Chaotic Attractor Prediction for Server Run-time Energy Consumption

Adam Lewis, Jim Simon, and Nian-Feng Tzeng
Center for Advanced Computer Studies, University of Louisiana, Lafayette, Louisiana 70504
{awlewis,jks1870,tzeng}@cacs.louisiana.edu

Abstract

This paper proposes a chaotic time series model of server system-wide energy consumption to capture the dynamics present in observed sensor readings of underlying physical systems. Based on the chaotic model, we have developed a real-time predictor that estimates actual server energy consumption according to its overall thermal envelope. This chaotic time series regression model relates processor power, bus activity, and system ambient temperatures for real-time prediction of power consumption during job execution to enable run-time control of their thermal impacts. An experimental case study compares our Chaotic Attractor Predictor (CAP) against previous prediction models constructed according to other statistical methods. Our CAP is found to be accurate within an average error of 2% (or 7%) and the worst case error of 7% (or 20%) for the AMD Opteron processor (or for the Intel Nehalem processor), based on executing a set of SPEC CPU2006 benchmarks.

1 Introduction

Pro-active techniques for thermal management minimize server energy consumption by adjusting the allocation of thread groups in a server to available cores so as to consume the least amount of energy by all thread groups. High-performance computing workloads result in the server being fully utilized with little resources available for reactive scheduling. Effective adjustment of the resource allocation depends upon predicting changes in temperature with reasonable accuracy, given fluctuations in the system workload. The interactions among many subsystems within a server blade require the use of full-system models that consider the thermal load of all components in a system. Thus, it is critical to quantitatively understand the relationship among thermal load, system utilization, and power consumption at the system level so as to best optimize the scheduling of workloads.

Time series models operate by observing past outcomes of a physical phenomenon so as to anticipate future values of that phenomenon; such models are concerned more with the behavior of a phenomenon than with how or why the phenomenon changes. Many time series-based models of processor energy consumption have been proposed [1] [2] [3]; recent work extends such models into the thermal domain [4]. However, energy consumption, ambient temperature, and processor die temperatures in servers can be affected by more than just past values of those measurements made in isolation from each other. Taking these interactions into account requires the application of multivariate time series.

Multivariate time series are typically handled in three broad classes of mechanisms: auto-regressive, integrated, and moving average models [5]. Each of these classes assumes a linear relationship between the dependent variable(s) and previous data points. However, we show through our analysis of experimental measurements of key processor metrics (i.e., sensor readings) that the assumption of linearity does not apply in all cases. Furthermore, our analysis indicates that the non-linear behavior of these series is *chaotic in nature*. It thus leads to our development of a Chaotic Attractor Predictor (CAP).

In summary, the contributions of this work are three-fold. (1) We show, by analyzing measurements from two different server systems, that server energy consumption exhibits non-linear time series possessing chaotic behavior, (2) we construct a model of server energy consumption using a technique for analyzing chaotic time series of sensor measurements during job execution, (3) we use measurements of server energy consumption on two different server systems to compare the performance of our model against those based on linear regression and auto-regressive techniques.

2 Background and Prior Work

An analytical model of server energy consumption was built earlier [6] by modeling energy consumption as a function of the work done by the system in executing its computational tasks and of residual thermal energy given off by the system in doing that work. The resulting dynamic system consumes energy expressed in the time domain as follows:

$$\begin{aligned} E_{system} &= \frac{dP_{system}}{dt} \\ &= f(E_{proc}, E_{mem}, E_{em}, E_{board}, E_{hdd}) \end{aligned} \tag{1}$$

where each of the terms in the above equation is defined as: (1) E_{proc}: energy consumed in the processor due to computations, (2) E_{mem}: energy consumed in the DDR SDRAM chips, (3) E_{em}: energy taken by the electromechanical components in the system, (4) E_{board}: energy consumed by peripherals that support the operation of the board, and (5) E_{hdd}: energy consumed by the hard disk drive during the system's operation.

