Characterizing Application Workloads on CPU Utilization for Utility Computing

Bruno Abrahao, Alex Zhang
Intelligent Enterprise Technologies Laboratory
HP Laboratories Palo Alto
HPL-2004-157
September 15, 2004*

capacity planning, principal component analysis, workload model, trace characterization and generation

We analyze CPU utilization traces of multiple applications running on a shared set of processors in a utility computing environment and apply PCA (Principal Component Analysis) technique to characterize each application's workload. We show that, in our dataset, the 12 applications under examination can be characterized by just three features, namely, periodic, noisy, and spiky. We then use these principal components for classifying applications, detrending the CPU usage behavior, and generating synthetic traces with amplification or suppression of the desired features. The workload characteristics that we derive using the PCA approach help application owners to better understand the behaviors of their applications, and also enable the system operator to better plan for capacity usage.

* Internal Accession Date Only
Approved for External Publication

Characterizing Application Workloads on CPU Utilization for Utility Computing

Bruno Abrahao, Alex Zhang
HP Labs, Palo Alto, California 94304, U.S.A.
August 31, 2004

***Abstract*— We analyze CPU utilization traces of multiple applications running on a shared set of processors in a utility computing environment and apply PCA (Principal Component Analysis) technique to characterize each application's workload. We show that, in our dataset, the 12 applications under examination can be characterized by just three features, namely, periodic, noisy, and spiky. We then use these principal components for classifying applications, detrending the CPU usage behavior, and generating synthetic traces with amplification or suppression of the desired features. The workload characteristics that we derive using the PCA approach help application owners to better understand the behaviors of their applications, and also enable the system operator to better plan for capacity usage.**

I. Introduction

Utility Computing [1] has recently become a wide-spread paradigm for data center operators to adapt to the changing business environment. Utility Computing offers two advantages: (1) lowering customer costs related to hardware, software license, deployment and maintenance of IT; and (2) allowing customers to adapt quickly to business changes and simplify IT environments. With utility computing, customers are able to obtain computing services as needed, deriving benefits from the optimal use of IT resources.

HP's SASU (Shared Application Server Utility) program [2] aims to create an adaptable, high performance, shared application infrastructure using the Utility Computing paradigm. Managed under HP Global Operations and IT, SASU uses HP's own technology, hardware and operating system (HP-UX with HP OpenView), providing adaptive IT infra-structure for hosting J2EE applications that are owned by internal HP business divisions. Currently in August 2004, SASU has twelve business applications running under BEA WebLogic on two HP RP8400 production servers. Within a few months, more than 20 applications will be deployed on SASU; new server hardware has already been procured and being integrated into SASU.

Several issues that arise in this kind of environment are (1) How to size the IT infra-structure; (2) How to allocate resources (CPU, Memory, network, etc) to different applications that share these resources (see [3]); and (3) How much and when to add (or remove) physical hardware when computing demand increases (changes) (see [4]). These decisions must be made to satisfy pre-specified service level objectives (SLOs) and at the same time to balance the cost of providing the service (such as by maintaining CPU utilization to a sufficient level). Central to all these issues is understanding and prediction of the resource usage characteristics (or behaviors) of the applications. For example, an application that exhibits "spiky" CPU usage behavior might require a higher share of CPU allocation to ensure its SLO to be met.

To address the first issue of capacity planning (or sizing), we propose a three stage approach [5], in which the planning horizon is divided into Pre-Installation, Test and Production phases.

The three stages differ from each other by the data availability and the desired precision of planning. The Pre-Installation capacity plan is driven by mathematical scaling and extrapolation models based on published (known) benchmark data. A tool called Pre-Installation Capacity Calculator [5] was designed to execute the models and to automate the analysis.

At the test (or integration) stage and the production stage in the application deployment lifecycle, the application has been installed and the computing utility is instrumented to collect usage data of the application (the trace) under a synthetic workload (at the test stage) or a real workload (at the production stage). The measurement data on workload (transactions) and performance (CPU usage) enables us to refine the workload and the performance models. We may derive the application's characteristics through analytical extraction discussed in this paper, thus enabling the application owner and the system operator to gain a better understanding of the application's behavior in the shared-resource computing environment. As a result, resource usage plans can be made more precise; capacity planning implications due to future business growth or business change can be analyzed.

In this study, we use PCA (Principal Component Analysis), a feature selection technique, and apply some analysis steps, called structural analysis [6], to an application dataset extracted from a SASU production server. By determining whether the whole system has low intrinsic dimensionality, it is possible to create a workload model that is described only by a small set of features such as periodic, noisy and spiky.

