Online Prediction of Battery Lifetime for Embedded and Mobile Devices

Ye Wen, Rich Wolski, and Chandra Krintz

Computer Science Department, University of California, Santa Barbara
{wenye,rich,ckrintz}@cs.ucsb.edu

Abstract. This paper presents a novel, history-based, statistical technique for online battery lifetime prediction. The approach first takes a one-time, full cycle, voltage measurement of a constant load, and uses it to transform the partial voltage curve of the current workload into a form with robust predictability. Based on the transformed history curve, we apply a statistical method to make a lifetime prediction. We investigate the performance of the implementation of our approach on a widely used mobile device (HP iPAQ) running Linux, and compare it to two similar battery prediction technologies: ACPI and Smart Battery. We employ twenty-two constant and variable workloads to verify the efficacy of our approach. Our results show that this approach is efficient, accurate, and able to adapt to different systems and batteries easily.

1 Introduction

Power is a critical resource for battery-powered embedded systems and mobile devices. As such, battery life must be monitored and managed within these systems to ensure maximum efficiency and effective prioritization on behalf of system users. While compile-time optimization of application code can reduce the battery consumption of individual applications, operating system support is needed to manage the combined power consumption of multiple programs executing in concert. Providing this support at the operating system level requires the ability to *predict*, accurately, remaining battery life given a dynamically changing system workload.

In this paper, we investigate an on-line statistical approach to battery lifetime prediction that combines recently observed power dissipation "history" with pre-computed off-line benchmark measurements. By dynamically incorporating on-line measurements, our approach is able to make predictions that take into account varying workloads, the "recovery effect" that batteries experiences when they are unloaded, and the charging-cycle effect that changes battery performance as batteries are repeatedly recharged.

Much of the prior work investigating battery dissipation and prediction is analytical, simulation based, or both [4, 6, 1, 11]. These systems attempt to provide accurate dissipation predictions off-line, for use in design or analytical contexts. Efficient analytical methods such as [14], and [15] consider the problems of on-line prediction, but do not include the statistical components needed to rapidly analyze dynamically changing workloads and operating conditions. While some approaches have considered statistical characteristics in combination with analytical models, they focus exclusively on battery dissipation in isolation [13, 16]. To be useful in an operating system resource management context, however, a battery lifetime prediction technique must be

- **fast** enough to make predictions so that real-time or near real-time decisions can be made,
- **power-efficient** enough to be run on the battery-powered device itself,
- **dynamically adaptive** so that it can take into account different user workloads, and environmental operating conditions (e.g. ambient temperature, battery recharge count, etc.), and
- **portable** so that a variety of battery and device combinations can be supported by the same operating system.

To address these challenges, our approach treats operating system power measurements from the battery as coming from a "black box." We use off-line profiling of the installed battery to establish a *reference signature* for its observed dissipation curve. We then use fast, on-line regression to predict deviations from this signature. Thus, our method uses benchmark data from the battery (in the form of a reference signature) to parameterize a statistical model that we evaluate on-line. Because the system uses measurements taken in the operating system, it is portable between devices and batteries. By using immediate on-line history, the system adapts to dynamic changes in system conditions.

We investigate the efficacy of our work by empirically evaluating our methods using the popular HP iPAQ running the Linux operating system. All of the necessary data for our method is obtained through standard hardware and operating system interfaces provided by Familiar Linux, a commonly used Linux implementation for iPAQ devices. We compare our results to those provided by two native Linux battery lifetime prediction systems: the Advanced Configuration and Power Management Interface (ACPI) and Smart Battery [5]. While considered to provide very rough estimation of battery lifetime, these utilities nonetheless meet the requirements that we describe above. That is, they implement fast, on-line, portable prediction method at the operating system level. Our method combines the attractive online features of ACPI and Smart Battery with prediction accuracy, and thus constitutes an fast, accurate, and adaptive prediction mechanism that can be used as the basis for "power-aware" operating system design.

To describe this work in greater detail, the remainder of this paper is organized as follows. In Section 2, we describe the methodology more completely. Section 3 provides an empirical evaluation of our method through direct experimentation and in Section 5 we draw brief conclusions from our investigation.

2 History-based Battery Lifetime Prediction

Our methodology consists of three components: a *reference signature* from the battery, a *curve transformation function* that changes coordinates to make fast prediction possible, and a *fast linear fitting* technique that makes predictions in the transformed space. The baseline observation that makes this methodology possible is that for constant but differing workloads, the "shape" of the battery dissipation curve is similar. Thus, using the trajectory produced by one workload, the lifetime implied by other constant workloads, can be predicted accurately. By transforming the coordinate space into one where simple linear fitting techniques are applicable, the predictions can then be made using computationally efficient techniques.

2.1 Linearity, Reference Curve and Voltage Curve Transformation

To determine the reference signature of a battery, we execute constant-power workloads on a quiescent system. A constant-power workload consists of repeated executions of single program instance from full battery charge until battery expiration. We describe the individual program instances in Section 3, but for the purpose of describing our methodology, the salient feature is that the power drain is constant with respect to the application workload, e.g., there is only a single program in each workload.