We can approximate an energy consumption solution for this system by considering (1) an initial energy state E_{system} at time $t = 0$ and (2) a set of physical predictors that approximate the values of E_{proc}, E_{mem}, E_{board}, and E_{hdd} at the next interval $t + \Delta t$. The result is a time series

$$e\_sys = \hat{f}(e\_proc_t, e\_mem_t, e\_em_t, e\_board_t, e\_hdd_t) \tag{2}$$

where each of quantities e corresponds to one or more physically observable predictors of the quantities in Eq. (1). The function $\hat{f}$ captures the method used to combine the physical observations into e_sys. We place two key requirements upon $\hat{f}$: (1) it must quickly compute estimates to be suitable for real-time prediction of energy and temperature changes, and (2) it must approximate the behavior of the original function f to an acceptable accuracy. Our problem now becomes what shall we use in our predictor to implement the $\hat{f}$ function?

A common approach for selecting predictors is to apply a linear auto-regressive (AR) combination of CPU utilization, disk-utilization, and hardware performance counters as estimators for the quantities in Eq. (2). Linear regressions model the relationship between one or more variables, such that the model depends linearly on unknown parameters to be estimated from a set of data. An excellent summary of models of this type can be found in [1], with recent representative examples described in [6], [2], and [3].

Other global auto-regressive techniques have been proposed for both energy and thermal modeling. For example, Coskun *et al.* [4] proposed the use of an Auto-Regressive Moving Average (ARMA) technique as a means to model changes in temperature as part of a thermally aware process scheduler. An ARMA predictor makes estimates of future values of a system by using past values. The ARMA model assumes that the process is a stationary stochastic process. In a stationary process, the probability distribution does not change with time. As a result, neither the mean nor the variance will change over time. ARMA predictors are not suited for data that exhibits sudden bursts of a large amplitude at irregular time epochs due to their underlying assumptions of normality [7]. This issue can be addressed by corrective mechanisms that adapt the predictor to the workload-related changes in a dynamic system. For example, Coskun *et al.* [4] addressed this issue by including a machine-learning based corrector that monitored workload changes and adjusted the parameters of their ARMA model.

Another approach to dealing with this problem is the use of an adaptive nonparametric regression scheme based upon partitioning the underlying time series. Multivariate Adaptive Regression Splines (MARS), introduced by Friedman [8], is an adaptive procedure that addresses non-linearity by fitting a weighted sum of basis functions to the data set, with the basis functions in three possible forms: (1) a constant value of 1 that represents the intercept term of the regression, (2) a hinge function of the form $max(0, x - c)$ or $max(0, c - x)$ that represents knots in the regression, or (3) a product of two or more hinge functions that model interactions between two or more variables. The selection of weights and hinge constants is performed by a two-pass algorithm, with its forward pass adding basis functions to the regression in an attempt to reduce the mean square error while attempting to improve the model in its backwards pass by removing terms based upon cross validation.

3 Linear Regression: AR and MARS

A set of experiments was carried out to evaluate the performance of power models built using AR and MARS techniques to approximate a solution for dynamic systems following Eq. (1). Observable predictors of system activity were chosen for each of the quantities in Eq. (2) under two test systems: (1) an Oracle/Sun x2200 (AMD Opteron) server and (2) a Dell PowerEdge R610 (Intel Xeon X5300 Nehalem) server. AR and MARS models for Eq. (2) were created from data collected by executing a set of high-performance computing benchmarks from the SPEC CPU2006 benchmark suite: bzip2, cactusadm, gromac, leslie3d, omnetpp, and perlbench [9]. These models were then used as a predictive tool for another set of CPU2006 benchmarks (astar, gobmk, calculix, and zeusmp) as a means of evaluating the predictive performance of each model.