We then use the extracted features for several purposes: classifying applications based on their behavioral features; detrending application traces; and generating synthetic workload with the option to amplify or suppress any of the features. Potential uses other than those described above are also possible.

This paper is organized as follows. In section II, we outline the need for an automatic means for extracting application characteristics from multiple traces in a shared and interdependent system. Section III describes the data (traces) collected from the SASU environment; Section IV gives the technical

background on the Principal Component Analysis (PCA) technique. We describe how we apply PCA to the SASU data in Section V and how we use the extracted features in Section VI. Finally, we summarize and conclude in Section VII.

II. Workload Characterization

It is known that the performance of a system depends heavily on the characteristics of its load. Starting from the Test Phase, it is possible to rely on some data (trace), collected from the system's log files, to create a model that describes, or approximates, the actual application's workload behavior. From these models, one can predict the impact of the load imposed by each application in the system, when the system goes to the Production Phase. Much of the work done in this direction focus on single application's load characterization, which considers each application at a time, creating a model from observation (i.e. a stochastic model) and extracting the parameters needed from the application trace.

In a single system there may be hundreds or thousands of applications running simultaneously. If one had to analyze each application at a time, it would be a rather exhaustive task. Furthermore, the whole system's performance is affected not only by the behavior of each single application, but also by the resulting execution of several different applications combined together. This means that the system's overall performance is given by the superposition of several timeseries that, in some instances, are not independent. A wide range of important problems require the whole system analysis, including resource assignment [3], [7], [8], queueing networks [9] and system's behavior analysis [10]. Since a single application analysis is itself a complex task, modeling the whole system behavior is even more difficult. The reason is that, considering each application as a timeseries, they form together a high-dimensional structure.

However, one can suppose that some applications share common behaviors as a function of time. For example, several applications could share the same periodic behavior due to steady peaks of utilization during the business hours and low utilization during the lunch hour and other non-business hours in the evenings and on weekends. On the other hand, some applications could present simultaneous short bursts (or spikes) of high demand, which are called flash crowds, often triggered by a special event. These observations lead us to believe that the high-dimensional structure of application timeseries, that appears to be complex, could be governed by a small set of features (i.e. correlated periodicity, simultaneous demand spikes) and, therefore, be represented approximately by a lower-dimensional representation. This minimum number of features, needed to closely approximate a high-dimensional structure, is called the intrinsic dimensionality of the data.

PCA (Principal Component Analysis) is a useful technique for feature selection. With PCA, one can process a large amount of data quickly to determine whether the whole system has low intrinsic dimensionality, and to identify the prominent features exhibited in the data, thus making it possible, by exploiting the common temporal patterns shared by applications, to create a workload model that is described only by a small set of features.

Several kinds of analysis could benefit from this study. For instance, all sort of algorithms could be carried out using a small input, that is, a small number of features instead of all original traces. We can imagine that, although SASU is currently under low utilization and has a relatively low number of applications running, the PCA analysis that we study in this paper will become more effective as the number of applications hosted on SASU grows.

III. Data Description

SASU's application servers are grouped under Functional Test, Integration Test, and Production groups. The multiple servers within each group (except the Functional Test group) are clustered for load balance purposes. In this fashion, there is one server dedicated to functional test, two for integration test and two for actual production. The servers run HP-UX operating system and OVPA (OpenView Performance Agent) to collect performance data.

The data considered in this study was collected from one of the two production servers, during the two-week period from from 08/02/2004 to 08/16/2004. The reason for collecting only two weeks worth of data is that SASU's CPU utilization had been increasing over time and, prior to this period, some applications did not have sufficient utilization to be analyzed. There were twelve applications running on this production server (HTX694).

OVPA (OpenView Performance Agent) aggregates (averages) the CPU utilization at every five minutes and writes to a log file which is subsequently extracted and stored in a central repository. The dataset lists CPU utilization percentages by each application (`APP_CPU_TOTAL_UTIL` OVPA metric) in 5-minute intervals during the two-week collection period and it is summarized in Table I. This table shows the maximum, the average and the standard deviation of the percentage of CPU utilization during the two weeks. The number in the first column is used, by convention in this paper, as a further reference to the applications. We have excluded the `PRM_SYS` application (a group of system management processes) and the `OTHER` group (undefined applications). From this table, it is possible to conclude that the server is still under light utilization.

From the data collected, we generate a CPU utilization measurement $m \times p$ matrix X, where the number of rows m is the number of time intervals (number of 300-second intervals within the two-week period) and the number of columns p is the number of applications in the system ($p = 12$). Note that $m >> p$.

