Towards Power Consumption Reduction by User Behavior Monitoring at Application level

M.Sc. Imran Asad Gul, Graduate School Trustsoft, Universität Oldenburg, Germany
Dr. Wilhelm Hasselbring, Software Engineering Group, Universität Kiel, Germany

Abstract

This paper gives an overview about our ongoing research which aims at adaptive power consumption optimization in enterprise systems where cost for operating the cooling systems has almost reached near server operating cost. Our methodology focuses at application level. This level can best describe user applications dependence on the underlying system and can play an effective role in power management decisions. We will also present our initial level experimentation which can set a solid base for our research.

1 Introduction

Enterprise systems have evolved into complex software systems with improved computational power i.e. performance because of their innovative and technological design. However demand for improved computational power continues to grow which results in high power consumption and its implications i.e. heat dissipation, hardware reliability [9] and its cooling costs at large data centers give raise to not only environmental issues but also decreases the physical life of the systems. Due to these concerns power optimization has become focal point for researchers. Power optimization doesn't necessarily guarantee energy optimization. Energy and power are interrelated terms. Energy is the amount of work done by a system during a particular time period whereas power is the rate at which system complete that work [22]. Hence energy and power are defined as

$$P = W/T \tag{1}$$

$$E = P \times T \tag{2}$$

where P is power and E_w is energy, T is specified time and W is total work done in that specified time. Energy is measured in *joules* and power in *watts*. The relation between power and energy can be understood by a simple example. If we have a container that can hold a particular amount of liquid for instance 20 litres. A pipe connected with with that container fills the container as and when needed. In this scenario, energy is the amount of liquid that the container can hold at any given time. While power is the rate at which liquid comes into that container.

It depends on the context what is really needed either energy or power. For example, in case of mobile systems, energy optimization would be a better option as this would increase their battery life but in case of enterprise systems i.e. servers, power optimization would be a better option because of high temperature concerns. Furthermore server consolidation due to space limitation in data centers and compact hardware technology such as blade servers further adds to this problem.

It has been widely accepted in literature [9, 10, 11, 14, 16] that power consumption optimization comes at the cost of performance loss which cannot be removed completely but minimized. Central processing units (CPU) are the dominant source of power consumption in enterprise systems. Modern processors support dynamic voltage and frequency scaling (DVFS). Dynamic power consumption is proportional to the operating frequency and square of voltage. This means that operating at lower frequency level can reduce considerable amount of power consumption per CPU cycle. DVFS refers to the scaling (low or high) of frequency and voltage supply depending on the workload. This means that an effective DVFS strategy needs accurate prediction of workload to have minimal effect on performance.

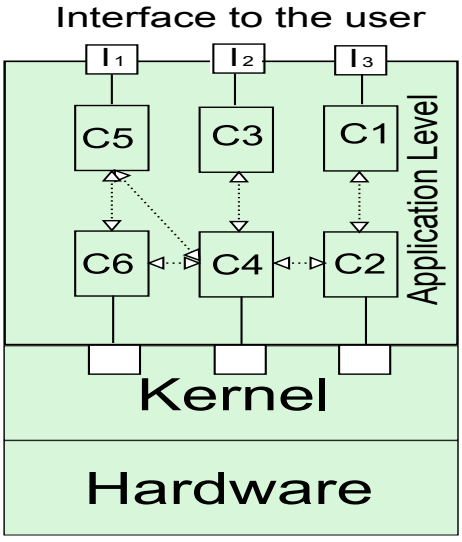

Figure 1: Application level

In this paper, we present (our work in progress) application's operation [21] based methodology for power consumption optimization by user behavior monitoring at application level. Application level refers to the top most level which is exposed to the user where user can be human, or another system. Fig 1 depicts an abstract view of a software and hardware system. User's application which is composed of components resides on the operating system and are accessible through interfaces I1 to I3. We call this level an application level. Because of such a direct contact, applications at this level can best describe their dependencies on the underlying system. Underlying system refers to the system software i.e. operating system and physical hardware i.e. processor and memory etc. Our work aims at identifying execution areas of an application that require less computational power. This information can be used for an effective DVFS having little or no effect on performance. Our approach comprises the following steps

1. Analyzing on-line user bahavior to model current workload. Behavior model shows operations that are being executed and operations that are going to be performed

2. Cost for each operation in terms of resource usage (CPU) is estimated and

3. System adaptation to new parameters accordingly to save power.

The rest of the paper is organized as follows. Section 2 presents motivation of this research. Section 3 describes background and related work. Section 4 discusses our approach. Section 5 our initial experimentation and finally Section 6 concludes and discusses future work.

