Setvectors for Memory Phase Classification

Michael Zwick, Marko Durkovic, Florian Obermeier and Klaus Diepold *

Abstract—**Phase classification has frequently been discussed the recent years as a method to guide scheduling, compiler optimizations and program simulations. In this paper, we propose a new classification method called *setvectors*. We show that the new method outperforms classification accuracy of state of the art methods by approximately 6 to 25 percent, while it has comparable computational complexity to the fastest state of the art methods. As a second contribution, we introduce a new method called *PoEC* (Percentage of Equal Clustering) to objectively compare phase classification techniques.**

Keywords: Computer systems, cache memory, phase classification

1 Introduction

It is well known that the execution of computer programs shows cyclic, reoccuring behavior over time. In one time interval, a program may get stuck waiting for I/O, in another time interval, it may likely stall on branch misprediction or wait for the ALU (Arithmetic Logic Unit) to complete its calculations. These intervals of specific behavior are called *phases*. *Phase classification* is the method that analyzes programs and groups program intervals of similar behavior to equal *phases*. [12, 8, 7, 5]

To identify phases, programs are split into intervals of an fixed amount of instructions. Then, these intervals are analyzed by some method to predict its behavior. Intervals showing similar behavior are grouped together to form a specific class of a *phase*. Therefore a phase is a set of intervals that show similar behavior while discarding temporal adjacency. The phase information of a program can be used to guide scheduling, compiler optimizations, program simulations, etc.

Several phase classification techniques have been proposed the recent years, many of which rely on code metrics such as *basic block vectors* [12] and *dynamic branch counts* [3]. Since code related methods only have low correlation with memory hierarchy behavior, several memory related phase classification methods have been proposed to predict L1 and L2 cache misses, such as *wavelet based phase classification* [5], *activity vectors* [11] and *stack reuse distances* [4].

*Lehrstuhl für Datenverarbeitung, Technische Universität München, Arcisstr. 21, 80333 München, Germany, {zwick, durkovic, f.obermeier, kldi}@tum.de

In this paper, we propose a new method for phase classification to predict L2 cache performance called *setvectors*. On the basis of ten SPEC2006 benchmarks, we show that the mean accuracy of our method outperforms the *activity vector* method by about 6%, the *stack reuse distance* method by about 18% and the *wavelet* based method by about 24%. Further, we introduce a new metric called *PoEC* (Percentage of Equal Clustering) to objectively evaluate different phase classification methods and make them comparable to one another.

The remaining of this paper is organized as follows: Section 2 describes state of the art phase classification techniques, section 3 presents our *setvector* method, section 4 introduces *PoEC* as our methodology to evaluate phase classification accuracy, section 5 presents our results and section 6 concludes the paper.

2 Phase Classification Techniques

In this section, we describe state of the art techniques we compared our *setvector* method to. Since we focus on phase classification for L2 cache performance prediction, we exclusively consider memory based techniques.

All the methods we apply use the same tracefiles comprised of memory references we gatherd using the *pin tool* described in [9], as it has been done in [5]. For cross validation, we use our MCCCSim (Multi Core Cache Contention Simulator) simulator [14] to obtain the hitrates for each interval. Since we initially started the evaluation of our method in comparison to the wavelet technique presented by Huffmire and Sherwood [5], we chose the same interval size of 1 million instructions as they did to make the results better comparable. All methods were evaluated on ten SPEC2006 test benchmarks, each comprised of 512 million instructions.

2.1 Wavelet based Phase Classification

In [5], Huffmire and Sherwood use haar wavelets [13] to perform phase classification.

First, they create 16×16 *matrices* for each interval of 1 million instructions. Therefore, they split each set of 10^6 instructions into 20 subsets of $10^6/20 = 50,000$ instructions forming 20 column vectors. They determine the elements of every such column vector by calculating $m = ((address\%M) \cdot (400/M))$ for each address in the corresponding subset, where '%' is the modulo operator

and $M = 16k$ the modulo size that has been matched to the L1 cache size. By iterating over each address of the 50k instructions, they fill up the rows of the column vectors by summing up the occurences of each m ($0 \leq m < 400$) in a histogram manner. After having calculated each of the 20 400×1 column vectors, they scale the resulting 400×20 matrix to a 16×16 matrix using methods from image scaling.

In a second step, they calculate the *haar wavelet transform* [13] for each 16×16 matrix and weight the coefficients according to [6].

In a third step, they apply the k-means clustering algorithm [10] on the scaled wavelet coefficients and compare the clusters with the hitrates of the corresponding intervals, gained by a cache simulator.

In our implementation of the wavelet technique, we followed Huffmire's and Sherwood's description except for the following: we split each 10^6 instructions in 16 intervals of 64k instructions each to omit the scaling from 20 to 16 columns and use our MCCCSim [14], that is based on Huffmire's and Sherwood's cache simulator anyway. Everything else we implemented as presented in [5].