Each column i of the matrix denotes an application timeseries, referenced by vector X_i, and each row represents an instance of all the applications at time t. The matrix X is used as the input for PCA. Figure 1 shows some examples of application load traces (applications 2, 6 and 7), which correspond to columns of the X matrix. The plots show the percentage of CPU utilization as a function of time. In some

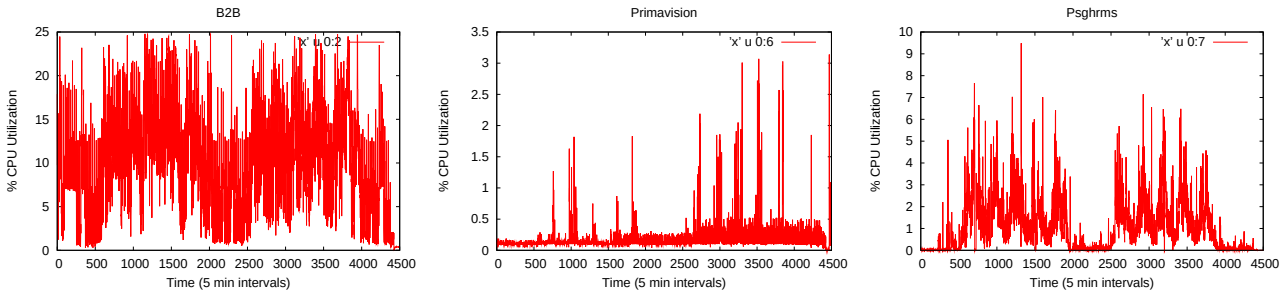

Fig. 1. Examples of Application Traces

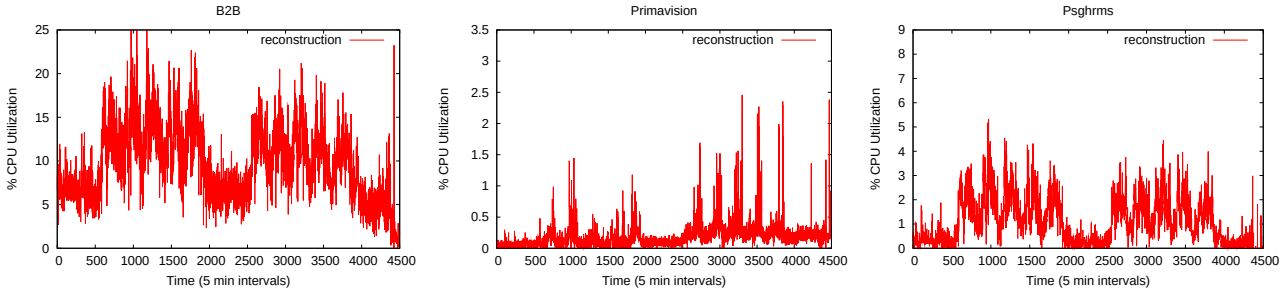

Fig. 2. Examples of Application Reconstructed Traces Using Three Features

TABLE I
APPLICATION LOAD SUMMARY: CPU UTILIZATION PERCENTAGES BY APPLICATION

Num.	**Application**	**Max**	**Avg**	**Std.Dev.**
01	ARC	14.31	0.08	0.04
02	B2B	24.79	9.59	5.64
03	Contivo	0.34	0.02	0.01
04	Esgui	0.38	0.22	0.03
05	Parallax	0.11	0.04	0.01
06	Primavision	3.14	0.19	0.25
07	Psghrms	9.49	1.08	1.21
08	Pshd	5.12	0.54	0.63
09	Psportal	18.56	0.97	1.65
10	Rdma	0.09	0.00	0.01
11	Rocket	0.60	0.01	0.02
12	Wwtbl	0.37	0.01	0.01

plots, it is possible to visually identify temporal patterns during the course of the two weeks.

IV. PRINCIPAL COMPONENT ANALYSIS

This section provides the basic concepts behind PCA. A detailed description of PCA can be find in [11]. The idea behind PCA is to place the data into a new set of coordinates, or components. This new hyper-space has the property that the first (principal) component points to the direction of the maximum data variation in magnitude, in terms of the Euclidean norm. After finding the first component, the second orthogonal component can be found by removing the information captured by the first component, and capturing the maximum variation of the residual; and so on. This idea can be illustrated graphically using a two-dimensional space. In Figure 3, there is an elliptical cloud being cut by a principal component along the direction that captures the maximum variation in magnitude from the data.

