Hierarchical Means: Single Number Benchmarking with Workload Cluster Analysis

Richard M. Yoo[†], Hsien-Hsin S. Lee[†], Han Lee[‡], and Kingsum Chow[‡]
School of Electrical and Computer Engineering[†]
Georgia Institute of Technology, Atlanta, GA 30332
Managed Runtime Division, Software and Solutions Group[‡]
Intel Corp., Hillsboro, OR 97123
{yoo, leehs}@ece.gatech.edu[†], {han.lee, kingsum.chow}@intel.com[‡]

***Abstract*— Benchmark suite scores are typically calculated by averaging the performance of each individual workload. The scores are inherently affected by the distribution of workloads. Given the applications of a benchmark suite are typically contributed by many consortium members, workload redundancy becomes inevitable. Especially, the merger of the benchmarks can significantly increase artificial redundancy. Redundancy in the workloads of a benchmark suite renders the benchmark scores biased, making the score of a suite susceptible to malicious tweaks. The current standard workaround method to alleviating the redundancy issue is to weigh each individual workload during the final score calculation. Unfortunately, such a weight-based score adjustment can significantly undermine the credibility of the objectiveness of benchmark scores. In this paper, we propose a set of benchmark suite score calculation methods called the hierarchical means that incorporate cluster analysis to amortize the negative effect of workload redundancy. These methods not only improve the accuracy and robustness of the score, but also improve the objectiveness over the weight-based approach. In addition, they can also be used to analyze the inherent redundancy and cluster characteristics in a quantitative manner for evaluating a new benchmark suite. In our case study, the hierarchical geometric mean was applied to a hypothetical Java benchmark suite, which attempts to model the upcoming release of the new SPECjvm benchmark suite. In addition, we also show that benchmark suite clustering heavily depends on how the workloads are characterized.**

I. Introduction

Processor architects use benchmark suites composed of real applications of interests to set the performance goals for a particular market segment a processor is designed for. These benchmark suites are used in early architecture planning phases to perform design space exploration and to obtain the trade-off among cost, performance, complexity-effectiveness, and energy consumption. Thus, the success of a new processor design can heavily depend on the selection and the method of calculating the overall benchmark score. Furthermore, different system vendors also rely on (standard) benchmark suites to compare their products against those offered by their competitors, making an accurate evaluation and a fair representation of the overall benchmarking results highly desirable.

The selection and inclusion of programs for a standard benchmark suite are typically done by a consortium formed by several member companies and academic institutions. Due to the aggregation of benchmark programs contributed from different parties, a composite benchmark suite often contains workload redundancy — i.e., several programs exhibit similar execution behavior. To be more specific, there are actually two types of workload redundancy: natural redundancy and artificial redundancy. *Natural redundancy* occurs when we sample the user workload space evenly. For example, since a large number of the consumer workloads are memory-intensive, it is likely that the final benchmark suite includes more memory intensive workloads instead of floating-point or I/O intensive workloads if we sample the user workload space. Since natural redundancy reflects the actual user workload spectrum, it is hard to conclude that natural redundancy is harmful.

In contrast, *artificial redundancy* happens when a new benchmark suite is created by merging a set of benchmark suites. Due to the increasing pressure from time and the lack of domain knowledge, it is getting more popular to release a new benchmark by merging workloads directly from existing benchmark suites. This trend is not limited to academic benchmarks, but also extends to standard benchmarks. For instance, the next release of MineBench [1], a data mining benchmark from academia, will incorporate workloads from ClusBench [2], a clustering workloads benchmark. Moreover, it has been indicated that the next version of the SPECjvm benchmark suite, SPECjvm2007, will incorporate workloads from the SciMark2 benchmark suite [3], [4]. Besides, the credibility of the new benchmark suite can also be inherited from the proven credibility of those existing workloads, providing a good justification for including them.

Unfortunately, such a workload adoption process tends to significantly increase artificial redundancy. Suddenly introducing a couple of foreign workloads into a self-contained benchmark suite can dramatically increase the workload redundancy, if the characteristics of the newly added workloads fail to demonstrate evenly diversified behavior across the entire workload spectrum. As such, these injected workloads will form an exclusive cluster of their own, hence rendering each other in the adoption set redundant.

For a proprietary benchmark suite, workload redundancy only affects the accuracy of the benchmark suite itself. However, workload redundancy in a standard benchmark suite can lead to many problems. When there exists homogeneous workloads in a benchmark suite, their similarity can significantly distort the scoring metric by amplifying their aggregated effect on the overall score. For example, if only two homogeneous workloads benefit from the increased cache size, the effect of this architectural parameter will be erroneously evaluated twice,

thereby undesirably enlarging the benefit from using a larger cache. Note that compiler or hardware enhancement techniques will be misleadingly targeted for those redundant workloads for such improvements. This not only affects the accuracy, but also the robustness of the scoring metric, therefore jeopardizing the credibility of the benchmark itself. As we can see, workload redundancy, especially the artificial redundancy, is a significant problem.

If detected, it would be the best to remove those redundant workloads from the benchmark suite. However, mutual interests from different parties can make this task rather difficult and political. As an example, consider the case where we create a benchmark suite by merging data mining and bioinformatics workloads. Since bioinformatics workloads are a subset of data mining workloads, most of the bioinformatics workloads would be redundant, when compared to the more general data mining workload. Nonetheless, if those applications in bioinformatics benchmark are equally important in bioinformatics area, it would be hard to drop any of those workloads. Although it might sound far fetched, this example accurately reflects the dilemma of a consortium-driven benchmark creation process. Based on some publicly available information [3], [4], it seems highly likely that most of the SciMark2 workloads will be included in SPECjvm2007 despite of the potential redundancy issue.

In such scenarios, we should rely on score calculation metrics to remove redundancy. Calculating averages over the workloads simply overlooks this redundancy problem. To cope with the problem, the scoring method itself should be aware of the workload redundancy. One possible solution, the one currently in use, is to augment the plain mean calculation with different weights for different workloads: the *weighted mean* approach. Nevertheless, this approach can undermine the objectiveness of a benchmark, since determining the exact value of those weights is always subjective.

To address these issues, in this paper, we propose a method that incorporates workload cluster information directly into the shape of the scoring equation. Specifically, we propose a set of new scoring methods called the *hierarchical means*, which are based on statistical analysis. These methods effectively cancel out the negative effects of workload redundancy. In our case study, we first conjured up a benchmark suite which attempts to model the upcoming new SPECjvm benchmark suite, based on some publicly available information. Then we applied one form of our hierarchical means, the hierarchical geometric mean, to this benchmark suite to study and analyze its behavior.