Linux permits application access to the voltage level reported by the battery on the iPAQ. During each complete benchmark run, the power level is recorded periodically to produce a drain trajectory. This trajectory can be expressed by function $F : t \rightarrow v$, mapping time t to battery voltage level v. v's value is between the open circuit voltage (approximately the voltage when the battery is fully charged) and the cut-off voltage (the voltage when the battery dies). In Figure 1(left), we show two typical voltage curves, which are obtained by repeatedly running benchmark programs ("IMem" – a memory read benchmark – and "IMemWC" – a cache-write benchmark – in this case) on an HP iPAQ until the battery dies. The x-axis represents the time and y-axis represents the voltage level. We describe the full experimental setup and benchmark information more completely in Section 3.

The voltage curves in the left graph of Figure 1 are inherently non-linear due to the internal electrochemical characteristics of the battery. This non-linearity limits our ability to predict remaining battery life efficiently. If we treat the trajectories as invertible continuous functions, however, we can make the observation that

$$V = F_1(t_1) = F_2(t_2) \tag{1}$$

for voltage V, and dissipation functions F_1 and F_2. If we use $\mathit{\Gamma}_{1,2}$ to represent the relationship between t_1 and t_2 under F_1 and F_2 for any voltage level V, we have:

$$F_1(\mathit{\Gamma}_{1,2}(t_2)) = F_2(t_2) \tag{2}$$

Furthermore, we can see that the voltage curves of constant workloads have very similar shapes. Based on this shape-similarity, we make the further simplifying assumption that the timing relationship $\mathit{\Gamma}_{i,j}$ for any two constant workloads, F_i and F_j, is a series of functions with the same form but different parameters,

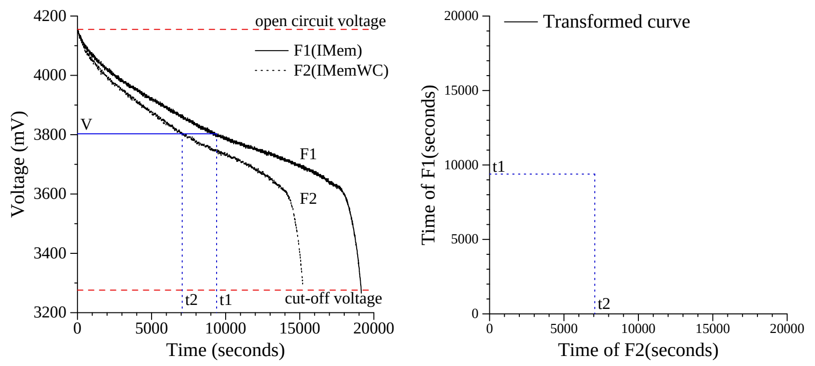


Fig. 1. Timing relationship between two voltage curves of constant workloads. F_1 is the curve of the workload that is generated by repeatedly running benchmark "IMem". And F_2 is the curve of benchmark "IMemWC". The left graph shows that for some voltage value V, F_1 reaches V at time t_1 and F_2 reaches V at time t_2. The right graph shows the linear relationship between t_1 and t_2 for any V.

denoted as $\Gamma(\phi_{i,j}, t)$, where $\phi_{i,j}$ is a specific set of parameters for F_i and F_j. For the curves in Figure 1, we now have:

$$F_1(\Gamma(\phi_{i,j}, t_2)) = F_2(t_2) \tag{3}$$

So:

$$\Gamma(\phi_{i,j}, t_2) = F_1^{-1}(F_2(t_2)) \tag{4}$$

We plot the Γ function for curves F_1 and F_2 in the right graph of Figure 1. The x-axis is the time of F_2 and the y-axis is the time of F_1. In this graph, the Γ curve appears very close to a linear function. Our experiments show that this strong linearity actually exists between any pair of constant workloads. Figure 2(left) shows another three Γ curves for pairs of constant workloads. The axes are similar to those in the right graph of Figure 1.

If we use one specific voltage curve of constant workload as the reference, denoted as F_{ref}, the Γ function between any curve F and the reference curve F_{ref} can be expressed by:

$$\Gamma(\phi, t) = F_{ref}^{-1}(F(t)) \tag{5}$$

Since Γ can be approximated by linear function, let $\phi = (a, b)$, and we have:

$$F_{ref}^{-1}(F(t)) = a * t + b \tag{6}$$

Note that here a and b vary for different constant workloads. The Γ function actually shows not only the timing relationship between two workloads, but also indicates the size of the load: the larger the slope of the curve, the higher is the power consumption and the shorter does the battery lifetime extend. We refer to the Γ function as the *transformed voltage curve*. Using a reference voltage curve, we can transform any non-linear voltage curve of constant workload into a linear