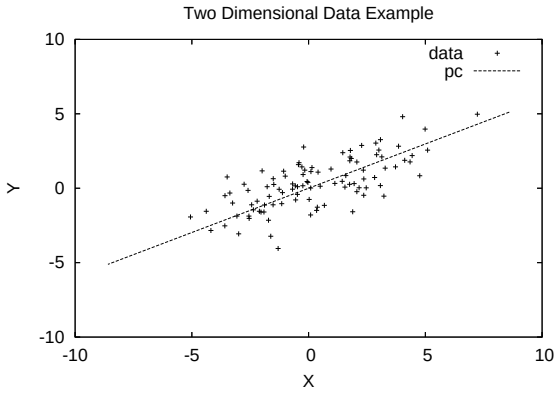

Fig. 3. Example of Finding the Principal Component

In practice, this is equivalent to finding the eigen-vectors of $X^T X$ [11].

$$X^T X v_i = \lambda_i v_i \quad i = 1, .., p \tag{1}$$

The eigen-values, λ_i, provide the magnitude of the variation along each component i, so, by convention, we sort these λ values in decreasing order. Furthermore, because $X^T X$ is symmetric positive definite, its eigen-values λ_i are nonnegative. The eigen-vectors, v_i, are vectors of size p and they form the transformation matrix V. To maintain consistency, we rearrange the vectors v_i according to its corresponding λ_i values in a decreasing order.

In the new mapped space, the contribution of principal component i is given by $u_i = X v_i$. The u_i vector of size

m is actually a feature shared by all applications along the principal component i. It can be normalized to unit length by dividing by the singular value $\sigma_i = \sqrt{\lambda_i}$. Since the V matrix is sorted in a way in which the first component represents the maximum variation in energy, u_1 is the most dominant feature, u_2 the second dominant and so on.

Conversely, the original data can be reconstructed from features as:

$$X = U(V^{-1}) \tag{2}$$

where U is the $m \times p$ feature matrix and V is a $p \times p$ matrix containing the eigen-vectors.

V. Data Manipulation

In order to examine the low intrinsic dimensionality of the SASU's application dataset, we apply PCA to the dataset. The methodology we are using here was first presented in [6] in analyzing network traffic behavior.

The singular values account for the data variability along their correspondent principal components. Extracting the singular values using PCA, we can determine how many features actually bring information to the dataset. In other words, there are some components that account for low variability and therefore have low degrees of representativeness in the dataset.

There are two possible causes for the low intrinsic dimensionality. First, if the magnitude of the loads differs greatly among the applications, the ones that have the greatest mean values will dominate the energy (or variance) in the dataset. Second, the other cause of low intrinsic dimensionality can be attributed to the common patterns among the timeseries, making the dimensions correlated and, therefore, redundant. Since we are interested in the latter, that is, the common behavior shared by the applications, in order to avoid the first effect, we normalize the timeseries data to zero mean and unit variance:

$$X_{i,t} = \frac{X_{i,t} - \bar{X}_{i,*}}{\sigma_i}. \tag{3}$$

The dimensionality analysis can be carried out using a scree plot. In this plot, the set of the sorted singular values is presented with their correspondent magnitudes. Figure 4 presents the scree plot for the SASU's application dataset. From this plot, we can see that the first two attributes are responsible for the largest changes in variation in the dataset. While features 1 and 2 account for a change of more than two points in magnitude, the other ten together account only for one.

If we consider only the first three attributes to reconstruct the original trace, it is possible that the reconstruction still preserve the main characteristics of the original, even though this process will incur some loss (distortion).

Some problems require that the data be strongly correlated in order to apply PCA, so the loss will be minimized. However, there are other cases, in which the precision in the absolute value of each point in the reconstructed trace is not a strict goal. For instance, in this particular study, we are mostly interested in capturing application behavior (*shapes* of the plots) rather than in the precise value of the CPU utilization

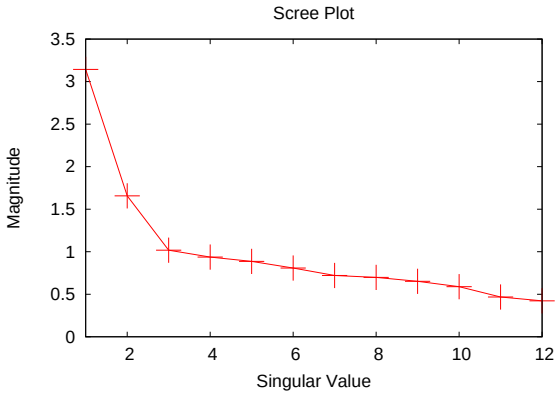

Fig. 4. Scree Plot for Normalized Data

at a particular time. Therefore, if the reconstruction using a handful number of attributes preserves, or mimics, the behavior of the original applications, it could be rather useful.