In summary, the main contribution of this paper is twofold.

- We propose a statistical approach to incorporate workload cluster information in benchmark suite score calculation.
- Based on this approach, we propose the use of a new set of scoring metrics.

The rest of the paper is organized as follows. In the next section we introduce the hierarchical means. Section III discusses the technique to detect clusters in a benchmark suite. The experimental settings and the results of our case study are presented in Section IV and Section V, respectively. Related research in workload characterization was described in Section VI. Finally, we conclude in Section VII.

II. The Hierarchical Means

In benchmark suite cluster analysis, workloads that demonstrate similar characteristics such as similar cache behavior, number of page faults, computational intensity, etc., are classified into the same cluster. Assuming that this workload cluster information is available, in this section we show how we incorporate this information into our new scoring methods — the hierarchical means. We discuss clustering and workload characterization in Section III.

For a benchmark suite comprised of n workloads, where the i^{th} workload showing performance value X_i, a plain geometric mean is calculated as:

$$\sqrt[n]{X_1 X_2 ... X_n}$$

For the same benchmark suite, given the workload cluster information, if the benchmark suite forms $i = 1, ..., k$ clusters, *Hierarchical Geometric Mean* (HGM) is calculated as:

$$\sqrt[k]{\sqrt[n_1]{X_{11}...X_{1n_1}}... \sqrt[n_k]{X_{k1}...X_{kn_k}}}$$

where n_i stands for the number of workloads in the i^{th} cluster, and X_{ij} stands for the performance of the j^{th} workload in the i^{th} cluster.

Simply put, HGM is a geometric mean of geometric means; each inner geometric mean reduces each cluster to a single representative value, which effectively cancels out the workload redundancy, while the outer geometric mean equalizes each cluster. Note that, when each workload is assigned a single cluster, the HGM gracefully degenerates to the plain geometric mean as shown below.

$$\sqrt[n]{\sqrt[1]{X_{11}}\sqrt[1]{X_{21}}...\sqrt[1]{X_{n1}}} = \sqrt[n]{X_1 X_2 ... X_n}$$

The key approach of HGM is to apply averaging process in a hierarchical manner to eliminate the workload redundancy. This approach can be readily extended to the arithmetic mean and the harmonic mean. For example, for the arithmetic mean, the *Hierarchical Arithmetic Mean* (HAM) can be calculated as:

$$\frac{\frac{X_{11}+...+X_{1n_1}}{n_1}+...+\frac{X_{k1}+...+X_{kn_k}}{n_k}}{k}$$

while for the harmonic mean, the *Hierarchical Harmonic Mean* (HHM) is calculated as:

$$\frac{k}{\frac{\Sigma_{j=1}^{n_1}\frac{1}{X_{1j}}}{n_1}+...+\frac{\Sigma_{j=1}^{n_k}\frac{1}{X_{kj}}}{n_k}}$$

Similar to the HGM, the HAM and HHM also gracefully degenerate to their respective plain means when each cluster contains only one single workload.

Compared to the conventional workaround relying on human's intervention for weighing workloads to reduce redundancy, the hierarchical means are more objective given that the clustering is performed based on a quantitative method.

III. Detecting Clusters in a Benchmark Suite

Clustering the applications for a benchmark suite is essentially a workload characterization problem. Workload characterization is a process that maps a workload to a *characteristic*

vector which is comprised of elements that best characterize the workloads such as the number of cache misses, the number of page faults, the ratio of computation to communication, etc. These elements can be anything from hardware performance counters, operating system counters, and Java Virtual Machine (JVM) counters, to microarchitecture-independent characteristics [5], [6] of the characterized workload. The values of the elements are usually obtained by sampling the execution behavior.

Clusters in a benchmark suite can be detected by first applying workload characterization to each workload and then performing distance based clustering analysis over the characteristic vectors. However, due to the high dimensionality of the characteristic vectors and the correlation among characteristic vector elements, dimension reduction and transformation will be necessary. We apply the Self-Organizing Map [7], [8] to satisfy this aim. Then the Hierarchical Clustering is applied to the reduced dimension, which yields the workload clusters.

A. Self-Organizing Map

Self-Organizing Map (SOM) [7], [8] is a special type of neural network which effectively maps high-dimensional data to a much lower dimension, typically 1-D or 2-D. It creates a visual *map* on the lower dimension such that two vectors that were close in the original n-dimension appear closer, and those distant ones appear farther apart from each other. Note that the reduced dimension has no physical interpretation as to the original dimension but the relative distance among the n-dimensional vectors. Due to this characteristic, SOMs are usually applied to obtain a better visualization for higher dimensional data.

By applying the SOM to a set of characteristic vectors for each workload, we can construct a visual map that discerns which workloads are similar. When two workloads appear closely on the map, it means that their n-dimensional characteristic vectors were close, in other words, these workloads are homogeneous.

The construction of a SOM is shown in Figure 1. Similar to the illustration, a SOM is typically comprised of a 2-D array of neurons, called *units*. Each unit contains a weight vector w_i and a location vector r_i. The weight vector has the same dimension with each characteristic vector, and the location vector specifies the unit's location in the 2-D grid. Note that, each characteristic vector will be broadcast to all the units. In the figure, for clarity, we only show the connections in the bottom row.

Training the SOM is simple. It is based on competitive learning [8]; each unit competes to resemble more about each input characteristic vector representing one application. The pseudo code is listed as follows.

Given:

A 2-D array of units, and a set of characteristic vectors

Initialize:

Assign a random initial value to each unit's weight vector

Repeat:

Randomly select a characteristic vector

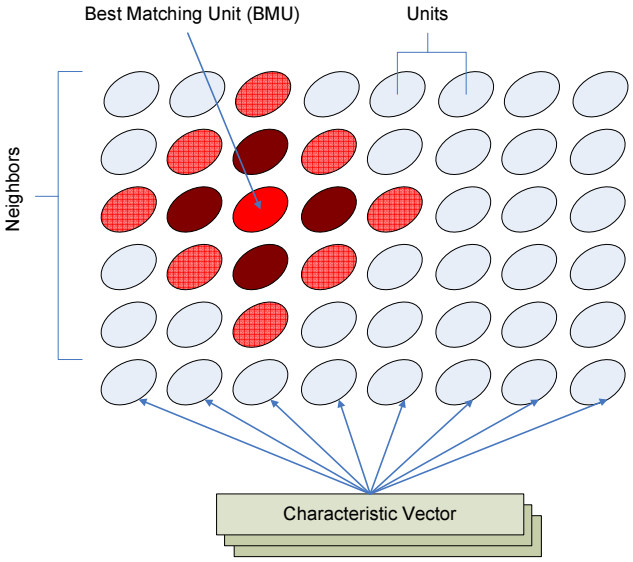


Fig. 1. A Typical Structure of SOM

Get the best matching unit

Adjust the weight of itself and its neighbors

Continue until converge

The first step in constructing a SOM is to initialize the weight vectors. The initial values of these weights are usually determined by sampling a subspace generated by the two major principal components [9] of the characteristic vectors. Nonetheless, the selection of the initial weights does not affect the final result significantly.