2 Motivation

2.1 OS power management and its impact on performance

Central processing units are the major energy consumer in enterprise computation systems. Modern CPUs are equipped with power saving and performance states called C-states and P-states. The Advance Configuration Power Interface [1] encourages power optimization by transition of unused components to lower power, in our case it is CPU. [1] defines C-states such as C0, C1, C2 and C3. C0 is fully active state i.e. the CPU execution state while all other states such as C1, C2 and C3 are idle states i.e. CPU execution is slowed down. C3 consumes lower power than C2 and C2's power consumption is lower than that of C1. C-states are managed by operating system power management (OSPM). Depending on the situation, the OSPM policy decides which C-state CPU has to enter. A particular latency time is associated with each C-state. [1] provides an overview of the C-states latency and power consumption, see Table 1. P-state defines frequency and voltage level while the CPU is in its execution state. Processor's running at high clock speed provides better performance but consume high power.

Table 1: ACPI power and latency depiction

C-state	Latency in ms	Power in mW
C1	20	1000
C2	40	750
C3	60	500

Furthermore, performance issues might prevent the CPU to enter in the idle states which means CPU remains active for nothing and a lot of CPU cycles are wasted. There are some obvious situations in which the OSPM C-state selection policy effects performance e.g. Consider a scenario in which an OSPM selects $C3$ which has high latency time (entry latency). During transition (going to idle) to state $C3$ a sudden interrupt arrives for CPU activation than the CPU will have to exit $C3$ state immediately after it enters this state. This would results in a big latency time having noticeable performance degradation.

The situation becomes more worse in case of multi-core processors because any core can enter into any C-state depending on OSPM policy, however the lowest C-state will be opted by the CPU. For example, consider another scenario. If there are two cores $Cr1$ and $Cr2$ and C-states are from $C0$ to $C3$. If depending on the situation OSPM selects $C1$ for $Cr1$ than $C1$ would be enabled automatically for $Cr2$. In a typical situation under light workload where $C3$ would be the best option for $Cr2$ but it could only enter $C2$.

In modern processors like AMD Opteron, Sun UltraSparc and Nehalem, each core can enter any C-state independent of the other e.g. $Cr1$ can enter $C1$ and $Cr2$ can enter $C3$. However, each state has to wait for a particular time before it can enter any C-State. This particular time can be called "remained-idle" time. For example, the remained-idle time for entering into $C2$ is 2000 ms. This implies that the core can enter into $C2$ only after it remained idle for 2000 ms and no interrupt arrived during that time. Now consider $Cr1$ remained idle for 1900 ms and suddenly an interrupt arrived. Because of that interrupt $Cr1$ would have to wait for another remained-idle lag.

In real world situations, all cores are rarely saturated with non-stationary workload in a multicore system. This implies that in highly diversified environments even if the whole resource (CPU in our case) is not utilized we are highly unsure of the requests that will arrive at the CPU. This would result in the selection of C-state by OSPM. Instead of having benefited from this technology, associated latencies with C-states and their selection mechanism i.e. one C-state at one time results in degradation of performance of overall system and power wastage. Forecasting the CPU's future activity i.e. workload and then implementing dynamic power management policy can potentially not only increase performance but also reduce power consumption. We emphasis on proactive forecasting rather

than reactive forecasting for example forecasting just by analysing the past events, this technique would not help because of unpredictable and changing nature of workload. By proactive (predictive) forecasting we mean monitoring the system for defined time and profiling its behavior along with user behavior. Such information can help to better understand the system under various conditions. Based on such profiled information, workload on the system under various conditions can be modelled and its analysis using machine learning techniques can perform pre-hand forecasting. As the system learns from its behavior, certainty in forecasting becomes more positive which can result in long-term forecasting.
By devising a proactive forecasting strategy, a trade off between performance and power consumption can be minimized. This would also include active monitoring at application level thereby relinquishing OSPM from selecting any C-state and scaling each core according to forecast. Our goal is to increase performance per watt.

2.2 Adaptation

Conceptually a computer system is composed of a software system and a hardware system. The software system can be thought of a layer of components laying on hardware. This implies that the upper layer (software layer) that is exposed to external entities determines for hardware to act. External entities can be other systems or humans. Automation on high scale has brought high complexity in software systems which exponentially has increased continuous human observation to change system's parameters as and when required. Adaptive or self-managed software systems answer this issue. Adaptation refers to change in system's behavior with respect to behavior of the external entities is a main characteristics of autonomous systems. This feature guarantees robust performance in autonomous system domains. Performance is an important attribute when it comes to quality of service that a computer system is supposed to provide. Adaptation of software systems has been realized in [3, 12, 13].