In Figure 2 we show the reconstructed traces of the three applications, presented earlier in Figure 1, using only three out of the twelve features. From this picture, we can see that the noticeable temporal patterns of the applications are preserved. The number of features can be chosen according to the error tolerance level allowed.

In order to evaluate the degree of correlation between the original and the reconstructed traces, we plot in Figure 5 the correlation coefficient as a function of the number of features used in the reconstruction, for the three applications shown in Figure 1 (original) and Figure 2 (reconstructed). We can see that the first three features significantly affect the correlation level for the three applications. Also, as we add more than the first three features, the correlation increases slowly in all the traces.

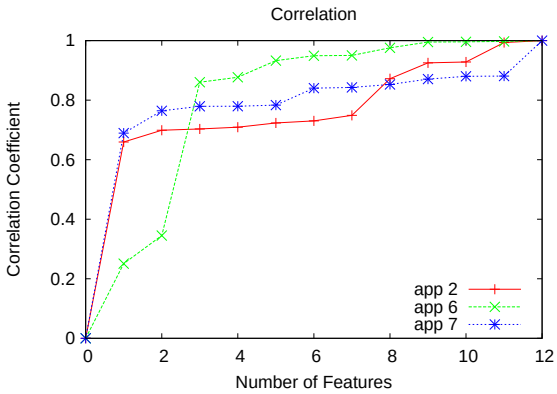

Fig. 5. Correlation Coefficient Between the Original to the Reconstructed Traces by Varying the Number of Features Considered

Since three features were able to capture the temporal characteristics of the original traces remarkably well, in the rest of this paper, we are going to consider three as the intrinsic dimensionality of the data.

It is also possible to examine the features and try to attribute meaning to them. The first three features, which are the most

dominant, are displayed at Figure 6. Visually, we can see a clear distinction among the features. Feature number one presents a strong periodic trend, in which each period is a complete business day (288 intervals of five minutes). There are also some longer valleys due to the low utilization during the weekends. Feature number two seems like a Gaussian noise that intensifies in the second week due to the SASU's increasing utilization over time. Finally, feature number three presents short lived spikes in each one of the business days, which also intensify in the second week.

The features are used as the basic units and together form a CPU utilization workload model. Some useful analysis can be derived from the model, as presented in the next section.

VI. Using the Extracted Features

This section shows how the model can be used to describe the system's behavior, classifying the applications and analyzing its components, and generating synthetic traces for test and simulation purposes.

A. Classifying Applications

One important issue in capacity planning is the ability to classify a large number of applications into relatively few groups (or types). The notion of classes gives the planner the possibility to distribute the load incurred by different types of applications over separate servers, to size the resource consumption for each group, and to decide which applications can be put together to share the same resources. In order to classify the applications, we analyze each component at a time.

The features capture the pattern of temporal variation common to the set of original traces; the extent to which this particular temporal pattern is presented on each original application trace is given by the entries of V_i, the eigenvectors.

The entries of the three dominant components are presented in Figure 7. The numbers in the x-axis correspond to the numbers of the applications in Table I. The greater the absolute value of the entry (y-axis), the stronger the correspondent feature is present in the original application load behavior. For example, we can see that in the third component in Figure 7, which corresponds to the "spiky" feature, application 6 has a strong presence of this feature in its behavior. Application 6 is shown in the middle plot of Figure 1. By looking at the plot of the original trace, we can confirm that this application has indeed a strong spiky characteristic.

In the same fashion, applications 2 and 7 have the strongest presence of the periodic behavior, as shown in the left plot of Figure 7. These two applications correspond to the other two applications in Figure 1, which present a strong periodic trend.

By examining the component entries, we can specify a criterion to classify the applications by their predominant features: periodic, noisy or spiky. To illustrate the idea, we use a simple arithmetic criterion to attribute traces into classes. To determine the traces that belong to a class, we select those traces in which their entry in the component is greater the mean value of the absolute values of all entries. That means that all application's entries in the first component greater than 0.27 were considered periodic, the ones greater than 0.26 in the second were considered noisy and the ones greater than 0.20 in the third were considered spiky. The result is summarized in Table II. Although some traces were not uniquely classified, this classification helps us to understand the predominant behavior of each trace.