After initialization, at each step, a characteristic vector representing a workload is randomly chosen, and a unit that best matches this characteristic vector is selected. A *best matching unit* (BMU) is defined as the unit that has the minimum Euclidean distance ($d_i = \sqrt{\sum_j ||x_j - w_{ij}||^2}$) between an input characteristic vector (x) and its weight vector(w_i).

In Figure 1, a BMU is highlighted in the middle of its neighbors. When a BMU is found for a given characteristic vector, the weight vector of that BMU is adjusted using the following equation. As a result, the BMU's weight vector will become even more similar to that characteristic vector. In the same way, the weight vectors of the *neighbors* of that unit are also adjusted.

$$w_i(n+1) = w_i(n) + h_{ci}(n)[x(n) - w_i(n)], \text{ where}$$

$$h_{ci}(n) = \alpha(n) * exp(-\frac{||r_c - r_i||^2}{2\sigma^2(n)})$$

In this weight vector adjustment, the new weight vector $w_i(n+1)$ is incrementally updated by a product of two items: the neighborhood kernel, h_{ci}, and the difference of the characteristic vector $x(n)$ and the current weight vector $w_i(n)$.

The neighborhood kernel is essentially a Gaussian function of the distance from the BMU. In this function, the r_c and r_i denotes the location vectors of the BMU and the i^{th} unit, respectively. Moreover, $\alpha(n)$ denotes the "learning-rate factor." This value determines the magnitude of the h_{ci} function; the larger the $\alpha(n)$ value, the larger the assimilation. The function $\sigma(n)$ controls the radius of the "neighbor"; neighborhood is defined as the range where h_{ci} assumes a non-zero value. Both $\alpha(n)$ and $\sigma(n)$ monotonically decreases as we progress for each learning step n. Figure 2 plots an h_{ci} function as n increases.

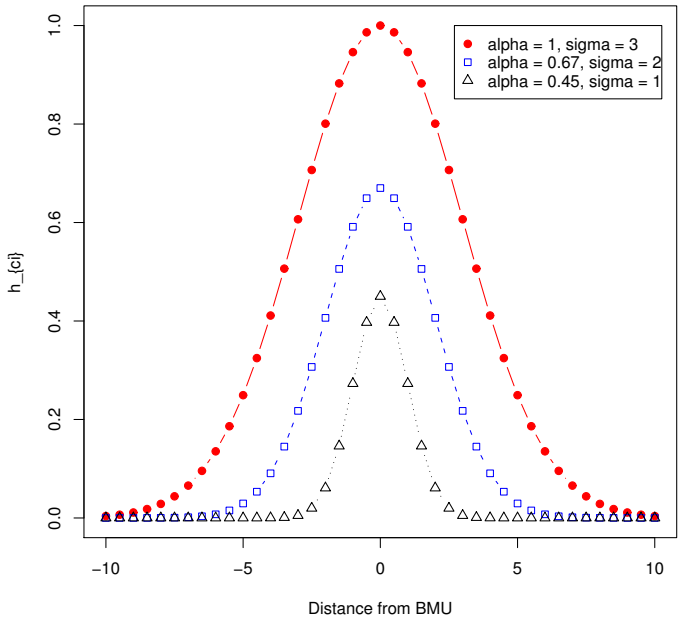


Fig. 2. Behavior of the h_{ci} Function

As the training process continues, the units that resembled the characteristic vector gets more like the characteristic vector, while the other units that did not become less alike the characteristic vector.

When the training is finished, we have a 2-D array of units where each unit responds to a specific input characteristic vector, in other words, each workload will be mapped to one particular unit. Note that when two or more workloads are similar enough, they can map to the same unit.

Compared to the previous approaches [5], [10], [11], [12] which used Principal Components Analysis (PCA) as a dimension reduction tool, SOM has a benefit that it preserves the entire information contained in the original dimension. In PCA, selectively choosing a few major principal components results in loss of information. This loss of information can be significant when the input characteristic vectors do not show a strict tendency over the dimension on which principal components span. Since PCA tries to map the input data with only linear components, the reduced dimension generated by the linear combinations of principal components can be a poor approximation of the original data if the inherent characteristic of input data is non-linear [9].

Moreover, when there are more than 2 principal components, it is hard to visualize the workload distribution in an intuitive manner. SOM is a good alternative for an accurate representation of high dimensional data in 2-D.

Using the 2-D map generated by SOM, we now focus on how to perform clustering.

B. Hierarchical Clustering

To obtain workload cluster information, we then apply hierarchical clustering to the reduced dimension generated by SOM. The pseudo code for hierarchical clustering is illustrated as follows.

Given:
X, *a set of m training points*

Initialize:
Assign each training point(X_i) *to a single cluster*($\mathbf{w}_i$)
($\mathbf{w}_i = X_i, i = 1, ..., m$)

Repeat:
Compute cluster-to-cluster distance $d(\mathbf{w}_i, \mathbf{w}_j)$ *for all pairs of clusters*

Find two clusters ($\mathbf{w}_p$, $\mathbf{w}_q$) *such that their distance is the minimum among all pairs of clusters*

Create a new cluster by merging those two clusters ($\mathbf{w}_k = \mathbf{w}_p \cup \mathbf{w}_q$)

Continue until all the points result in a single cluster

In this notation, $d(\mathbf{w}_i, \mathbf{w}_j)$ stands for cluster-to-cluster distance. Among the many cluster-to-cluster distance definitions, in our specific case we chose it to be the distance of the furthest pair of points from each cluster. In mathematical notation, $d(\mathbf{w}_i, \mathbf{w}_j) = max_{x \in \mathbf{w}_i, y \in \mathbf{w}_j} d(x, y)$. Again, in this notation, $d(x, y)$ stands for point-to-point distance. We chose Euclidean distance as the point-to-point distance.

In the beginning, the algorithm assigns each point a cluster. At each iteration the closest pair of clusters are merged to create a new cluster, reducing the number of clusters by one each time. The algorithm proceeds until all the points result in a single cluster. Clustering result can be represented as a *dendrogram* which visualize which workloads form a cluster at which merging distance. At a specific merging distance, clusters that are located closer than the merging distance should merge. The lower the merging distance the more "similarity" those workloads have. By varying the merging distance, we can determine how many workload clusters exist in a benchmark suite.