Since the top-left element of the coefficient matrix corresponds to the mean value of the original matrix, we also clustered all top-left elements of the coefficient matrices and compared the results of the clustering process to the L2 hitrates of the corresponding intervals.

As the wavelet transform seems not an obvious choice of algorithm for this problem, yet it achieved good results shown by Huffmire and Sherwood, we decided to replace the wavelet transformation by a SVD (singular value decomposition) $\mathbf{M} = \mathbf{U\Sigma V^T}$in another experiment to probe if a more general method could find more information in the data matrix. We clustered both columns and rows of $\mathbf{U}$ and $\mathbf{V}$ respectively but could not find any impressive results, as we will show in Section 5.

2.2 Activity Vectors

In [11], Settle et al. propose to use "activity vectors" for Enhanced SMT (simultaneous multi-threaded) job scheduling. The *activity vectors* are used like a phase classification technique to classify program behavior with respect to memory hierarchy performance. The *activity vector* method has been proposed as an "online classification method" that relies on a set of on-chip event counters that count memory accesses to so-called *super sets*.

We implemented the activity vector method in software and applied the same tracefile data that we applied to all our other simulations mentioned in this paper.

To use the activity vector method as a phase classification method, we clustered a) the vectors and b) the length of each vector and compared the clustering results with the L2 cache performance of the corresponding intervals.

In section 5 we show that the activity vector method on average achieves better results than the wavelet method.

2.3 Stack Reuse Distances

In [1], Beyls and D'Hollander propose to use the "stack reuse distance" as a metric for cache behavior. They define the stack reuse distance of a memory access as "the number of accesses to unique addresses made since the last reference to the requested data" [1].

In section 5 we show that, on average, the classification performance of the stack reuse distance method lies between the wavelet and the activity vector method, whereas its calculation takes a huge amount of time.

2.4 Other techniques

Although many other techniques for phase classification have been proposed such as *basic block vectors* [12], local/global stride [8], working set signatures [2], we omitted to compare our *setvector* technique to those methods since it has been shown that they are outperformed by other methods, for example the *wavelet* method [5].

3 Setvector based Phase Classification

In this section, we describe our *setvector* based phase classification method.

The *setvectors* are as easily derived as they are effective: For all addresses of an interval and an n-way set-associative cache, determine the number of addresses with *different key* that are mapped to the same cache set.

That means: Given a L2 cache with 32 bit address length that uses b bits to code the byte selection, s bits to code the selection of the cache set and $k = 32 - s - b$ bits to code the key that has to be compared to the tags stored in the tag RAM, do the following:

- Extract the set number from the address, e.g. by shifting the address k bits to the left and then unsigned-shifting the result $k + b$ bits to the right.
- Extract the key from the address, e.g. by unsigned-shifting the address $s + b$ bits to the right.
- In the list for the given set, determine whether the given key is already present.
- If the key is already present, do nothing and process the next address.
- If the key is not in the list yet, add the key and increase the counter that corresponds to that set.

More formally written: Starting with a tracefile

$$\mathbf{T} = \{a_i | 1 \leq i \leq t\}$$

made up of memory addresses $a_1 ... a_t$ that are grouped into intervals of a fixed amount of instructions, we split the tracefile into a set of *access vectors* $\mathbf{a_i}$, each representing an interval of several a_i:

$$\mathbf{T} = \begin{bmatrix} \mathbf{a}_1 & \dots & \mathbf{a}_c \end{bmatrix}$$

Now, for each $\mathbf{a}_i$ and caches with a row size of r byte and a way size of w byte, we derive the set vector

$$\mathbf{s}_i = \begin{bmatrix} s_1 & \dots & s_s \end{bmatrix}^T$$

by

$$S_i \leftarrow \left\{ \frac{a}{r} |\ a \in \mathbf{a}_i, \frac{(a\%w) - ((a\%w)\%r)}{r} = i, \frac{a}{r} \notin S_i \right\}$$

$$s_i = min(2^n - 1, \sum_{a/r \in S_i} 1)$$

with '%' representing the modulo operator and n the maximum number of bits allowed to code the set vector elements, if there should be such a size constraint.

This way, each element of a set vector contains for each corresponding interval the number of addresses that belong to the same cache set, but have a different cache-key – saturated by n, the number of bits at most to be spent for each vector element.

Due to this composition, the setvectors directly represent cache set saturation, a measure that is highly correlated with cache misses.

In section 5, we show that on average, the *setvector* method outperforms all methods mentioned in section 2.