TABLE II
Application Classification

Num.	**App**	**f1**	**f2**	**f3**	**Class**
1	ARC	0.30	0.09	0.24	Periodic, Spiky
2	B2B	0.37	-0.18	0.08	Periodic
3	Contivo	0.27	0.38	0.19	Noisy
4	Esgui	0.28	-0.16	-0.06	Periodic
5	Parallax	0.29	-0.21	0.36	Periodic, Spiky
6	Primavision	0.14	0.18	-0.78	Spiky
7	Psghrms	0.39	-0.26	-0.15	Periodic
8	Pshd	0.37	-0.24	-0.31	Periodic, Spiky
9	Psportal	0.23	-0.34	0.05	Noisy
10	Rdma	0.22	0.51	-0.05	Noisy
11	Rocket	0.23	0.21	-0.03	Noisy
12	Wwtbl	0.28	0.40	0.17	Periodic,Noisy

B. Detrending Applications

Resource assignment [3] is a useful technique for capacity planning. This method takes advantage of the seasonal complementarities of the periodic behavior of the traces and exploits negative correlations between different applications' demand in a shared resource environment. It also considers the effect of unusual events, such as unexpected peaks.

This capacity planning technique can also benefit from the structural analysis. It is possible to detrend application traces to examine one effect at a time.

Suppose one is interested in determining the seasonal behavior of the applications. In this case, we would be interested mostly in the periodic behavior. Thus, we can reconstruct the original application traces annulling the noisy and spiky components. This will reconstruct a trace governed only by its periodic trend and, therefore, the analysis could be carried out more precisely. Sometimes, in analyzing the original trace directly it is difficult to distinguish and isolate only the effect of the periodicity.

Analyzing the periodic trend alone, one could complement the valley of a period with a peak from another. For example, applications being utilized by users physically located on the East and West coasts, running in the same system, could differ in terms of utilization as a function of the hours of the day and therefore the load periodicity could be complemented in order to maximize resource utilization.

On the other hand, to study anomalies and unexpected events in the system, one could be interested in separate the invisible spikes from the other effects. This can be done by considering only the third component in the reconstruction.

As an illustration we detrend an actual application shown in Figure 8, presenting each effect, period, noise and spike, separately in Figure 9. It is interesting to see the intensity of the noise in the second plot and, also, the spikes in the