IV. Experimental Settings

In this section we discuss our settings for the case study to demonstrate our methodology. For the experiments, we composed a hypothetical Java benchmark suite which attempts to model the upcoming new SPECjvm benchmark suite. This benchmark suite is detailed in Section IV-A. The benchmark was then executed on two different machines to compare

Workload	Benchmark Suite	Version	Input Set	Description
201.compress	SPECjvm98	1.04	s100	A Java port of the 129.compress benchmark from SPEC CPU95, which implements modified Lempel-Ziv method (LZW).
202.jess	SPECjvm98	1.04	s100	A Java Expert Shell System based on NASA's CLIPS expert shell system. The workload solves a set of puzzles commonly used with CLIPS by applying a set of if-then statements to a set of data.
213.javac	SPECjvm98	1.04	s100	The Java compiler from the JDK 1.0.2.
222.mpegaudio	SPECjvm98	1.04	s100	An application that decompresses audio files that conform to the ISO MPEG Layer-3 audio specification.
227.mtrt	SPECjvm98	1.04	s100	A multi-threaded raytracer that works on a scene depicting a dinosaur.
FFT	SciMark2	2.0	regular	Performs a one-dimensional forward transform of 4K complex numbers. This kernel exercises complex arithmetic, shuffling, non-constant memory references and trigonometric functions.
LU	SciMark2	2.0	regular	Computes the LU factorization of a dense 100x100 matrix using partial pivoting. Exercises linear algebra kernels (BLAS) and dense matrix operations.
MonteCarlo	SciMark2	2.0	regular	Approximates the value of Pi by computing the integral of the quarter circle $y = \sqrt{1 - x^2}$ on [0,1]. It chooses random points with the unit square and compute the ratio of those within the circle.
SOR	SciMark2	2.0	regular	Performs Jacobi Successive Over-relaxation (SOR) on a 100x100 grid. Exercises typical access patterns in finite difference applications.
Sparse	SciMark2	2.0	regular	Uses an unstructured sparse matrix stored in compressed-row format with a prescribed sparsity structure. This kernel exercises indirection addressing and non-regular memory references.
Hsqldb	DaCapo	2006-08	default	Executes a JDBCbench-like in-memory benchmark, executing a number of transactions against a model of a banking application.
Chart	DaCapo	2006-08	default	Uses JFreeChart to plot a number of complex line graphs and renders them as PDF.
Xalan	DaCapo	2006-08	default	Transforms XML documents into HTML.

TABLE I
CONSTRUCTED BENCHMARK SUITE

their performance; individual workload scores and workload cluster information were obtained from those two machines. A reference machine was also introduced for the baseline workload score. The hardware settings will be discussed in Section IV-B. Section IV-C describes our workload characterization methodology.

A. Benchmark Suite Composition

Based on the publicly available information, we composed a hypothetical Java benchmark suite which attempts to model the upcoming new SPECjvm benchmark suite by adopting workloads from 3 different benchmark suites: SPECjvm98 [13], SciMark2, and DaCapo [14] benchmark suite. Table I describes the composed benchmark. In total, there are 13 workloads in our benchmark suite. Among those workloads, 5 of them are retained from the SPECjvm98, the other 5 of them are adopted from SciMark2, and the rest are from DaCapo.

SPECjvm98 [13] has been by far the standard in client side Java benchmark. SciMark2 is a Java benchmark suite for scientific and numerical computing, and has been indicated to be included in the next release of SPECjvm benchmark suite [3], [4]. It specifically addresses the numeric computation capability of the underlying Java framework. In contrast, DaCapo benchmark suite [14] is being developed for garbage collection research; it shows significantly increased execution time and object creation, large enough to trigger frequent garbage collection. Inclusion of DaCapo suite reflects the prolonged needs for a longer and heavier benchmark from the virtual machine researchers. As a score metric for the individual workload, we use the execution time speedup over a reference machine.

To the best of our knowledge, we believe that this benchmark suite would be representative enough to imitate the current status of the upcoming SPECjvm benchmark suite although the actual release version is yet to be disclosed and may eventually be different.

B. Hardware Settings

We executed the above benchmark on 3 different machines: machine A, B, and a reference machine listed in Table II. Machine A and B are the machines for performance comparison, while the reference machine is used to normalize the execution time of each workload. Each workload was executed 10 times on each machine, and the average execution time was used as a representative program execution time.

C. Workload Characterization

We applied two types of workload characterization. In the first approach (Section V-B), to determine the effect of machine specific cluster information on the hierarchical means, we used the SAR counters provided by Linux. The SAR program collects operating system level performance counters such as CPU utilization, the numbers of context switches, interrupts,

Machine A

CPU	Dual Intel Xeon CPU 3.00 GHz HyperThreading disabled
L2 Cache	2 MB
Bus Speed	800 MHz
Memory	2 GB
OS	Red Hat Enterprise Linux WS release 4 2.6.9-34.0.1.ELsmp
JVM	BEA JRockit R26.4.0-jdk1.5.0_06 32 bit Edition

Machine B

CPU	Intel Pentium 4 CPU 3.00 GHz HyperThreading disabled
L2 Cache	512 KB
Bus Speed	800 MHz
Memory	512 MB
OS	Red Hat Enterprise Linux WS release 4 2.6.9-42.0.3.ELsmp
JVM	BEA JRockit R26.4.0-jdk1.5.0_06 32 bit Edition

Reference Machine

CPU	Sun UltraSPARC III Cu 1.2 GHz
L2 Cache	8 MB External
Bus Speed	800 MHz
Memory	1 GB
OS	Solaris 8
JVM	Sun Java HotSpot build 1.5.0_09-b01

TABLE II
HARDWARE SETTINGS

page misses, etc. Due to the virtual execution environment of Java applications, operating system level counters become significant in characterizing Java workloads [15], [16]. We used a couple hundred counters by collecting all the counters that SAR provides. While the workload was executed till completion for 10 times, 15 samples were collected for each counter, with an even time interval. In characteristic vector for each workload, the average value of those samples was used as a representative counter value. Those counters that did not vary over workloads were discarded because they provide no useful information in distinguishing workloads. Moreover, each counter was standardized prior to the cluster analysis, i.e., subtract the mean and divide by standard deviation.