The results under these linear regression techniques (AR and MARS) are summarized in Tables 1 and 2. All three techniques predict well over the long term, with an average error ranging between 1.7% and 3.1% depending upon method and benchmark. However, they suffer from poor performance in the short term, with maximum errors ranging from 7.9% to 9.3% for the AMD Opteron server and in the range of 15% and 44% for the Intel Nehalem server. An analysis of the power traces for the different benchmarks reveals hints of this behavior. Consider the power trace shown in Fig. 1 for the SPEC CPU2006 zeusmp benchmark executed on an AMD Opteron server. We saw indications of (1) periodic behavior throughout the length of the run and (2) large swings in the power draw though the course of the benchmark run. Similar behavior was observed for other benchmarks on both server systems. Thus, it is reasonably conjectured that non-linear dynamics exist in the two test systems. Furthermore, the behavior of the power draw is such that the linear regression-based predictors will occasionally mis-predict by large amounts (up to 44%, as indicated in Table 2). The physical behavior of the system uncovers that such large swings in power draw cannot be completely attributed to noise and, as result, we must account for them within our model.

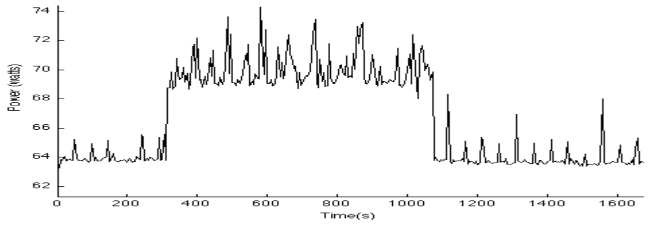

Fig. 1: Power trace for zeusmp on AMD Opteron.

4 Chaotic Behavior

We performed an analysis on the data collected from our test systems to determine if the behavior of our time series can be attributed to a certain form of chaotic behavior. A chaotic process is one which is highly sensitive to a set of initial conditions. Small differences in those initial conditions yield widely diverging outcomes in such chaotic systems. In order to determine whether a process is chaotic, we must be able to show that it demonstrates high sensitivity to initial conditions, topological mixing, and an indication that its periodic orbits are dense [10]. After analyzing our experimental data, we believe that the power consumption of a server demonstrates chaotic behavior, as detailed next.

In order to evaluate a server's sensitivity to initial conditions, we consider the Lyapunov exponents of the time series data observed while running those benchmarks described in the previous section. The Lyapunov exponent quantifies the sensitivity of a system such that a positive Lyapunov exponent indicates that the system is chaotic [10]. The average

Table 1: Model errors for AR, MARS, and CAP on AMD Opteron server

	AR			**MARS**			**CAP** (with $n = 5$, $p = 200$, $r = 16$)		
	Avg	**Max**	**RMSE**	**Avg**	**Max**	**RMSE**	**Avg**	**Max**	**RMSE**
Benchmark	**Err %**	**Err %**		**Err %**	**Err %**		**Err %**	**Err %**	
astar	3.1%	8.9%	2.26	2.5%	9.3%	2.12	0.9%	5.5%	0.72
games	2.2%	9.3%	2.06	3.0%	9.7%	2.44	1.0%	6.8%	2.06
gobmk	1.7%	9.0%	2.30	3.0%	9.1%	2.36	1.6%	5.9%	2.30
zeusmp	2.8%	8.1%	2.14	2.8%	7.9%	2.34	1.0%	5.6%	2.14

Table 2: Model errors for AR, MARS, and CAP on Intel Nehalem server

	AR			**MARS**			**CAP** (with $n = 5$, $p = 200$, $r = 18$)		
	Avg	**Max**	**RMSE**	**Avg**	**Max**	**RMSE**	**Avg**	**Max**	**RMSE**
Benchmark	**Err %**	**Err %**		**Err %**	**Err %**		**Err %**	**Err %**	
astar	5.9%	28.5%	4.94	5.4%	28.0%	4.97	1.1%	20.8%	1.83
games	5.6%	44.3%	5.54	4.7%	33.0%	4.58	1.0%	14.8%	1.54
gobmk	5.3%	27.8%	4.83	4.1%	27.9%	4.73	1.0%	21.5%	2.13
zeusmp	7.7%	31.8%	7.24	11.6%	32.2%	8.91	3.3%	20.6%	3.31

Lyapunov exponent can be calculated using:

$$\lambda = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} ln|f'(X_n)|.$$

We found a positive Lyapunov exponent when performing this calculation on our data set ranging from 0.01 to 0.28 (or 0.03 to 0.35) on the AMD (or Intel) test server, as listed in Table 3, where each pair indicates the parameter value of the AMD server followed by that of the Intel server. Therefore, our data has met the first and the most significant criterion to qualify as a chaotic process.