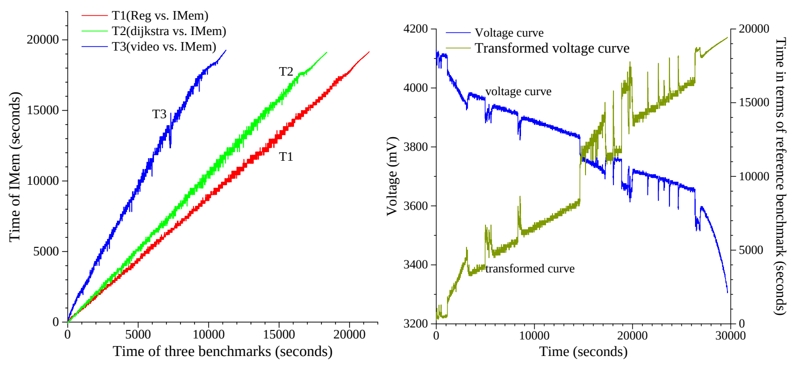


Fig. 2. Timing relationship (Γ function). The left graph plots the timeing relationship for three pairs of constant loads: "Reg" vs. "IMem", "dijkstra" vs. "IMem", "video" vs. "IMem". The right graph shows both the voltage curve and the transformed curve (also based on "IMem") for "real.load.5".

form, which is friendly to fast statistical methods, e.g., linear curve fitting, for remaining battery lifetime prediction.

In real life, workloads may not be constant. We can approximate the power consumption of a variable workload with a piecewise constant curve. Such an approximation is reasonable since the tasks of a workload are composed of consecutive execution of a sequence of operations, whose power consumption can be regarded as constant. Given this assumption, the transformed voltage curve for a variable workload should appear as a piecewise linear curve under ideal conditions.

However, the actual voltage curve of a variable load also exhibits the *recovery effect*. The *recovery effect* refers to the phenomenon that a battery regains some capacity when the load decreases. Through observation, we find that the recovery effect occurs whenever the load changes. If the load decreases, the voltage will "jump up" to a higher value instead of monotonically decreasing. If load increases, the voltage will "jump down" sharply. On the transformed curve, the "jumping" direction is inverted because an advance in time equates to a drop on voltage. Figure 2(right) shows the original voltage curve (from top-left to bottom-right) of a variable workload "real.load.5" and its transformed curve (from bottom-left to top-right) in the same graph. Both curves share the same x-axis, which represents the workload execution time. The left y-axis shows the voltage level for the voltage curve; the right y-axis shows the corresponding time for the transformed voltage curve given the reference curve. Both curves exhibit fluctuations due to the *recovery effect*.

Despite the *recovery effect* and the use of piecewise linear estimation, simple statistical methods can still achieve accurate prediction on the transformed curve for variable workloads. We present this data as part of our empirical evaluation in Section 3. In addition to the *recovery effect*, our curve transformation methodology also captures a number of other challenging battery characteristics that often limit prediction accuracy performance in other techniques [9], e.g., the *rate* *capacity effect* (when the battery is discharged under different workloads, it registers different capacities) and the *cycle aging effect* (battery capacity gradually diminishes after repeatedly being discharged).

Our transformation actually is equivalent to a coordination system switch (from $(time, voltage)$ to $(time, time)$). Since the reference curve is a one-to-one function, we don't lose any information during transformation. Thus, the transformed curve keeps all of the characteristics of the battery discharge that the original voltage curve describes. Based on transformed curve, and due to the nature of the statistical methods we use, our prediction methods are insensitive to all these non-ideal phenomena and can still make an accurate prediction.

2.2 Prediction Methods

In the remainder of the paper, we refer to the transformed voltage curve as the *history curve* since it provides us with a history of battery consumption by the system up to the point at which we make a prediction of remaining battery life. In addition, to make this prediction, we considered a number of different methods. We evaluate each method using the *prediction error* of each. Assume that function P_{t_0} is the function we use to model the history curve, where t_0 is present time. Let v_e be the threshold voltage with which the battery is considered exhausted. Prediction error can be expressed by:

$$error = \left| \frac{L_p - L}{L} \right| = \left| \frac{P_{t_0}^{-1}(u_e) - L}{L} \right| \tag{7}$$

where $u_e = F_{ref}^{-1}(v_e)$, L_p is the predicted lifetime and L is the actual measured lifetime. Figure 3 demonstrates the calculation of prediction error.

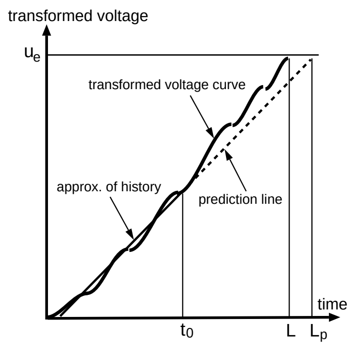


Fig. 3. Prediction error calculation

The first prediction method that we considered uses the average power consumption rate of the history curve to estimate that of the future. According to Equation (6), the slope at any point of the transformed voltage curve indicates the magnitude of the power consumption rate (the ratio between P and P_{ref}). As such, we can model future power consumption at time t as $P_{t_0}(t) = k_{t_0}t$, where k_{t_0} is the mean slope of the history curve before t_0. We refer to this method as *Mean Slope Prediction(MSP)*. We then improved this method by making a prediction line that begins at the current point $(t_0, G(t_0))$ instead of $(0, 0)$: $P_{t_0}(t) = k_{t_0}t + G(t_0) - k_{t_0}t_0$. We call it *Mean Slope Point Prediction(MSPP)*.