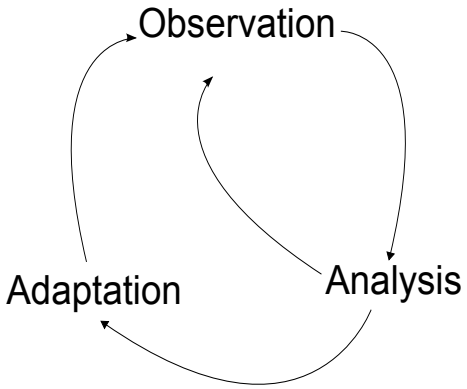

Figure 2: Adaptation cycle

Power optimization schemes trade off performance for power reduction. It can be minimized (as mentioned earlier) if software systems are studied in the context of adaptive systems. For this reason, our research is influenced by [20], continuous adaptation cycle as depicted in Fig 3. which would lay the base for this research. This adaptation cycle is a rough representation of a feed-back loop in autonomous systems. Following are the main operations

- Observe the user behavior to model workload, this refers to the monitoring of user interaction with the system i.e. How many users are active or present? what is the intensity of the requests generated by the users and what is the application's behavior in response?
- Analyse the monitored user behavior, i.e. based on the previous usage patterns, forecast the future patterns for a particular time interval.
- Adapt according to the forecast results to reduce power consumption i.e. components (CPU) which are not under heavy load can slow down or go idle.

3 Background and Related Work

Power consumption and its optimization has always been a key issue in mobile, hand-held and portable devices. However during the last decade, it has become a focal point for enterprise systems. High power consumption leads to considerable heat dissipation which not only increases cooling cost but it also adversely effects the reliability [9]. This gets worst at data centers where cooling apparatus requires from 50% to nearly the same of the power that computational hardware needs [5].
Power consumption can be differentiated into (i) dynamic power consumption and (ii) static power consumption and hence can be computed by the following equation.

$$P_T = P_D + P_S \tag{3}$$

where P_T is total power consumption, P_D is dynamic power consumption and P_S is static power consumption. We will restrict ourselves to dynamic power consumption which is

$$P_D = ACfV^2 \tag{4}$$

where A is the activity factor i.e fraction of the circuit which is switching. V is voltage supply, f is clock frequency and C is physical capacitance.
There exist four ways by which dynamic power consumption can be optimized [22] and each way possess its own trades-off

1. By reducing the capacitance. This can be done by reducing the size of transistors. This technique has adverse effect on performance
2. By reducing the switching activity i.e. clock gating means to activate the logic block only when it is

needed. It is an effective way of reducing both power and energy in processors and widely used in Pentium 4

3. By reducing the clock frequency but it also has negative effect on performance thereby increasing the execution time of the task

4. By reducing voltage supply because it increases gate delays but doesn't provide good results if used independently

Since P_D is proportional to the operating frequency and square of voltage, the combination of last two techniques i.e. frequency scaling and voltage scaling called dynamic voltage and frequency scaling (DVFS) can give optimal results for power and energy optimization and is most widely used technique. This implies that operating at lower frequency level can reduce considerable amount of power consumption. Dynamic voltage and frequency scaling (DVFS) refers to scaling of both frequency and supply voltage to reduce power consumption. The workload on the processor varies at different time intervals and when there is less workload, frequency and voltage supply can be lowered to save power consumption with having little or no effect on the performance. DVFS is an efficient technique for power saving but complexity arises when it comes to the prediction of workload. Workload prediction or forecasting refers to the estimation for the task in terms of time and resources. Workload has unpredictable nature so it is difficult to forecast future workload with reasonable accuracy however different statistical and machine learning techniques can be used to address this issue. Most of the existing DVFS approaches lies in three categories [22].

1. **Interval-based approaches:** In interval-based approaches, DVFS estimates the intervals for which the processor remains busy and idle i.e it forecasts the workload and scale the frequency accordingly. Forecasting can be efficient if workload is regular which is not the case. Because of irregularity, forecasting becomes very difficult if not impossible.

2. **Intertask approaches:** Intertask based approaches are task oriented which means, frequencies are assigned to different tasks depending on the estimation and it remains the same during the course of the task's execution. Besides irregularity in workload which creates problems in forecasting and estimation, these approaches do not consider internal structure of the program which can really improve DVFS.

3. **Intratask approaches:** Intratask approaches are also task oriented. They estimate the task execution paths within a task as different paths require different frequency/voltage scaling. Different execution paths require different amount of cycles to execute.

There exists three levels at which power optimization can be made [11].

- Computer architectural level refers to the energy efficient design of hardware e.g. hardware such as processors and I/O devices can go to sleep state if they are not in use

- System level refers to the efficient use of underlying hardware

- Application level, the top most level which is directly exposed to the user. This level can best describe the system behavior as a result of user behavior i.e. application need for resources such as CPU and its impact on performance.