In our second approach (Section V-C), to workaround the dependency of hierarchical means on machine characteristics, we employed a totally architecture independent characteristic: the Java methods usage. Note that this characteristic totally depends on the organization of the source code itself. Since Java provides heavy library support via standardized API, characterization of Java workloads by method profiling has been popular in many researches [17], [18]. While the workloads are running, we collect the method coverage information with hprof facility provided by recent JVMs. Then we create a list of the complete method names (e.g., java.lang.String.subString, java.lang.Object.hashCode, etc.) that appear on the hprof result along with a characteristic vector of the same width as the number of Java methods. When a certain method is called by a workload, the corresponding bit in the characteristic vector for that workload is set to 1; otherwise it is set to 0. The characteristic vectors are then used to determine workload clusters. We discarded those methods that 1) only one workload used, or 2) all the workloads used, since these two extremes tend to bias the SOM learning process. The bit fields in characteristic vectors were also standardized. Note that SOM shows robust behavior over PCA approach, for this type of discrete data shows high nonlinearity [9]. For non-Java workloads, other microarchitecture independent workload features such as instruction mix, memory strides, etc. [5], [6] can be used instead.

V. RESULTS

In section V-A we provide the speedup of each workload on machine A and B, with respect to the reference machine. Then we study the behavior of hierarchical means in Section V-B and Section V-C when the workload cluster is determined with operating system level counters, and when determined with the language level feature.

A. Workload Execution Time

	A	B	ratio(=A/B)
jvm98.201.compress	4.75	3.99	1.19
jvm98.202.jess	5.32	3.65	1.46
jvm98.213.javac	3.97	2.37	1.68
jvm98.222.mpegaudio	6.50	6.11	1.06
jvm98.227.mtrt	2.57	1.41	1.82
SciMark2.FFT	1.09	1.07	1.02
SciMark2.LU	1.19	0.90	1.32
SciMark2.MonteCarlo	0.75	0.98	0.76
SciMark2.SOR	1.22	1.31	0.93
SciMark2.Sparse	0.71	0.90	0.80
DaCapo.hsqldb	1.16	2.31	0.50
DaCapo.chart	5.12	2.77	1.85
DaCapo.xalan	1.88	2.62	0.71
Geometric Mean	2.10	1.94	1.08

TABLE III
RELATIVE WORKLOAD SPEEDUP ON MACHINES A AND B

The relative speedup of workloads on each machine is given in Table III. At the end of the table shows the overall score calculated by plain geometric means. In this scoring metric, machine A shows an 8% performance improvement over machine B. This will be used as our base of improvement in the following sections.

B. Cluster Analysis Based on SAR Counters

Clustering results can appear differently on different machines. To study the effect of this behavior on the hierarchical means, we analyzed the cluster distribution on both machine A and B. Then we compare the scoring results from hierarchical means. Section V-B.1 and Section V-B.2 discuss about the results on machine A and B, respectively.

1) Cluster Analysis on Machine A: Figure 3 illustrates the results of applying the SOM to SAR counter samples collected on machine A. In this figure, colored cells represent the location of the workloads on the reduced dimension. The closer the two cells are, the more similar the two workloads represented by those cells. For example, workloads FFT and LU from SciMark2, and mtrt and jess from SPECjvm98 are similar when characterized based on SAR counters.

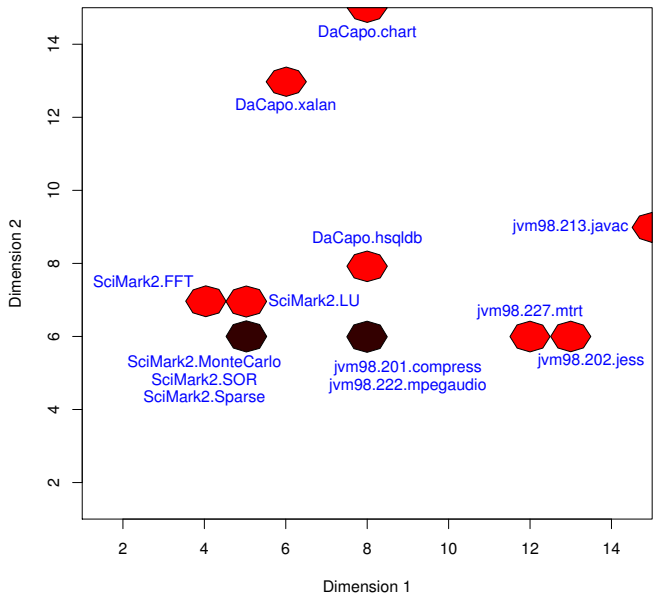


Fig. 3. Workload Distribution on Machine A

Moreover, darker cells indicate that there are multiple workloads that map to the same cell. These workloads can be regarded as particularly similar. MonteCarlo, SOR, and Sparse from SciMark2 fall in this case. Compress and mpegaudio from SPECjvm98 also tends to highly resemble each other.

Note that the distance among the workloads represent only the relative similarity in the benchmark suite under consideration. So in a different benchmark suite composition, two workloads that appear similar in our benchmark suite might appear distinct. Nonetheless, one noticeable trend from the figure is the coagulation of SciMark2 workloads. Workloads from DaCapo benchmark suite tend to spread across Dimension 2, while SPECjvm98 workloads spread across Dimension 1. In contrast, workloads from SciMark2 form a dense cluster around (Dimension 1, Dimension 2) = (5, 7). This means that when compared to the other workloads in the benchmark suite, SciMark2 workloads are radically different that they fail to mix in with the rest. Moreover, since they form a highly dense cluster of their own, workloads in SciMark2 become redundant to each other. Nonetheless, since they occupy the majority of the benchmark suite, 5 out of 13 workloads, it is highly likely that these workloads might significantly affect and bias the overall scoring metric. This particular behavior of SciMark2 workloads is somewhat expected, because SciMark2 explicitly stresses numerical computation capabilities of the underlying Java framework.

The clustering behavior of SciMark2 becomes prominent when we draw dendrograms to determine the number of clusters. Figure 4 represents the clustering results based on the relative distances measured from Figure 3.

In each of these dendrograms, the y-axis indicates the merging distance, and the boxes grouping the workloads represent the cluster formation at that particular merging distance. Workloads that locate closer to each other than the merging distance form a cluster; so with the same workload distribution, depending on the value of the merging distance we choose, multiple clustering decisions can be made. For example, in Figure 4(a), when the merging distance is set to 4, the entire benchmark suite is divided into 4 clusters — javac cluster of its own, two clusters comprising jess and mtrt, and chart and xalan, and the other cluster comprised of the rest of the workloads. In a similar manner, Figure 4(b) shows that 6 clusters are formed when the merging distance is set to 2.