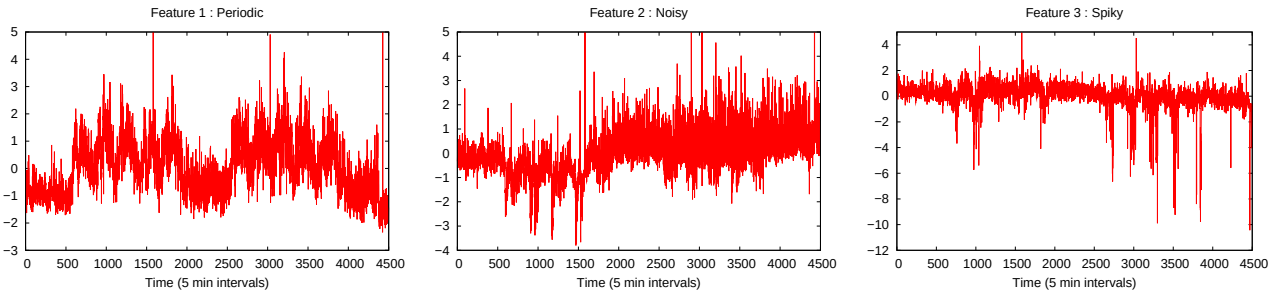

Fig. 6. The First Three Features: Periodic, Noisy, Spiky

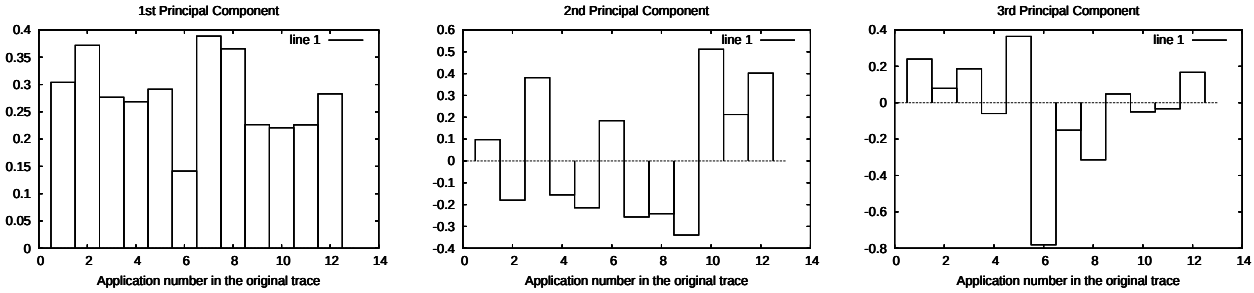

Fig. 7. The First Three Components

third plot, which were indistinguishable when looking at the original trace

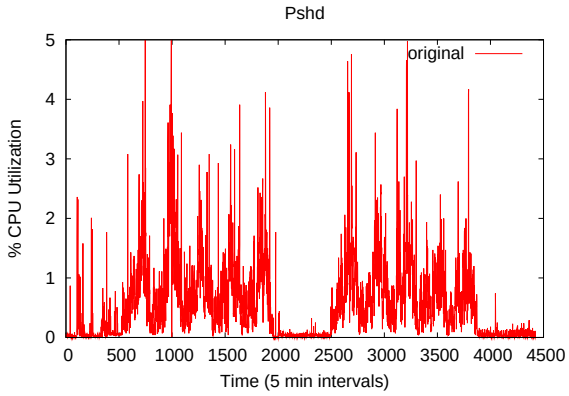

Fig. 8. Pshd: an application to be detrended

C. Generating Synthetic Workload

Synthetic workloads are useful for testing and simulation purposes. Not only do we want to reproduce a typical behavior of the existing applications, but also we would like to create a model and change some of its parameters, in order to evaluate the effect of a new, slightly different application.

As an example, we may want to simulate the behavior of a new application which is connected to a headlines news database. This new application has the same noisy behavior as an existing one, lets say application y. However, due to a special unexpected news event, there is a peak during the business hours of some days. To evaluate the impact of this new application in the system, we can generate a synthetic trace from the y model, intensifying the peak effect of application y. Doing so, we will be able to evaluate the performance implications of the new application in the system due to the effect of burstiness incurred by the special news event.

Similarly, it is also possible to suppress or undermine the presence of a feature in an application trace in order to reproduce some desirable behavior.

The reconstruction of a original trace can be accomplished by

$$U(V^{-1})_i = X_i^r \tag{4}$$

The synthetic trace generation is basically a reconstruction with a new combination of the basic features, that is, playing with the values of the inverted matrix V^{-1}. Equivalently, equation 4 can be written as

$$U \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = X_i^r$$

where v_1 is the parameter related to the periodic component, and v_2 and v_3 to the noisy and spiky components, respectively.

As an illustration, application 2 has the following configuration:

$$U \begin{bmatrix} 0.37 \\ -0.18 \\ 0.08 \end{bmatrix} = X_2^r$$

Lets say that we want to reproduce the behavior of application 2, but with half of the amplitude induced by the periodic feature. In order to produce this result, we just attribute half of the value in entry v_1, that is $v_1 = 0.18$. The resulting trace can be seen in Figure 10. This figure compares the reconstruction of application 2 (left), using the original configuration, to the

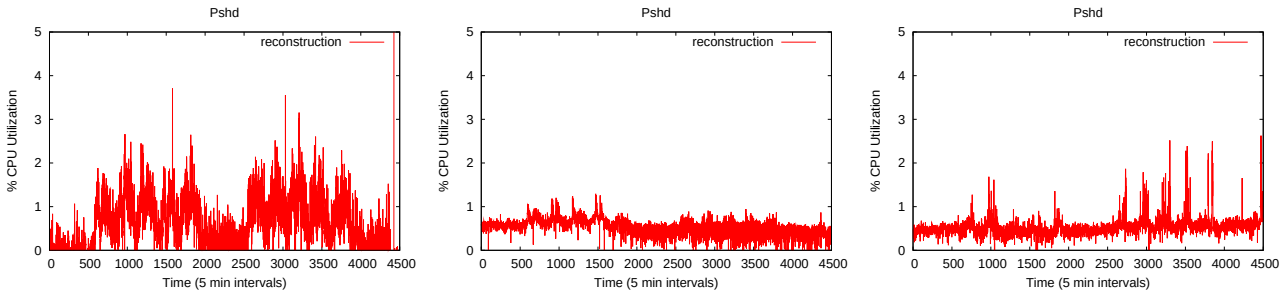

Fig. 9. Detrending Application Pshd (8): a) Periodic, b) Noisy and c) Spiky behaviors from the same application plotted separately

new synthetic generated application (right), that mimics its behavior but presents half of its periodic amplitude.