Operating system power management under the umbrella of "software power management" remained an area of focus for many researchers e.g. [7, 15, 16].
Application-level power management is a new concept as compared to hardware controlled power management and needs to be explored. Pereira et al. [19] puts forward a concept of application guidance. This so called guidance or hint information is gathered by an API responsible for communication between application and the operating system. Guidance information is transferred down to operating system via another API to make better decision for power optimization i.e. Another approach by Anand et al. [2] where application decides which resource to use based on its cost. i.e. If local storage is turned down and could cost more power and latency time to wakeup, it would probably be cheap to use network to fetch the data.
Hotta et al. [10] proposed a profile based DVFS strategy. Program is divided into regions which are instrumented to measure the optimal execution time and power consumption for each region during "trial runs". During actual run, program behavior is adapted according to the profile. Our approach is similar to this because we also build a profile which we call learning phase, however our profile is not power profile, we just model the user behavior.
An effective DVFS strategy tries to minimize the power consumption by satisfying performance constraints however Miyoshi at al. [18] proposed that completing the task at peak frequency and then entering to the sleep state can conserve considerable power consumption as compared to running at low frequency. Similarly execution at lower frequency level will conserve energy but execution time gets increased thereby increasing the total power consumption [14]
We argue that running the task at the frequency which is required can save more energy as compared to throttling first and then entering into deep sleep state. This is because entering into deep sleep state and then exiting for the same would have particular latency attached with it. Though DVFS at different levels of frequencies and voltage has also transition times but latest innovations in the design of microprocessors towards power optimizations has reduced

these transitions cost to the minimum level, making DVFS an efficient way for power reduction.

Another approach called "PowerNap" is proposed by Meisner at al. [17]. It is based on energy conservation of the entire system or the components of the system that are not in use by transitions from active or high performance state to idle state.

4 Our Approach

Our approach centers around the idea of giving user application a wider role in management decisions particularly in power management. This refers to the autonomic or adaptive behavior of an application towards power saving strategies. Applications that have tendency to adapt according to their environment are self-managed or adaptive software systems in the domain of autonomic computing [12]. Software adaptation as described by [20] is a three phase cycle depicted in Figure 3. Observation refers to the monitoring of software system to get useful information about the environment. Analysis is the step where decision are opted based on monitoring observations and finally adaptation is the phase where these decision are implemented for example[1], if an observation of a task shows that it requires 10 ms to complete and analysis shows that its deadline is 100 ms and suggests the lowering of frequency by 1/10 cycles/second. Then adaptation would be to adapt to the suggested frequency level.

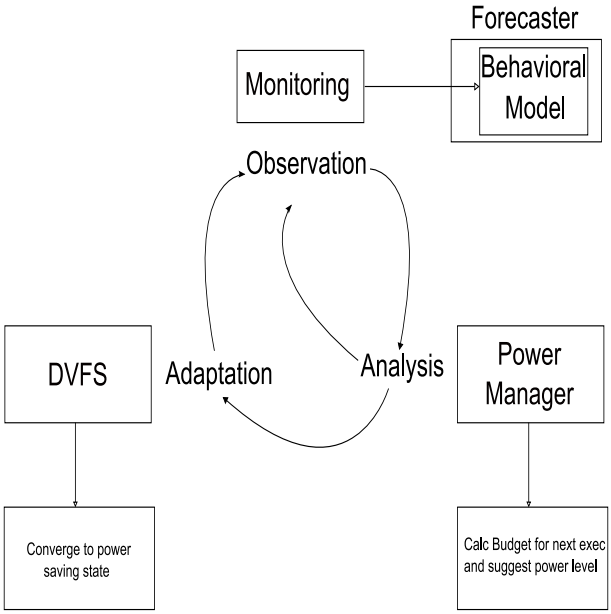

Figure 3: Adaptation Cycle

[1]this example is taken from [22]

4.1 Self-management in our context

Applications are composed of components which provide operations that might be requested by other components, external users or systems. A sequence of operations represents a complete request from its start to end of an execution called execution trace. Our goal is to identify the regions of execution traces which are not CPU intensive. This implies that certain operations may require low computational power as compare to others in whole execution traces. Applying DVFS during those operations saves considerable power at the cost of little performance loss. We refer to performance loss as transition time and transition energy called transition cost [4]. The basic idea of application level power management requires high level involvement of an application in power management decision making [22].