In particular, Figure 4(b) aligns well with the information we obtained by analyzing Figure 3. In this figure, SciMark2 forms a cluster of their own at a merging distance around 2. The low merging distance also means that the clusters formed in the figure are more dense when compared to those clusters formed at a higher merging distance. Note that, at the same merging distance, workloads from DaCapo and SPECjvm98 are already divided into multiple clusters. Thus, not only from the observation of the SOM diagram in Figure 3, but also from these dendrograms, it is confirmed that SciMark2 workloads form a dense cluster of their own on machine A.

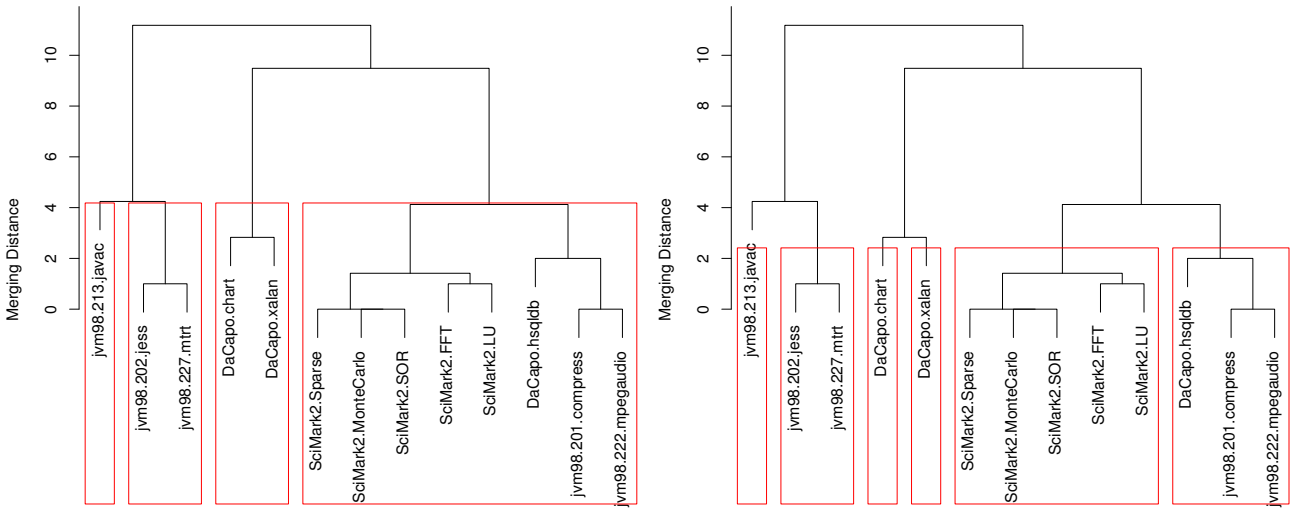
	A	B	ratio(=A/B)
2 Clusters	2.58	2.06	1.25
3 Clusters	2.62	2.18	1.20
4 Clusters	2.89	2.22	1.30
5 Clusters	2.70	2.24	1.21
6 Clusters	2.77	2.31	1.20
7 Clusters	2.63	2.40	1.10
8 Clusters	2.34	2.15	1.09
Geometric Mean	2.10	1.94	1.08

TABLE IV
HIERARCHICAL GEOMETRIC MEAN BASED ON CLUSTERING RESULTS FROM MACHINE A

With these clustering results, we can then apply the hierarchical geometric mean to calculate the overall score that offsets the effect from workload redundancy. Table IV shows the results. Each row in the table denotes the score calculated from the hierarchical geometric mean at the specific cluster distribution. Note that, as the number of clusters increases, the ratio of two scores over machine A and B converges to the ratio of the plain geometric mean (=1.08). Also, note that the ratio could be quite different from the case of the plain geometric mean when the effect from workload redundancy has been removed. For the score distribution on machine A, we recommend the 6 clusters case as the norm since 1) it aligns well with the SOM analysis results, and 2) since the fluctuation of ratio values tends to dampen around 5, 6 cluster cases.

2) Cluster Analysis on Machine B: The same cluster analysis as in Section V-B.1 was performed on the SAR counters collected for machine B. Firstly, Figure 5 shows the SOM analysis results for workloads executed on machine B.

Although the distribution is somewhat different from Figure 3, workloads from DaCapo and SPECjvm98 suite still spread across Dimension 2 and Dimension 1, respectively. As shown, the SciMark2 workloads again form a dense cluster near the lower left of the figure. This behavior is significant since SciMark2 workloads appear as a single cluster on two different machines. This again endorses that SciMark2 workloads are



(a) 4 Clusters

(b) 6 Clusters

Fig. 4. Clustering Results on Machine A

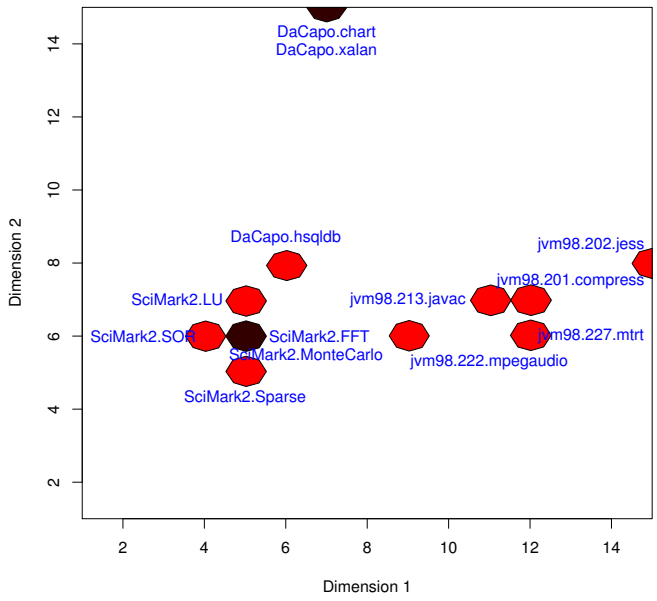


Fig. 5. Workload Distribution on Machine B

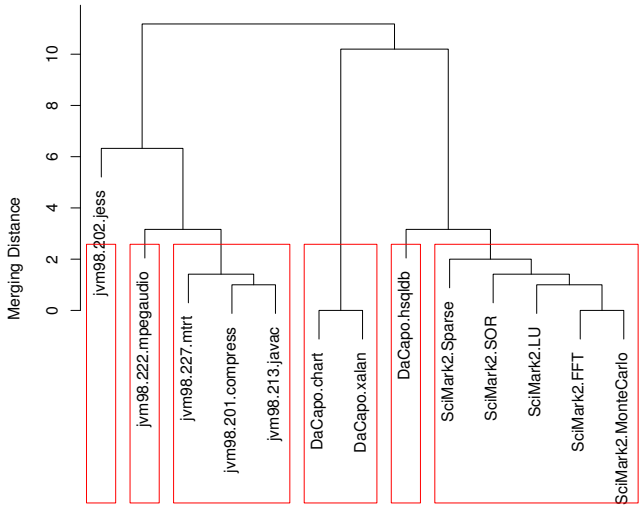


Fig. 6. Clustering Results on Machine B

indeed similar, when characterized with SAR operating system level counters.