We next considered a model that uses linear *Least Square Fit(LSF)*[12]. Assume that k_{lsf} and b_{lsf} are the slope and intercept of the linear regression, the prediction function will be $P_{t_0}(t) = k_{lsf}t+b_{lsf}$. We call this method *LSF Prediction(LSFP)*. As we did for Mean Slope Prediction, we also consider the efficacy of using LSFP when the prediction line starts at the current time (as opposed to the beginning of time $(0, 0)$). We call this method *LSF Point Prediction(LSFPP)*, and implement it as $P_{t_0}(t) = k_{lsf}t + G(t_0) - k_{lsf}t_0$.

All four of these methods are computationally efficient. For example, using method LSFPP or LSFP, a single prediction for a median-sized voltage curve (about $4,000$ readings) takes 250 milliseconds on average on the 206MHz StrongARM processor of the HP iPAQ; this is equivalent to about 0.25 joule of energy consumption. Since MSPP and MSP have lower computational complexity and they can be computed incrementally, they consume even less energy.

Given the history curve and prediction functions, the final question that we must address concerns the length of history that is required to make the best prediction. In Section 3.3, for each prediction method, we empirically evaluate a range of different history sizes to answer this question. This study also gives us insight into how well our techniques perform given a small amount of history.

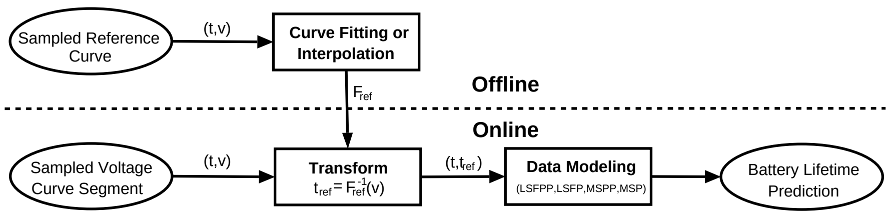


Fig. 4. Prediction procedure.

Figure 4 summarizes the procedure of making a prediction. The current, sampled, voltage curve, which is a sequence of (t, v) pairs (t is a timestamp, v is voltage), is transformed using the reference curve. Next, the above prediction methods are applied to the transformed curve, which is a sequence of (t, t_{ref}) pairs ($t_{ref} = F_{ref}^{-1}(v)$), and a prediction is made. Note that the reference curve must also be transformed from a sequence of discrete pairs to a continuous mathematical form using curve fitting or interpolation. Note also that the online part of the prediction procedure includes the computation of the inverse of *reference curve* function. We use the Newton-Raphson method [12] to make the approximation. This method is also very efficient practically. In our experiment, it takes about 3 iterations on average to get a result with an error within 0.001.

3 Evaluation

We evaluated our prediction methods using two models of HP iPAQ: H3650 and H3835. H3650 is equipped with a 1000 mAh Danionics DLP 305590 lithium-ion polymer battery and H3835 has a 1400 mAh Danionics DLP 345794 battery [3]. Since the results for these two models are similar, we only present the H3835 results in this paper; the trends, however, are the same.

We installed Familiar Linux v0.6.1 [10] and the Opie environment [10] on the iPAQ to perform all experiments. Opie provides the graphical user interface and a set of applications such as games, a media player, and a calendar, that we use as benchmarks. The iPAQ has an internal voltage sensor reporting accurate battery voltage measurements via Linux "/proc" system. We implemented a logging program that reads the current battery voltage from "/proc/asic/battery"(ACPI-like battery status report) into a file stored locally. For each benchmark, we first fully charge the iPAQ battery. We then start the workload benchmark and the logging program simultaneously. The logging program runs periodically with an interval of 6 seconds. We use this empirically selected interval since it is short enough to catch significant changes in voltage and long enough to reduce interruption. The benchmark runs continuously until the battery dies.

3.1 Workload Benchmarks and Reference Benchmark

We generated the constant workload by repeatedly running a single benchmark program, for which one-time execution time is very short (within 4 minutes). We evaluated 14 such programs. They include the benchmarks that we hand-coded to execute of a single type of instruction (*Reg* (register instructions only), *IMem*(loads, out of cache), *IMemC*(loads, in cache), *IMemW* (stores, out of cache) and *IMemWC* (stores, in cache)). In addition, we included *dijkstra*, *fft*, *ispell*, *jpeg*, *sha* and *susan*, from the MiBench Suite [7], and three multimedia programs: *audio*, *video* and *videoaudio*. Each of the MiBench programs represents one of six application categories: network, telecommunication, office, consumer, security and automotive, covering a broad range of typical embedded system applications. The three multimedia programs play MPEG format audio, video and video with audio respectively. All of these programs exercise many hardware functions in an embedded or mobile device, e.g., CPU, memory, flash, audio/video components, and backlight.

