
8. Kansal, A., Zhao, F., Liu, J., Kothari, N., Bhattacharya, A.A.: Virtual machine power metering and provisioning. In: 1st ACM Symposium on Cloud Computing, SoCC 2010, pp. 39–50. ACM, New York (2010)
9. Li, T., John, L.K.: Run-time modeling and estimation of operating system power consumption. SIGMETRICS Perform. Eval. Rev. 31, 160–171 (2003)
10. Rivoire, S., Ranganathan, P., Kozyrakis, C.: A comparison of high-level full-system power models. In: Conference on Power Aware Computing and Systems, HotPower 2008, p. 3. USENIX Association, Berkeley (2008)
11. Schall, D., Dustdar, S., Blake, M.: Programming human and software-based web services. Computer 43(7), 82–85 (2010)
12. Wei, Y., Blake, M.B.: Service-oriented computing and cloud computing: Challenges and opportunities. IEEE Internet Computing 14(6), 72–75 (2010)