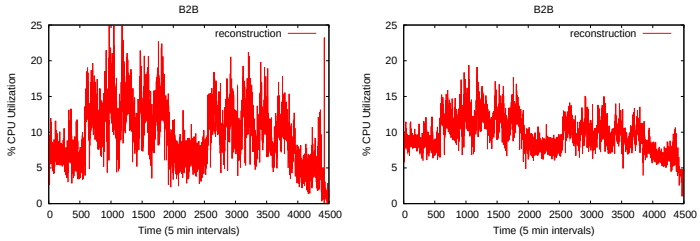

Fig. 10. Comparing the reconstruction of application 2 to a the new synthetic generated trace making a change in its configuration

VII. Conclusions

In order to characterize the workload of applications in a shared utility computing environment, we applied the Principal Component Analysis (PCA) technique to an CPU utilization dataset extracted from an HP SASU's production server. It was possible to create a workload model using a small number of features. We demonstrated how to classify applications by their predominant feature, how to examine the structure of the behavior of an application and how to generate synthetic workload.

Further analysis is required in order to validate the model. For instance, as SASU is a changing environment from which we have obtained our dataset for the PCA analysis, it would be necessary to prove the temporal stability of the applications. We should be able to confirm that, if we want to describe the system behavior in the future, the basic features do not change much with time. In other words, the question is whether the structure described is useful for analyzing data that was not part of the input of the analysis. Furthermore, it is possible that a new application has a completely different behavior from the existing ones. For this reason, the insertion of a new application requires a test to determine whether the PCA model should be rebuilt from scratch.

Another possibility is to automate the process to generate periodical reports to the IT manager in a way that it would be possible to analyze the applications' behaviors during several periods of time.

Finally, we believe that, in the future, when hundreds of applications are being supported and as the CPU utilization increases, the advantage of using the PCA technique in managing the larger system will become more significant – instead of analyzing a large number of times series at a time, one could use this approach in reducing the complexity of system's characterization and synthetic trace generation.

Application behavior on the consumption of other system resources such as network bandwidth and disk I/O can also be similarly analyzed.

VIII. Acknowledgment

We wish to thank Fereydoon Safai, project manager of Capacity Planning with the Decision Technologies Department, Intelligent Enterprise Technologies Laboratory, for his overall support of our research effort. Thanks are due to Cipriano (Pano) Santos for his numerous valuable comments on an earlier version of this paper. We also thank the comments and project help from other members of the project team, Dirk Beyer, Jerry Shan, Yunhong Zhou, and Richard Stormo throughout the project. Finally, we have benefited from the technical discussion with Professor Virgilio A. F. Almeida of Federal University of Minas Gerais, Brazil, in the earlier phases of the project.

References

[1] "HP grid and utility computing," http://devresource.hp.com.
[2] "Shared application server utility - service brief," http://sasu.corp.hp.com/index.htm.
[3] D. Beyer, C. Santos, J. Shan, and Y. Zhou, "SASU resource assignment/capacity planning," HP Labs, Palo Alto - California, Tech. Rep.
[4] F. Safai, A. Zhang, D. Beyer, Y. Zhou, J. Shan, V. Almeida, and R. Stormo, "An optimization model for cost-effective capacity expansion," April 2004, hP Labs - unpublished research note.
[5] F. Safai, J. Shan, D. Beyer, A. Zhang, and S. Jain, "IT capacity planning: A three-stage approach," HP Labs, Palo Alto - California, Tech. Rep., January 2004.
[6] A. Lakhina, K. Papagiannaki, M. Crovella, E. D. K. Christophe Diot (Intel Research), and N. Taft, "Structural analysis of network traffic flows," in *ACM Sigmetrics*, 2004.
[7] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak, "Statistical service assurance for applications in utility grid environments," HP Labs, Tech. Rep., 2002, hPL-2002-155.
[8] J. Rolia, A. Andrzejak, and M. Arlitt, "Automating enterprise application placement in resource utilities," in *Self-Managing Distributed Systems: 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2003*, vol. 2867/2004, January 2004, pp. 118–129.
[9] D. Menascé and V. Almeida, *Capacity Planning for Web Services*. Prentice Hall, 2000.
[10] D. Thiébaut, "On the fractal dimension of computer programs and its application to the prediction of the cache miss ratio," *IEEE Transactions on Computers*, vol. Volume 38 , Issue 7, July 1989.

[11] H. Hotelling, *Analysis of a complex of statistical variables into principal components*. J. Educ, Psy, 1933.