We generate part of the variable workloads by simulating the real usage of a PDA. The simulation program is composed of a set of hand-held device applications (e.g. multimedia, note-taking, etc) provided by the Opie toolkit [10]. Each application runs a specified period of time during the simulation. Different patterns of variable workloads are generated by different configurations of the simulation program. The simu.random workload is generated by randomly executing one of these applications in uniform distribution. The other three workloads (simu.30, simu.50 and simu.70) are generated in the following way. The simulation program continuously allocates time slots of random length to either an idle mode or a specific set of applications that are specialized in similar functions (e.g. audio/video), according to a predefined distribution. During

Table 1. Workload description. IMem is used to generate *reference curve* exclusively.

Benchmarks	Type	Comments
Reg	constant: single instruction	register instruction ONLY
IMem	constant: single instruction	memory read instruction, 100% cache miss
IMemC	constant: single instruction	memory read instruction, 100% cache hit
IMemW	constant: single instruction	memory write instruction, 100% cache miss
IMemWC	constant: single Instruction	memory write instruction, 100% cache hit
dijkstra	constant: single operation	shortest path algorithm benchmark
fft	constant: single operation	Fast Fourier Transform benchmark
ispell	constant: single operation	a fast spelling check benchmark
jpeg	constant: single operation	JPEG encoder/decoder benchmark
sha	constant: single operation	SHA secure hashing algorithm benchmark
susan	constant: single operation	image recognition benchmark
audio	constant: single operation	play a 210-second MP3 audio file with audio output, back light off
video	constant: single operation	play a 142-second MPEG1 video file without audio, back light on
videoaudio	constant: single operation	play a 142-second MPEG1 video file with audio output, back light on
simu.random	variable	simulated random workload, no sleep time
simu.30	variable	simulated random workload, 30% probability to sleep
simu.50	variable	simulated random workload, 50% probability to sleep
simu.70	variable	simulated random workload, 70% probability to sleep
real.load.1 ... real.load.5	variable	5 real workloads, voltage curve recorded when PDA is used by people

each non-idle time slot, the applications within the specific set are also executed randomly following a predetermined distribution. In this way, simu.30 keeps the device busy during 70% of the time on average. Similarly, simu.50 and simu.70 have a device usage frequency of 50% and 30% respectively.

We also obtained 5 real variable workloads, whose voltage curves are recorded when users played with the iPAQ in a common way, e.g., playing games, viewing pictures and videos, listening to music, and making a schedule using the calendar. These workloads were obtained by loaning the iPAQ to individual students and then recording the power dissipation each student induced. Table 1 summarizes the total 23 constant and variable workloads.

Finally, we pick the IMem benchmark to generate the *reference curve*. A key contribution of our method is that any constant-workload benchmark can be used to generate *reference curve*; the fluctuations in a variable workload require smoothing if it is to be used as the *reference curve*. The reason for this is that our method relies on similarities in the *shape* of the curves; all constant-workload benchmarks exhibit similar shape, as such, they can be used as the reference curve with statistically similar results.

Since the sampled *reference curve* is composed of a sequence of discrete pairs $(time, voltage)$, we cannot use it to compute the transformed voltage, $F_{ref}^{-1}(v)$, since it is not expressed in terms of v. Instead, we model the reference curve off-line using a high order polynomial and polynomial least square fit [12]. The IMem curve can be fit by a polynomial of order 15 (the coefficient of determination of the fit, R^2, is 0.99955). We then use this polynomial as the *reference curve* function on-line to make each prediction.

3.2 Results of Prediction For Constant Workloads

Given the voltage curve of a constant load, we first generated the history curve (transformed curve) using the reference polynomial. Then, for every 50th point (approximately 5 minutes), we apply each of our prediction methods, LSFPP, LSFP, MSPP, MSP, to make a prediction. We next calculate the error for each prediction point using Equation 7. Finally, we have a sequence of prediction errors ("moving errors") for the entire battery lifetime.

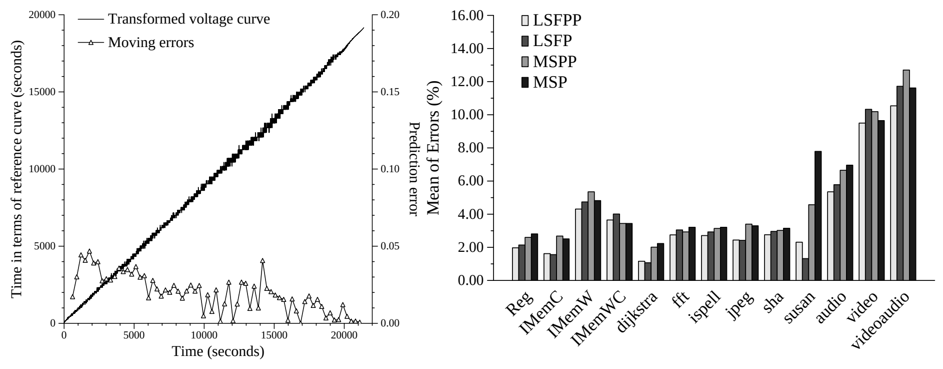


Fig. 5. Prediction performance for constant loads. The left graph shows the transformed voltage curve and moving errors for "Reg". The right graph shows the average prediction errors for all constant loads and all methods.