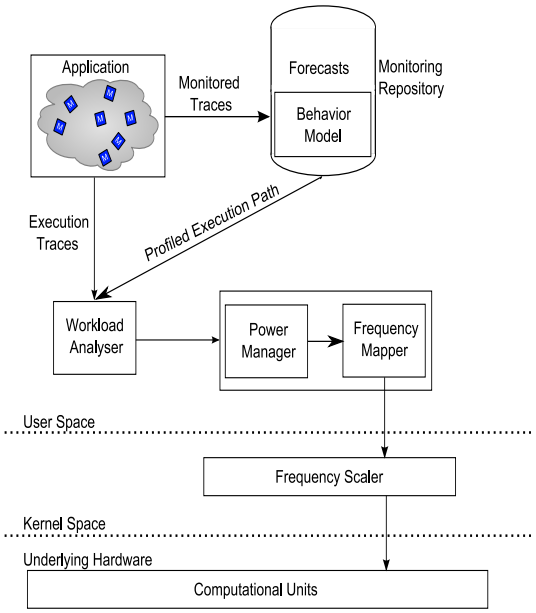

Figure 4: Conceptual Design

Figure 4 depicts our approach towards power optimization. The main components of our approach are forecasting based on behavior model, power manager and frequency mapper. These three components are representation of the logical phases of adaptation cycle. Behavior model is the observation phase which is based on monitoring the application. Power manager is the analyser that can suggest adaptation operations based on the observation. Frequency mapper that normalises the suggested adaptation operations which are then propagated down to hardware. We discuss them in detail.

4.2 Forecasting

As mentioned in Section 1, an efficient DVFS needs effective forecasting of workload. Workload refers to the

number of tasks being processed by the CPU. Workload has non-deterministic nature which makes it very difficult to exactly forecast what is coming next. For example, In case of web server. User interaction invokes certain components to execute a task. It is very difficult to predict user's next move or in other words what is going to execute next. We overcome this problem through behavior modelling by continuous application monitoring which we call learning phase. A learned behavior model is a so called pattern which depicts each and every walk of user. Based on such pattern, we can make near to accurate forecasts.

4.2.1 Behavior Model

Behavior model represents user interactions with the system. We use Markov chains to model user behavior from monitoring data as a result of continuous monitoring [21]. We use markov chains because they provide a common stochastic means to describe dynamic system behavior. For detail discussion, we refer to our previous work [21] and [8]

Monitoring The application is instrumented with monitoring prob which gets monitoring data called monitoring record and put into the repository. Each record provides information about individual operation executions such as start time, end time and operation name etc.

Clustering However, in case of web or application servers, more than one user accesses the system which results in a huge number of Markov chains having many of them identical or with little difference. Clustering is used to group related Markov chains to get accumulative behavior model.

Fig 5 represents a behavioral model of the users interactions with a sample application that we used for our initial experimentation. Behavioral model is a graphical representation of user's walk during its session.

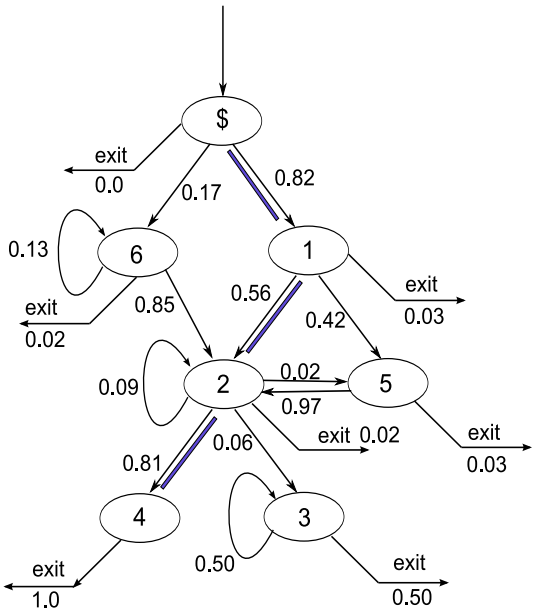

Figure 5: Adaptation cycle

The $ symbol represents the start of user interaction. Because of space limitation, we numbered the operations or methods that were instrumented. As we can see in Fig 5 that certain operations have high probabilities. This means that those operations were accessed by the most users.

4.3 Power Manager

Power manager compute the current workload on the system. It then assigns an appropriate frequency for the calculated workload to save power by taking following considerations into account.

- How much resource (CPU) is required for the given operation. This also refers to the deadline (defined by SLAs) which has to be met in order to conserve performance and how much of the CPU time is actually available for the given operation?
- What should be the appropriate frequency for the given operation?
- Based on the above calculation, adapt to the new frequency level if required

The low power consumption goal cannot be achieved if the transition time and energy is higher than the energy consumed by an operation as a result of the above-mentioned calculations even if an operation execution would still meet the deadline.

4.3.1 Workload Characterization

Workload characterization means deriving a workload model that can best describe its behavior [6]. Our goal (work in progress) is to derive a workload model that refers

to the aggregate of the requests by the users that are accessing certain operations. Currently, we use our behavioral model that can not only model upcoming operations that a user is going to perform but can also time stamps such start time and end time for a given operation during learning phase as discussed in Section 4.2.