The second indication of the chaotic behavior of the time series in Eq. (2) is an estimate of the Hurst parameter H for the data sets collected in each benchmark. A real number in the range of $(0, 1)$, the Hurst parameter is in the exponents of the covariance equation for Fractional Brown motion (fBm) [10]. If the value of the Hurst parameter is greater than 0.5, an increment in the random process is positively correlated and long range dependence exists in the case of time series. In a chaotic system, a value of H approaching 1.0 indicates the presence of self-similarity in the system. As demonstrated in Table 3, the time series data collected in our experiments all have values of H close to 1.0, ranging from 0.93 to 0.98 (or 0.93 to 0.97) on the AMD (or Intel) test server.

Table 3: Indications of chaotic behavior in power time series (AMD, Intel)

Benchmark	**Hurst Parameter** (H)	**Average Lyapunov Exponent**
bzip2	(0.96, 0.93)	(0.28, 0.35)
cactusadm	(0.95, 0.97)	(0.01, 0.04)
gromac	(0.94, 0.95)	(0.02, 0.03)
leslie3d	(0.93, 0.94)	(0.05, 0.11)
omnetpp	(0.96, 0.97)	(0.05, 0.06)
perlbench	(0.98, 0.95)	(0.06, 0.04)

5 Predicting from Chaos

From a predictive standpoint, the unpredictable deterministic behavior of chaotic time series means that it is difficult to build a predictor that takes a global parametric view of the data in the series. However, it is possible to generate a highly accurate short-term prediction by reconstructing the attractor in the phase space of the time series and applying a certain form of least square prediction to the resulting vector space [11].

5.1 Chaotic Predictor: CAP

Given the time series introduced in Eq. (2), we define X_t to be the value of e_{system} at time t, and r to be the total number of sensors and OS measures to provide metric readings. According to Taken's Delay Embedding Theorem [10], there exists a function $\hat{f}(X_t)$ whose behavior in the phase space reflects the behavior of the attractors in the original time series. Our problem now becomes finding a means to approximate $\hat{f}$.

We introduce the concept of a Chaotic Attractor Predictor (CAP) that defines $\hat{f}$ in terms of linear least squares regression of a multivariate local polynomial of degree r. Multivariate local linear regression is a common non-parametric technique for time series approximations. With CAP, we extend this concept to predict the behavior of a chaotic time series by following the approximation method proposed earlier [11]. CAP is a predictor that exhibits the computational advantages of polynomial time complexity while capturing the dynamics of test systems.

Let x be an observation (involving r metric readings) at some future time $t + \Delta t$ and X_u be a prior observation (involving r metric readings) at time u for $u = t-1, \ldots, t-p$. For CAP, we use the standard d-variate normal density function, with $\|x\|$ being the norm of vector x:

$$K(x) = (2\pi)^{-\frac{m}{2}} exp(-\|x\|^2/2)$$

as a tool to localize the neighborhood in which we define our polynomial. We do this through *kernel weighting* with a defined bandwidth matrix H for localization by assigning a weight of $K_H(x) = |H^{-1}|K(H^{-1}x)$. This can be simplified by taking the bandwidth matrix $H = hI_r$, with h being a scalar value and I_r being the identity matrix of order r.