Figure 5(left) shows the history curve (solid line curve) and the corresponding "moving errors" (marked-line curve) for the Reg benchmark. Both curves share the same x-axis, which represents time. The transformed voltage curve uses the left y-axis that shows the transformed voltage in terms of reference curve time. The error curve uses the right y-axis that shows the error value. As we can see from the graph, all prediction errors are within 5%.

In Figure 5(right), we show the average prediction error for each benchmark and each method. The y-axis is average percentage error.

Except for the three media benchmarks (audio, video, and videoaudio), the predictions for the constant workloads have an average error below 5% for all methods. The predictions for the media benchmarks have average errors around 10% due to the local fluctuations of power consumption when they run. Among the four prediction methods, LSF-based methods perform a slightly better than MS-based methods. This is because *least square fit* provides a better model for linear data than mean slope does. LSFPP is the best method among all. Overall, the performance of all the four methods is similar.

3.3 Results of Prediction For Variable Workloads

We follow the same procedure to make predictions for variable workloads. A typical voltage curve and transformed voltage curve for variable workload is shown in Figure 2(right).

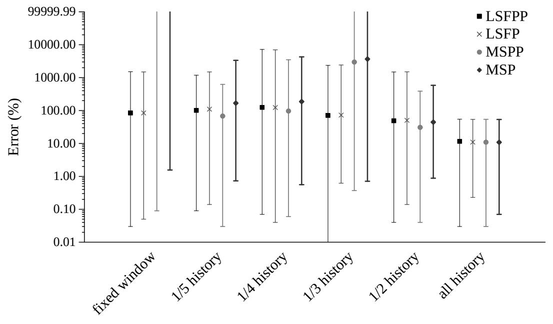


Fig. 6. Prediction performance for different history size. This figure shows the mean, max and minimum of "moving errors" of the 4 methods for real.load.5 benchmark, using different history size.

First, we explore how much history we need to make the best prediction. We tried the four methods using different history lengths: a history window with last 50 sample points, the last 1/5 , 1/4, 1/3, 1/2 of history, and all of the history. Figure 6 shows the mean (shown as the markers), maximum (shown as the top of the bar) and minimum (shown as the bottom of the bar) of prediction errors for real.load.5 benchmark. The y-axis shows the value of prediction errors (percentage) using a log scale. The data reveals that the prediction based on the entire history has both the smallest average error and error range for most methods. We found similar results for all other benchmarks. This tells us that the methods based on recent history are not able to make an accurate prediction of the future. In the results that follow, we only use the entire history to make a prediction.

Table 2. Prediction performance of ACPI, Smart Battery (the rolling average) and our methods based on entire history.

	real.load.5		IMemWC	
Methods	mean %	stdev %	mean %	stdev %
ACPI	60.28	27.70	42.96	71.30
rolling average	60.16	28.36	40.55	71.39
LSFPP	21.29	65.63	3.65	1.67
LSFP	20.76	66.19	4.01	1.22
MSPP	14.19	19.74	3.44	1.96
MSP	14.16	19.85	3.44	1.99

We next compare the performance of our methods against that of ACPI, a standard, commonly used, power management service that provides battery life estimation [2], and Smart Battery's rolling average algorithm [5]. The Familiar Linux reports detailed battery status through "/proc/asic/battery" file. We extract the ACPI-like battery life estimation directly from the file. We also simulate the Smart Battery's rolling average algorithm using the battery information from the file: at each prediction point, first calculate the average current within last 1 minute; then take the division of present remaining battery capacity and the average current as the present battery life estimation [5]. In Table 2, we show the mean (column 2 and 4) and standard deviation (column 3 and 5) of prediction errors using ACPI's battery life estimation, Smart Battery's rolling average algorithm and our four methods using the entire history. Columns 2 and 3 show data for "real.load.5", variable workload, benchmark; columns 4 and 5 show data for "IMemWC", constant workload, benchmark. Results for all other benchmarks are similar; each of our four methods significantly outperforms ACPI and Smart Battery's rolling average for both constant and variable workloads.