4.4 Frequency Mapper

It maps the calculated frequency to the nearest frequency that CPU supports. For example, the outcome of power manager is that the given operation requires 150 MHz frequency without violating deadline and transition constraints however, the hardware only supports {200,600,800,1000}MHz, it would be feasible to adapt to 200 MHz.

5 Experimentation

We perform a small experiment as a proof-of-concept which would laid the basis for our future work. For this purpose, we used a small application written in Java that simulates 20 users at a time. The application has an interface which let the users to perform some actions such as

- Searching for a book
- Weather report
- Getting information about some computer game
- Music albem releases

Each action corresponds a method which is instrumented. Table 2 gives the action names with their assigned numbers. We assigned these numbers because of space limitation. User actions with their associated numbers and execution time in micro seconds are shown in Table 2.

Table 2: Actions

Opr name	Assigned num	Mean exec time in ms
menu	$	0
bookSearch	1	148
whatsNew	2	54
weatherReport	3	2432
library	4	215
offers	5	201
gamesInfo	6	10170

5.1 Monitoring

Since the users interaction with the application is probabilistic. This means that during every execution phase, which we refer to our learning phase, each user interacted randomly. During the execution, a behavioral profile for each user is created. 20 users had 20 corresponding profiles or Markov chains. As discussed in Section 4.2.1, related Markov chains are combined together to form a cluster. Since we have a small system, all our behavioral profiles were nearly the same. We clustered them to form a single common behavioral profile as depicted in Figure 5. The dark colored line along the path in Figure 5 represents mostly accessed path by the users.

5.2 Power Manager and adaptation

We used time stamps associated with actions shown in Table 2 to calculate CPU resource needed to perform specific operation. For example, the mean execution time for the operation "weatherReport" is 148 μ seconds and the SLA between user and weatherReport service provider component says that the complete request should be served in 300 μ seconds. Then the CPU frequency could be scaled to the half of its frequency provided if the transition execution time and service provision time do not violate SLA. For the sake of simplicity, we define the response time limit to be 200 μ seconds in our experiment. The experiment was performed using AMD Athlon 64 processor which supports number of frequencies and their associated power as shown in Table 3. For dynamic voltage and frequency scaling, we used linux kernel CPUFreq.

Table 3: AMD Athlon 64 supported frequencies

Frequency (GHZ)	Watt (W)
3.0	131.2
2.8	124.5
2.6	116.5
2.4	113.2
2.2	107.4
2.0	105.6
1.8	103.7
1.0	94.6

It is important to note that these experimentations were performed as a proof-of-concept which would be the basis for our future work. We conducted two kinds of experiments.

- Non-power aware
- Power aware

Non-power aware experiment is without any power saving strategies however, we perform DVFS in power aware experiment. Each experiment can run for a defined amount time with defined number of users. Our experiment lasted from 60 to 75 seconds during which 20 users interact. The response time limit for every operation that user perform is 200 μ seconds.

5.3 Results

For our initial power measurement experiments, we use ordinary multimeter that can take samples every second. Since CPU is one of the power dominant component, its frequency can really effect the overall power consumption.

5.3.1 Non-power Aware System

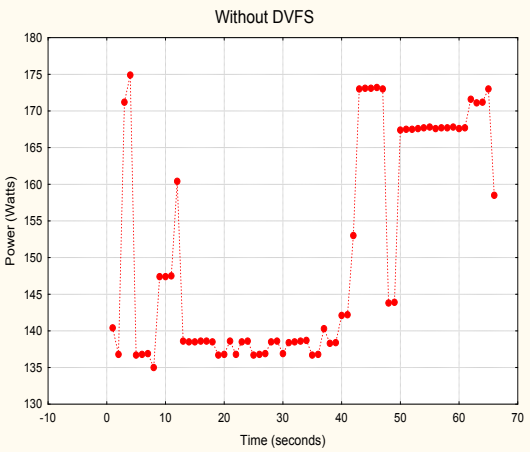

Figure 6: Power consumption in non-power aware system

Figure 6 depicts power consumption of the system during the test run. It can be observed that the system is drawing power above 135 watts per second which is the minimum power level that the system consumes.