A local constant approximation for $\hat{f}$ is defined next in terms of a locally weighted average [5] over the next n observations, based on the prior p observations of $X_{t-1}, \ldots, X_{t-p}$ (each with r metric readings):

$$\hat{f}(x) = \frac{\sum_{t=p+1}^{n+p} O_p * K_H(X_{t-1} - x)}{\sum_{t=p+1}^{n+p} K_H(X_{t-1} - x)}$$

with $O_p = (X_{t-1}, \ldots, X_{t-p})^T$.

The process can be improved by defining a local linear approximation via applying a truncated Taylor series expansion of $\hat{f}$:

$$\hat{f}(X) = \hat{f}(x) + \hat{f}^{'}(x)^T (X - x).$$

The coefficients of the polynomial $\hat{f}$ are then determined by minimizing

$$\sum_{t=p+1}^{n+p} \left[X_t - a - b^T (X_{t-1} - x)\right]^2 * K_H(X_{t-1} - x). \quad (3)$$

with respect to a and b, which are estimators to $\hat{f}(x)$ and $\hat{f}'(x)$, respectively. The predictor generated by solving Eq. (3) can be explicitly written, according to [5], as

$$\hat{f}(x) = \frac{1}{n} \sum_{t=p+1}^{n+p} (s_2 - s_1 * (x - X_{t-1}))^2 * K_H((x - X_{t-1})/h) \quad (4)$$

with $s_i = \frac{1}{n} \sum_{t=p+1}^{n+p} (x - X_{t-1})^i * K_H((x - X_{t-1})/h)$ for i = 1 or 2.

There are three steps involved in the process of establishing a CAP predictor: (1) creating a training set for the process, (2) using the observations from the training set to find the appropriate delay embedding using Takens Theorem and then apply the nearest neighbors algorithm in the embedded set to identify the attractors, and (3) solving the resulting linear least squares problem that arises from applying Eq. (3) to the attractors using the function expressed by Eq. (4). The time complexity of creating a predictor is governed by the third step in the process. The task of reconstructing the state space by delay embedding is linear in time as one must make up to d passes through the observations, under the embedding dimension of d. Thus, the time required is $O(dn)$, where n is the number of next observations. Then, it becomes a matter of applying a naive form of k-th nearest neighbors algorithm to identify the points in the attractors. This step involves finding the squared distance of all the points in the nearest analogs in the Takens set and then sorting the result to determine the d-nearest neighbors. This step takes $O(n \log n + n)$. We avoid the cost of computing the linear least squares solution in the third step by using the explicit expression given in Eq. (4). The time complexity of computing this expression can be shown to be $O(n * n)$, with $O(n)$ due to computing s_i, for i = 1 or 2. As a result, the time complexity for establishing a CAP predictor equals $O(n^2)$. It should be noted that the construction of a CAP predictor is done only once for a given server, irrespective of applications executed on the server.

5.2 Evaluation and Results

We evaluated the predictive performance of CAP versus linear regression techniques by applying a local linear CAP to the same collected data used in Section 3. The training set (bzip2, cactusadm, gromac, leslie3d, omnetpp, perlbench [9]) for the model was created by taking the geometric mean of all the time series involved in evaluating linear regression techniques. Next, the process described in Section 5.1 was employed to determine the attractors of the resulting time series, followed by generating an approximating polynomial to fit the attractor. The resulting predictor was applied to those benchmarks (i.e., astar, gobmk, calculix, and zeusmp) used to evaluate linear regression techniques. The results of this experiment are summarized in Tables 1 and 2 under the column "CAP".

CAP predicts power changes in the long term with an average error in the range from 1.0% to 1.6% for the AMD Opteron server and 1.4% to 4.2% for the Intel server. Maximum errors are far better under CAP than under its linear regression counterparts, ranging from 5.5% to 6.8% only for the AMD Opteron server, in contrast to as large as 9.7% under MARS. Better prediction was observed similarly for the Intel server, with the maximum error dropping to 21.5% from 33.0% under MARS, accompanied by corresponding improvement in the Root Mean Square Error (RMSE) for all benchmarks. As expected, CAP provides a better fit overall and at the start as well as the end of the series. This behavior, as opposed to a regression spline technique like MARS, results from better prediction of piecewise local polynomials