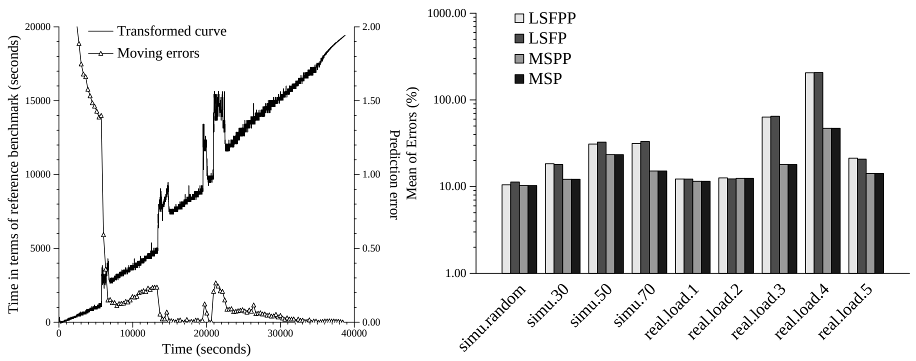
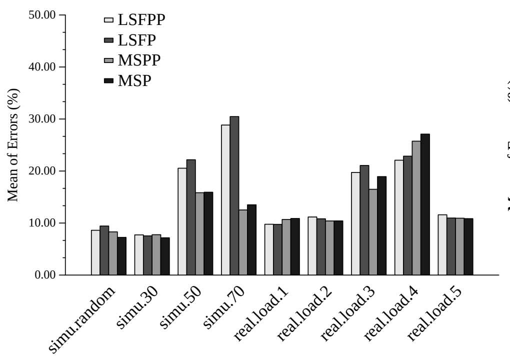


Fig. 7. Prediction performance for variable workloads. The left graph shows the transformed curve and moving errors for "real.load.4". The right one shows the average prediction errors for all variable loads and all methods. y-axis is in log scale.

Figure 7(right) shows the average prediction errors for all of the variable workload benchmarks. At the first glance, it seems that LSF-based methods perform very poorly for variable workloads. For example, LSFPP has an average error of 205.68% for real.load.4.

A detailed analysis shows that the prediction by LSFPP has some huge spikes in prediction error at the start time of the experiments when there is little history available. Figure 7(left) illustrates the relationship between the transformed voltage curve and the "moving errors". The axes have are similar to those in Figure 5(left). Before time 6000, there is no program running and the power consumption is very low creating a line segment with small slope at the beginning of the transformed voltage curve. The forecaster only knows about history and it makes a prediction that the battery will last for a much longer time than actual will. Immediately after some process starts to run, the curve goes up and the forecaster begins to realize the actual power consumption. As such, the prediction error also starts to drop. During the remaining time, the prediction error is much smaller.

To isolate this initial noise in the "moving errors", we calculate the mean for the last 95% of prediction errors (we call it the *trimmed prediction error*). That is, we discard the initial 5% (in terms of time) of the prediction errors to allow each forecaster to calibrate. Figure 8(left) shows the mean of the trimmed prediction errors for all variable workloads; the errors are smaller than without trimming. For example, the average prediction error of real.load.4 by LSFPP drops to 22.08%. These results indicate that for variable workloads, MSPP and MSP methods outperform LSFPP and LSFP.



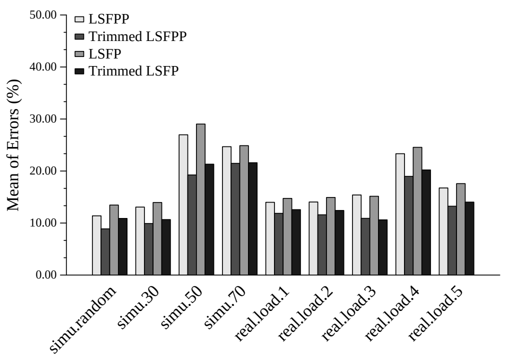


Fig. 8. Average prediction errors for variable workloads using a trim of 5%(left) and inverted curve(right).

Since the least square fit is not symmetric for the x and y axes (it tries to minimize the sum of distance squared along the y-axis), we also investigated the efficacy of our methods on the curves with switched x and y axes. We call the new curves "inverted transformed voltage curve". Since it is meaningless to average the inverse of a slope that represents the power consumption, we do not apply MSPP and MSP methods to the inverted curve. We show the results in Figure 8(right) as average prediction error using variable workloads for both trimmed and non-trimmed LSF methods using the inverted transformed voltage curve. The results indicate that the prediction performance using the inverted curve is more stable. In addition, it does not suffer from the early-stage spikes in the "moving errors"; as such, it is an alternative to trimming. In general, LSFPP outperforms LSFP.

In summary, we find that LSF-based methods are slightly better than MS-based methods for constant workloads. For variable workloads, MSPP performs best. In general, we believe MSPP is the best method for our online battery lifetime prediction. MSPP is cheaper to compute and even though it is slightly less accurate under constant conditions, it is less sensitive to variability in the measurement history.

4 Related Work

Battery life estimation, an integral component of power management systems, is provided in many mobile devices via both hardware and operating system support, such as that specified by Advanced Power Management (APM) [8] and more recently by Advanced Configuration and Power Interface (ACPI) [2]. In particular, ACPI, to which we compare our results, uses the division of remaining battery capacity and present rate of battery drain to estimate remaining battery life [2]. Smart battery is another similar technology that estimates battery life given a user-specified rate (e.g. present rate) or rolling average over a fixed interval (normally 1 minute) [5]. Such simple approaches to prediction consider only a very short discharge history and thus can be highly inaccurate, as we show in 3.3.