Figure 6 shows the dendrogram for the cluster analysis performed for machine B. When the merging distance is chosen as 3, SciMark2 workloads again manifest as an exclusive cluster.

The overall benchmark scores calculated by the hierarchical geometric mean, based on the clustering result on machine B, is given in Table V.

In this score distribution, as with the same reason for the case of machine A, 5 or 6 cluster case seems to be the most representative. However, the ratio for this case, 1.02 $\sim$ 1.04 is quite different from the case for machine A which shows a ratio of 1.20 $\sim$ 1.21 in its 5, 6 cluster cases. This seemingly different result comes from the different cluster distributions on each machine. We should emphasize that, in order to accept the hierarchical means as a standard, a reference cluster distribution on a reference machine should be determined first since clusters might appear differently on different machines. On the contrary, since the SciMark2 workloads appeared as a single cluster on

	A	B	ratio(=A/B)
2 Clusters	2.42	2.12	1.14
3 Clusters	2.39	2.14	1.11
4 Clusters	2.88	2.42	1.19
5 Clusters	2.39	2.34	1.02
6 Clusters	2.75	2.64	1.04
7 Clusters	2.30	2.27	1.01
8 Clusters	2.11	2.10	1.00
Geometric Mean	2.10	1.94	1.08

TABLE V
HIERARCHICAL GEOMETRIC MEAN BASED ON CLUSTERING RESULTS FROM MACHINE B

both machines, we can now assert that SciMark2 workloads should be treated as a single cluster no matter which cluster distribution is chosen.

To amortize the effect of machine specific clustering information on the hierarchical means, in the following section, we study the behavior of the hierarchical geometric means when the cluster information is obtained from a totally machine independent feature: the utilization of Java methods.

C. Cluster Analysis Based on Java Method Utilization

As a workaround to the dependence of hierarchical means on the machine specific clustering results, we characterize the workloads by their Java method utilization. Method utilization of each workload was represented as a bit vector. These vectors were then used as characteristic vectors to determine the workload cluster. This clustering result is totally independent from the architectural characteristics.

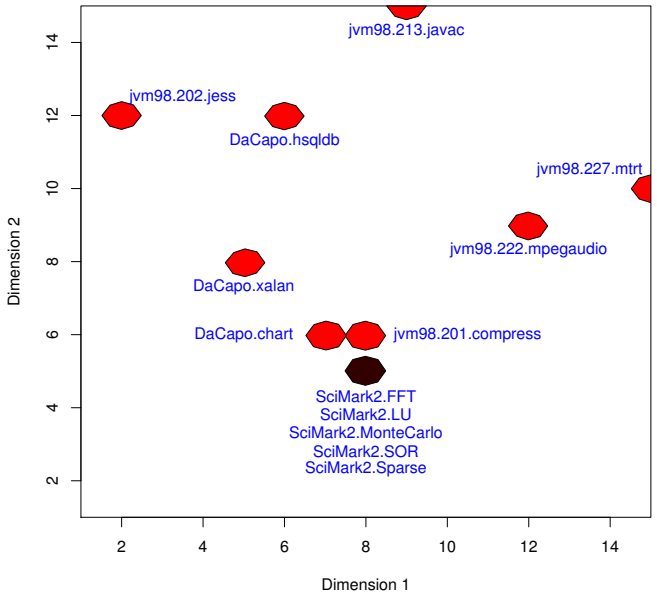


Fig. 7. Workload Distribution When Characterized with Java Method Utilization

Figure 7 shows the results of SOM analysis. It can be seen that the clustering result is quite different from the clustering results based on SAR counters. For example, jess and mtrt

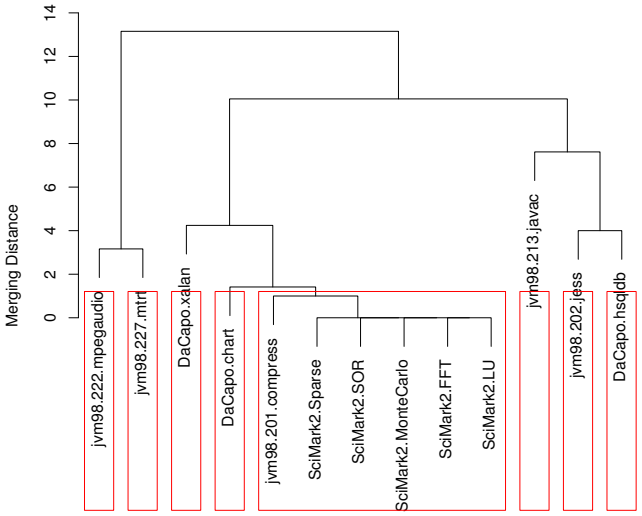


Fig. 8. Clustering Results Based on Java Method Utilization

from SPECjvm98 were fairly similar when characterized by SAR counters. In this analysis, however, they are located on the two extremes. DaCapo benchmark's chart and xalan also show improved separation. On the contrary, SciMark2 workloads form an even denser cluster. This shows that SciMark2 workloads differ from other workloads not only in their execution time behavior, but also in their source code characteristics. This is due to the fact that SciMark2 workloads heavily rely on self contained math libraries. Since SciMark2 workloads map to the same single cell, they appear in a single cluster no matter which merging distance is chosen. This behavior is shown in Figure 8.

	A	B	ratio(=A/B)
2 Clusters	2.76	2.30	1.20
3 Clusters	2.65	2.31	1.15
4 Clusters	2.82	2.36	1.20
5 Clusters	2.59	2.38	1.09
6 Clusters	2.57	2.46	1.05
7 Clusters	2.75	2.52	1.09
8 Clusters	2.89	2.52	1.15
Geometric Mean	2.10	1.94	1.08

TABLE VI
HIERARCHICAL GEOMETRIC MEAN BASED ON JAVA METHOD UTILIZATION

Table VI summarize the hierarchical geometric mean scores based on this clustering result.