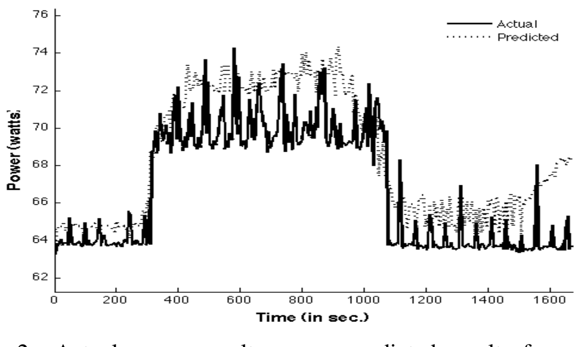

Fig. 2: Actual power results versus predicted results for zeusmp benchmark under MARS for AMD Opteron.

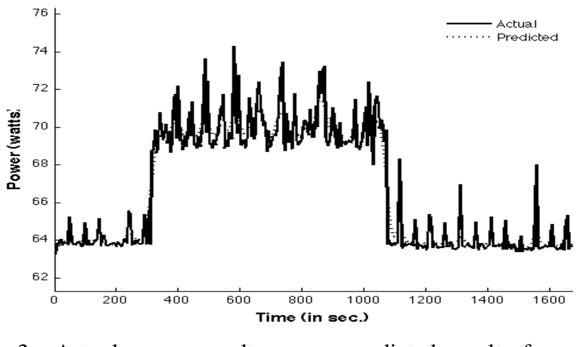

Fig. 3: Actual power results versus predicted results for zeusmp benchmark under CAP for AMD Opteron.

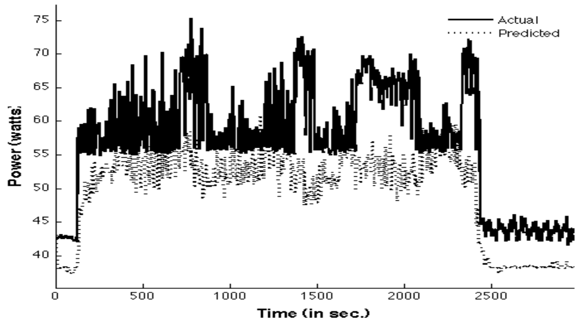

Fig. 4: Actual power results versus predicted results for zeusmp benchmark under MARS for Intel Nehalem.

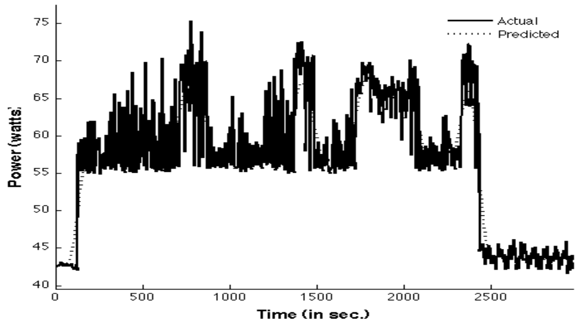

Fig. 5: Actual power results versus predicted results for zeusmp benchmark under CAP for Intel Nehalem.

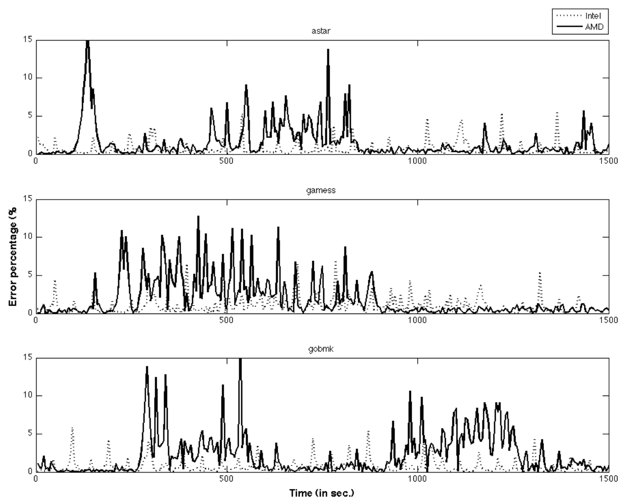

Fig. 6: CAP error rates versus time for three other benchmarks.