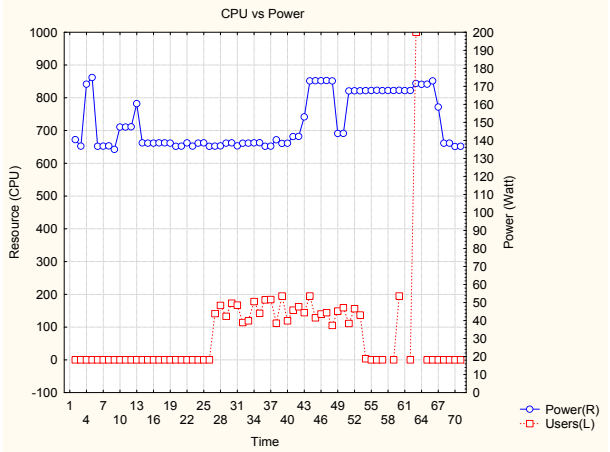

Figure 7: Power consumption in non-power aware system

Figure 7 depicts the users that are interacting along a time line and power consumption while users interaction. The big red spike shows that some particular method or operation took a lot more processing thereby violating the response limit. However, it is interesting to see that majority of the response limits are met.

5.3.2 Power Aware System

Now we present, how a system reacts that possess power aware strategies.

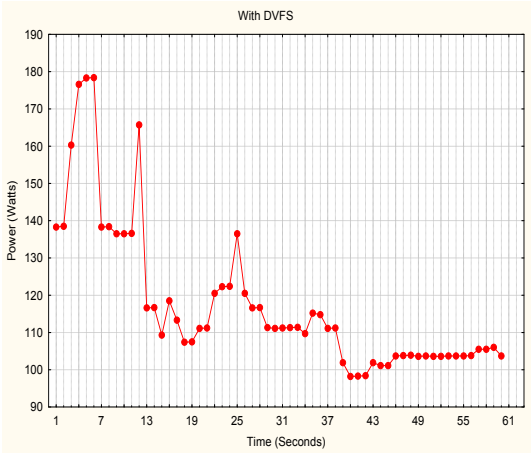

Figure 8: Power consumption in power aware system

As the time line for the experiment increases as shown in Fig 8, the power consumption of the system becomes lower. As discussed earlier, we model each and every walk of the user, our power-aware strategy takes behavioral model into account and perform DVFS before the most probable operations that user would perform.

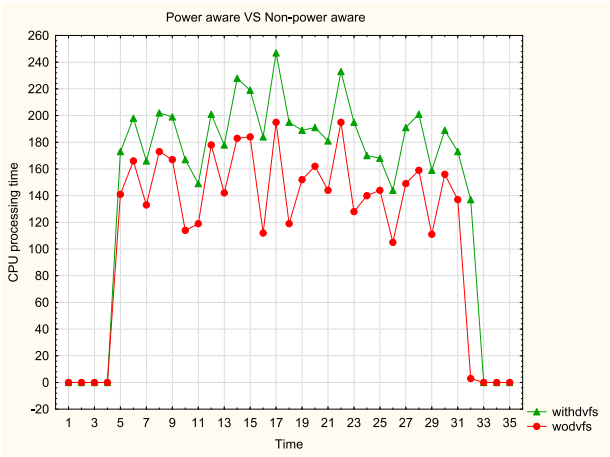

Figure 9: CPU load vs in power aware system

Figure 9 represents a comparison between CPU execution time when the system is non-power aware and when the system is power aware. We can see that there are 7 requests that exceeded beyond the defined response time limit.
Figure 10 compare the power consumption results from both systems, power aware and non-power aware. Interestingly, our power aware system consumed a lot less power as compare to non-power aware system with majority of the requests served under the time limit.

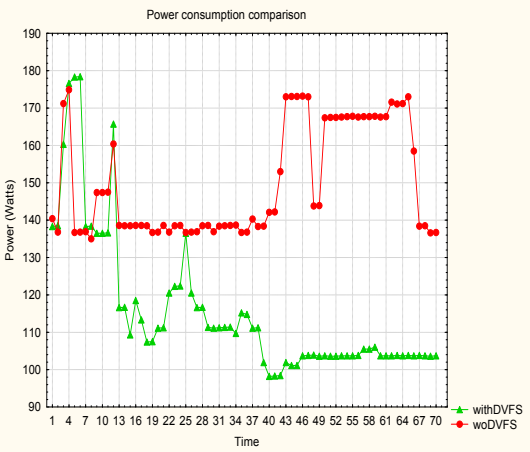

Figure 10: Comparison between power aware and non-power aware system

6 Conclusion and Future Work

In this paper we outlined our proposed approach (in progress) and initial level experimentation for power optimization at application level by user behavior monitoring. High heat dissipation as a result of high power consumption decreases reliability of hardware systems. DVFS is an efficient way of reducing power consumption, however it can only be used efficiently when future workload information is known. We presented our proposed approach for forecasting the workload. Though, our experimentation is based on the request level processing however, our goal is to derive a workload model that refers to the aggregate of the users load on the system.
At a glance, monitoring appears to be an additional overhead however this additional overhead can result in an accurate workload forecast. The power manager computes the lowest frequency and voltage level for the incoming operations which should not violate the deadline and timing constraints and finally it is mapped to the appropriate frequency and voltage supported by the hardware. Below is the future work for our proposed methodology

- As described in Section 4.3, the power manager has to be implemented which needs further exploration of frequency and voltage transition (time and cost). Furthermore studying correlation between time needed for an operation to execute and time the CPU actually has for that operation and its impact on performance.
- Despite of having a near to accurate behavior pattern, certainty cannot be guaranteed. Devising strategies for recovering and recovering from uncertain forecasts and studying its effects on performance.
- Development of a framework that can best evaluate our proposed approach. We plan to have evaluation by simulation and lab experiments by using real client server environment.