Another area of related research is model-based battery life estimation. Given the discharge profile of the entire lifetime of the battery, a well-designed battery model can give highly accurate battery life estimation. One such accurate model, described in [4], uses the low-level electrochemical phenomenon of battery discharge. This model is commonly used as a simulator to verify other battery models. More efficient simulation models include PSPICE-based models [6], discrete-time VHDL models [1] and a Markov chain model [11]. In [14] and [15] two efficient analytical models are proposed. The model described in [14], builds a relationship between current profile and battery lifetime. In [15], a fast prediction model is used to estimate the remaining battery capacity, which takes into account the recharge cycle aging and temperature. An approach for combining analytical models and statistical methods is proposed by [13].

These models are different from our method in that they require the complete discharge profile during a battery's life to make estimation. In other words, they make a "calculation" instead of a "prediction". Our method predicts battery life without knowledge of future workload. Model-based methods have other limitations to make them unsuitable for online, dynamic and adaptive battery life prediction, such as the need for large number of parameters and high computation cost (especially for simulation models).

In the work most related to ours [16], the authors estimate battery lifetime by exploiting the linear relationship between the system load and the drain time required to reach a specified voltage. They then apply statistical methods to make a prediction. This work differs from our work in two ways. First, the linearity they exploit is between the load and the time at which a certain voltage level is reached. As such, they must obtain a large number of load-versus-lifetime samples to generate an accurate curve fit. Using our method, only one reference voltage curve (generated off-line) is required. Second, this prior work studies only the lifetime estimation for constant workloads. It is not clear whether it can be applied to variable workloads. Our method naturally extends to variable workloads and we empirically evaluate it using both workload types.

5 Conclusions

We investigate battery lifetime prediction using a purely statistical method and only data that is readily available from the OS /proc file system. By using a statistical technique, our approach takes into account variations in workload, application profile, and battery charge rates, particularly those caused by the recovery of the battery during idle periods. We describe a coordinate transformation that converts a dynamic voltage curve into a form that enables more robust prediction of future behavior. We implement and empirically evaluate two variations of statistical methods on the transformed curve to make predictions. The experimental results show that we are able to achieve high prediction accuracy under both constant and variable workloads.

Our approach is simple, efficient, accurate, and flexible. In addition, it can be easily incorporated into current operating systems on popular hardware. As part of future work, we plan to investigate combinations of different prediction methods to further improve the accuracy of our method. Since there is not a universal method that is best for all cases, we are also seeking a way to leverage the power of different methods.

References

1. L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi. A discrete-time battery model for high-level power estimation. *In Proceedings of Design, Automation and Test in Europe*, 2000.
2. Compaq, Intel, Microsoft, Phoenix, and Toshiba. Advanced configuration and power interface specification, 2002.
3. Danionics lithium-ion polymer battery. `http://www.danionics.com/sw828.asp`.
4. M. Doyle, T. F. Fuller, and J. Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *Journal of Electrochem Society*, 141(1):1–9, January 1994.
5. Smart Battery System Implementers Forum. Smart battery data specification(v1.1), 1998.
6. S. Gold. A PSPICE macromodel for lithium-ion batteries. *In Proceedings of Annual Battery Conference on Applications and Advances*, pages 215–222, 1997.
7. M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. *IEEE 4th Annual Workshop on Workload Characterization*, December 2001. Austin, TX.
8. Intel and Microsoft. Advanced power management(apm) bios interface specification, 1996.
9. D. Linden and T. B. Reddy. *Handbook of Batteries(3rd edition)*. McGraw-Hill, 2002.
10. Linux for handheld devices. `http://www.handhelds.org`.
11. D. Panigrahi, C. Chiasserini, S. Dey, R. Rao, A. Raghunathan, and K. Lahiri. Battery life estimation of mobile embedded systems. *The 14th IEEE International Conference on VLSI Design*, 2001.
12. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (Second Edition)*. Cambridge University Press, 2002.
13. D. Rakhmatov and S. Vrudhula. Time-to-failure estimation for batteries in portable electronic systems. *In Proceedings of the International Symposium on Low Power Electronics and Design*, August 2001.
14. D. Rakhmatov, S. Vrudhula, and D. A. Wallach. Battery lifetime prediction for energy-aware computing. *In Proceedings of the International Symposium on Low Power Electronics and Design*, August 2002.
15. P. Rong and M. Pedram. Remaining battery capacity prediction for lithium-ion batteries. *Conference of Design Automation and Test in Europe*, March 2003.
16. K. C. Syracuse and W. Clark. A statistical approach to domain performance modeling for oxyhalide primary lithium batteries. *In Proceedings of Annual Battery Conference on Applications and Advances*, January 1997.