versus the regression spline being global in nature. A detailed comparison between MARS and CAP predictors for the AMD server can be seen in Fig. 2 and Fig. 3, where $n = 5$, $p = 200$ and $r = 16$. CAP exhibits far smaller difference between actual and predicted power consumption amounts under the CPU2006 zeusmp benchmark. Likewise, a contrast between MARS and CAP predictors for the Intel server under the zeusmp benchmark is demonstrated in Fig. 4 and Fig. 5, where $n = 5$, $p = 200$ and $r = 18$. Fig. 6 illustrates the error rates for the other three evaluated benchmarks over their execution intervals. Those benchmarks exhibit similar behavior to what is seen under the zeusmp benchmark, shown in Figs. 2-5.

6 Conclusion

In this paper, we have shown that models constructed from global auto-regressive methods, such as AR, ARMA, and MARS, demonstrate behavior that makes them problematic for predicting server energy consumption. The proposed CAP overcomes the limitations of the previous linear regression-based methods by addressing the non-linear aspects of the time series data while capturing the underlying chaotic behavior of the dynamic physical system. CAP involves $O(n \log n)$ time complexity and requires no additional hardware beyond those made available in recent processors, nor any tool outside those provided by operating systems. As a result, CAP can support high-performance and real-time application workloads, readily applicable for run-time energy consumption prediction.

Acknowledgement

This work was supported in part by the U.S. Department of Energy (DOE) under Award Number DE-FG02-04ER46136 and by the Board of Regents, State of Louisiana, under Contract Number DOE/LEQSF(2004-07)-ULL.

References

[1] S. Rivoire, “Models and Metrics for Energy-efficient Computer Systems,” Ph.D. dissertation, Stanford Univ., 2008.

[2] A. Bhattacharjee and M. Martonosi, “Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors,” *Proc. of the 36th Int’l. Symp. on Computer Architecture*, pp. 290–301, 2009.

[3] J. Reich, M. Goraczko, A. Kansal, J. Padhye, and N. Computing, “Sleepless in Seattle no longer,” *Proc. of the 2010 USENIX Annual Tech. Conf.*, 2010.

[4] A. K. Coskun, T. S. Rosing, and K. C. Gross, “Proactive temperature balancing for low cost thermal management in mpsocs,” *Proc. of the 2008 IEEE/ACM Int’l. Conf. on CAD*, pp. 250–257, 2008.

[5] G.E.P. Box and G.M. Jenkins and G.C. Reinsel, *Time Series Analysis, Forecasting and Control*. New York, NY, USA: Prentice Hall, 1994.

[6] A. Lewis, S. Ghosh, and N.-F. Tzeng, “Run-time energy consumption estimation based on workload in server systems,” *Proc. of the 2008 Workshop on Power Aware Computing and Systems (Hotpower’08)*, 2008.

[7] H. Tong, *Non-linear Time Series: A Dynamical System Approach*. New York, NY, USA: Oxford Univ. Press, 1993.

[8] J. Friedman, “Multivariate adaptive regression splines,” *Annals of Statistics*, vol. 19, pp. 1–142, 1991.

[9] J. L. Henning, “SPEC CPU2006 benchmark descriptions,” *Computer Architecture News*, vol. 34, no. 4, Sept 2006.

[10] J. Sprott, *Chaos and Time-Series Analysis*. New York, NY, USA: Oxford Univ. Press, 2003.

[11] Kenichi Itoh, “A method for predicting chaotic time-series with outliers,” *Electronics and Communications in Japan, Part 3: Fundamental Electronic Science*, vol. 78, no. 5, pp. 1529–1536, Apr. 1995.