It is by now clear that workload clustering heavily depends on how the workloads are characterized. By employing other microarchitecture independent workload features, e.g., instruction mix, memory stride, etc. [5], [6], we expect the workload clusters to appear similar over a variety of machines.

VI. Related Work

Studies in workload clustering behavior in a benchmark suite are well described in [5], [10], [11], [12]. Cluster information has been usually applied for benchmark subsetting. Especially, in [10], [11] the authors apply the cluster information to subset a benchmark suite while preserving the inherent benchmark characteristics intact. Different from these studies, our study apply the workload cluster information to provide insight on benchmark behavior for improving the workload selection process and the scoring metric in a quantitative manner.

These studies usually incorporate PCA as a dimension reduction technique. Compared to the usual PCA approach, our approach utilizes SOM which effectively performs non-linear regression over high dimensional data without losing the information contained in the original domain. This characteristic was particularly suitable for our bit-vectorized Java method utilization information. Moreover, compared to PCA, SOM has significantly better visualization capabilities.

Benchmark suite scoring metric has been a controversial topic. Good consensus on the debate can be found in [19], [20], [21]. Simply put, it is a war between arithmetic mean and geometric mean. Our scoring metrics do not stand on either side; rather, they improve upon both.

VII. Conclusion

In this paper we focus on workload redundancy problem in a benchmark suite. Redundancy in workloads of a benchmark suite is inevitable as a new benchmark suite is often contributed by several parties; sometimes they are constructed by simply combining several benchmark suites into one, which substantially increases the likelihood of redundancy. Such redundancy, in particular the artificial redundancy, renders the benchmark scores biased, making the score of a suite susceptible to malicious tweaks and generating misrepresented performance indices. To address this problem, we propose a set of new scoring methods, the *hierarchical means* that incorporate workload cluster information in overall score calculation. In addition, one can use our methods to characterize and evaluate a new benchmark suite in a quantitative, objective manner.

As a case study, we apply our proposed scoring methods over a hypothetical Java benchmark suite, which attempts to model the upcoming new SPECjvm benchmark suite. Throughout the case study, we show that workload clustering heavily depends on how the workloads are characterized. Nonetheless, the workloads from SciMark2 benchmark suite consistently coagulate into an exclusive cluster, regardless of which characterization method is applied. These workloads significantly increase the workload redundancy in a benchmark suite. We expect that our scoring method can offer a quantitative and objective analysis, providing insightful information for improving benchmark selection process as well as effectively reducing the negative effects from the redundant workloads wherever needed.

Acknowledgments

This research was initiated under the Software and Solutions Group Research Intern Program at Intel Corp. We thank Dr. Shih-wei Liao for his interest and support in this research. He first suggested the use of the term 'Hierarchical Geometric Mean.' We also thank Ben Matasar and Yanping Wang for initial comments and inputs on this research.

References

[1] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary, "MineBench: A benchmark suite for data mining workloads," in *Proceedings of the 2006 IEEE International Symposium on Workload Characterization*, October 2006, pp. 182–188.

[2] O. Altun, N. Dursunoglu, and M. Amasyali, "Clustering application benchmark," in *Proceedings of the 2006 IEEE International Symposium on Workload Characterization*, October 2006, pp. 178–181.

[3] R. Pozo and B. Miller, "SciMark, a web-based benchmark for numerical computing in Java," *In Summary of Activities for Fiscal Year 2005, Mathematical and Computational Sciences Division, NIST Information Technology Laboratory*, pp. 46–47, 2005.

[4] *LAPACK/ScaLAPACK Development Forum*, http://icl.cs.utk.edu/lapack-forum/archives/lapack/msg00190.html.

[5] L. Eeckhout, J. Sampson, and B. Calder, "Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation," in *Proceedings of the 2005 IEEE International Symposium on Workload Characterization*, 2005, pp. 2–12.

[6] K. Hoste and L. Eeckhout, "Comparing benchmarks using key microarchitecture-independent characteristics," in *Proceedings of the 2006 IEEE International Symposium on Workload Characterization*, October 2006, pp. 83–92.

[7] S. Haykin, *Neural Networks: A Comprehensive Foundation, 2nd Ed.* Prentice Hall, July 1998.

[8] T. Kohonen, *Self-Organizing Maps, 3rd Ed.* Springer, May 2006.

[9] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification, 2nd Ed.* Wiley-Interscience, October 2000.

[10] H. Vandierendonck and K. D. Bosschere, "Experiments with subsetting benchmark suites," in *Proceedings of the IEEE 7th Annual Workshop on Workload Characterization*, October 2004, pp. 55–62.

[11] J. J. Yi, R. Sendag, L. Eeckhout, A. Joshi, D. J. Lilja, and L. K. John, "Evaluating benchmark subsetting approaches," in *Proceedings of the 2006 IEEE International Symposium on Workload Characterization*, October 2006, pp. 93–104.

[12] A. Phansalkar, A. Joshi, and L. K. John, "Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite," in *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*. New York, NY, USA: ACM Press, 2007, pp. 412–423.

[13] "SPEC JVM98 benchmarks." Standard Performance Evaluation Corporation, http://www.spec.org/jvm98/.

[14] S. M. Blackburn, R. Garner, C. Hoffman, A. M. Khan, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann, "The DaCapo benchmarks: Java benchmarking development and analysis," in *OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications*. New York, NY, USA: ACM Press, Oct. 2006.

[15] M. Hauswirth, P. F. Sweeney, A. Diwan, and M. Hind, "Vertical profiling: Understanding the behavior of object-oriented applications," in *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2004, pp. 251–269.

[16] K. Chow, R. Morin, and K. Shiv, "Enterprise Java performance: Best practices," in *Intel Technology Journal, volume 7, issue 1*, February 2003, pp. 32–46.

[17] G. Chen, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "PennBench: A benchmark suite for embedded Java," in *the IEEE 5th Annual Workshop on Workload Characterization*, November 2005.

[18] M. T. Conte, A. R. Trick, J. C. Gyllenhaal, and W.-M. W. Hwu, "A study of code reuse and sharing characteristics of Java applications," in *the IEEE 1st Annual Workshop on Workload Characterization*, November 1998.

[19] J. E. Smith, "Characterizing computer performance with a single number," *Communications of the ACM*, vol. 31, no. 10, pp. 1202–1206, 1988.

[20] J. R. Mashey, "War of the benchmark means: time for a truce," *SIGARCH Computer Architecture News*, vol. 32, no. 4, pp. 1–14, 2004.

[21] L. K. John, "More on finding a single number to indicate overall performance of a benchmark suite," *SIGARCH Computer Architecture News*, vol. 32, no. 1, pp. 3–8, 2004.