References

[1] ACPI. Advanced configuration and power interface spefication http://acpi.info.

[2] M. Anand, E. B. Nightingale, and J. Flinn. Ghosts in the machine: interfaces for better power management. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 23–35, New York, NY, USA, 2004. ACM.

[3] E. Arnautovic, H. Kaindl, and J. Falb. An architecture for gradual transition towards self-managed software systems. *SIGSOFT Softw. Eng. Notes*, 31(6):1–2, 2006.

[4] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A dynamic voltage scaled microprocessor system. *Solid-State Circuits, IEEE Journal of*, 35(11):1571–1580, Nov 2000.

[5] R. S. C. D. Patel, C. E. Bash and M. Beitelmal. Smart cooling of data centers. In *Proceedings of IPACK*, July 2003.

[6] M. Calzarossa, L. Massari, and D. Tessera. Workload characterization issues and methodologies. In G. Haring, C. Lindemann, and M. Reiser, editors, *Performance Evaluation: Origins and Directions*, pages 459–482. Springer-Verlag, 2000. Lect. Notes Comput. Sci. vol. 1769.

[7] C. Gniady, A. R. Butt, Y. C. Hu, and Y.-H. Lu. Program counter-based prediction techniques for dynamic power management. *IEEE Transactions on Computers*, 55(6):641–658, 2006.

[8] I. Gul, N. Sommer, M. Rohr, A. van Hoorn, and W. Hasselbring. Evaluation of control flow traces in software applications for intrusion detection. In *Multitopic Conference, 2008. INMIC 2008. IEEE International*, pages 368–373, Dec. 2008.

[9] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. T. Kandemir, T. Li, and L. K. John. Using complete machine simulation for software power estimation: The SoftWatt approach. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (8th HPCA'02)*, pages 141–150, Boston, MA, USA, Feb. 2002. IEEE Computer Society.

[10] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi. Profile-based optimization of power performance by using dynamic voltage scaling on a PC cluster. In *IPDPS*. IEEE, 2006.

[11] A. Kansal and F. Zhao. Fine-grained energy profiling for power-aware application design. *SIGMETRICS Perform. Eval. Rev.*, 36(2):26–31, 2008.

[12] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan. 2003.

[13] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *FOSE '07: 2007 Future of Software Engineering*, pages 259–268, Washington, DC, USA, 2007. IEEE Computer Society.

[14] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with pace. In *SIGMETRICS '01: Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 50–61, New York, NY, USA, 2001. ACM.

[15] Y.-H. Lu, L. Benini, and G. De Micheli. Low-power task scheduling for multiple devices. In *CODES '00: Proceedings of the eighth international workshop on Hardware/software codesign*, pages 39–43, New York, NY, USA, 2000. ACM.

[16] Y.-H. Lu, L. Benini, and G. De Micheli. Operating-system directed power reduction. In *ISLPED '00: Proceedings of the 2000 international symposium on Low power electronics and design*, pages 37–42, New York, NY, USA, 2000. ACM.

[17] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pages 205–216, New York, NY, USA, 2009. ACM.

[18] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 35–44, New York, NY, USA, 2002. ACM.

[19] C. Pereira, R. Gupta, and M. Srivastava. PASA: A software architecture for building power aware embedded systems. In *Proceedings of the IEEE CAS Workshop on Wireless Communications and Networking - Power efficient wireless ad hoc networks*, Sept. 2002.

[20] M. Rohr, S. Giesecke, W. Hasselbring, M. Hiel, W.-J. van den Heuvel, and H. Weigand. A Classification Scheme for Self-adaptation Research. In *Proceedings of the International Conference on Self-Organization and Autonomous Systems In Computing and Communications (SOAS'2006)*, Sept. 2006.

[21] M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stoever, S. Giesecke, and W. Hasselbring. Kieker: Continuous monitoring and on demand visualization of Java software behavior. In *Proceedings of the IASTED International Conference on Software Engineering 2008 (SE 2008)*, pages 80–85. ACTA Press, Feb. 2008.

[22] V. Venkatachalam and M. Franz. Power reduction techniques for microprocessor systems. *ACM Comput. Surv.*, 37(3):195–237, 2005.