4 Metrics to Compare Phase Classification Techniques

In [8], Lau et al. define the *Coefficient of Variation* (CoV) as a metric to measure the effectiveness of phase classification techniques. CoV measures the standard deviation as percentage of the average and can be calculated by

$$CoV = \sum_{i=1}^{phases} \frac{\frac{\sigma_i}{average_i} \cdot intervals_i}{total\ intervals}. \quad (1)$$

Huffmire and Sherwood adapt this metric by omitting the division by the average, resulting in the *weighted standard deviation*

$$\sigma_{weighted} = \sum_{i=1}^{phases} \frac{\sigma_i \cdot intervals_i}{total\ intervals}. \quad (2)$$

Being derived from standard deviation, both CoV and $\sigma_{weighted}$ denote better clustering performance by smaller values. However, the CoV metric ($\sigma_{weighted}$ as well) may describe the standard deviation of the L2 cache performance in each phase, but not the correlation between L2 cache performance and the different phases, what should be the key evaluation method for phase classification. Therefore, we developed the *PoEC* (Percentage of Equal Clustering) metric that can be calculated as follows:

Consider the cluster vector $\boldsymbol{\gamma}$ as a vector that holds, for each index, the phase of the corresponding 16×16 scaled matrix. In a first step, we sort the elements of the cluster vector $\boldsymbol{\gamma}$ according to its phases, such that $\forall_{i \in 1..indeces}$: $\gamma_i \leq \gamma_{i+1}$.

In a second step, we calculate the percentage of equal clustering (PoEC) by

$$PoEC = 2 \cdot \left[min\left(\frac{\sum_{i=1}^{indeces} (\gamma_{h,i} == \gamma_{x,i})}{indeces}, 0,5 \right) - 0,5 \right] \quad (3)$$

This way, high correlation between L2 cache performance and the cluster vectors result in PoEC values near 1 and low correlation corresponds to values near 0, with $0 \leq PoEC \leq 1$.

Figure 2 shows the difference between those metrics by clustering some SPEC2006 benchmarks in two phases ("good ones" and "bad ones") using the wavelet method and plotting the phases (each ring corresponds to one interval) against the L2 hitrate of the corresponding intervals. As L2 cache hitrates of the same magnitude should be classified into the same class, a good clustering is achieved if one class contains higher hitrates and the other contains lower hitrates, as it is the case for the "milc", "soplex", "lbm" and "bzip2" benchmarks.

In Figure 2a), we calculated the CoV value according to formula 1 for each benchmark and arranged the plots according to their CoV value. While analyzing Figure 2a), one can observe the following: There are benchmarks that achieve good clustering, such as "soplex", "milc", "bizp2" and "lbm". And there are benchmarks that do not cluster well at all, such as "hmmer", "libquantum", "gobmk". But the point is: The clustering quality *does not* fit the CoV value the plots are arranged by. Although not plotted in this place, the $\sigma_{weighted}$ metric shows similar behavior.

In Figure 2b), we calculated the $PoEC$ value according to formula 3 and arranged the plots according to their

PoEC value. Although the clustering is the same, this time the clustering quality *does fit* the PoEC value the plots are arranged by.

Therefore we decided to omit both $\sigma_{weighted}$ and CoV and to perform our evaluation using our new *PoEC* metric.

5 Results

In this section we discuss the results gathered from our simulations.

5.1 Classification Accuracy

In Figure 1, we plotted the *PoEC* values for each of the mentioned methods for ten SPEC2006 benchmarks. For the *activity vectors*, we clustered a) the vector itself and b) the magnitude |activity vector| of the activity vector. For the *haar wavelet* method, we clustered both the scaled matrix and the left-top matrix element "haar wav.[0][0]". For the *setvectors*, we clustered the magnitude of the setvectors; for the *stack reuse distances*, we clustered the stack distance vector, and for the *SVD scaled matrices* the results shown originate from the column space of $\mathbf{U}$. Column/Rowspace of $\mathbf{V}$ didn't achieve any better results. *PoEC* values near 1 indicate good classification performance, *PoEC* values near 0 indicate poor classification performance. The benchmark *mcf* for example shows superior performance for the *activity vectors* method, the *wavelet* method and the *setvector* approach and poor classification performance for the *stack distance* and *SVD* method.

Figure 3 depicts the mean classification performance of each method averaged on the ten SPEC2006 benchmarks. The *setvector* approach outperforms all other methods and achieves about 6% better classification performance than the next best method, the *activity vectors* method, that performs just slighly better than the *stack reuse distance* method. While the *wavelet* method still shows fair results, the singular value decomposition of the scaled matrix has apparantely not been a good idea at all.

5.2 Computational Performance

The computational performance of the mentioned methods can not easily be compared in this paper, because the methods have been implemented in different programming languages. The *setvector* and *activity vector* methods have completely been implemented in slow Ruby 1.8.6, the *wavelet* and *SVD* methods have been implemented in a mixture of Ruby and R and the *stack reuse distances* method has completely been implemented in C for performance reasons.

The calculation of the scaled matrices used by the *wavelet* method has been timed for $14\mu s$ per simulated instruction. The calculation of wavelet coefficients in Ruby was

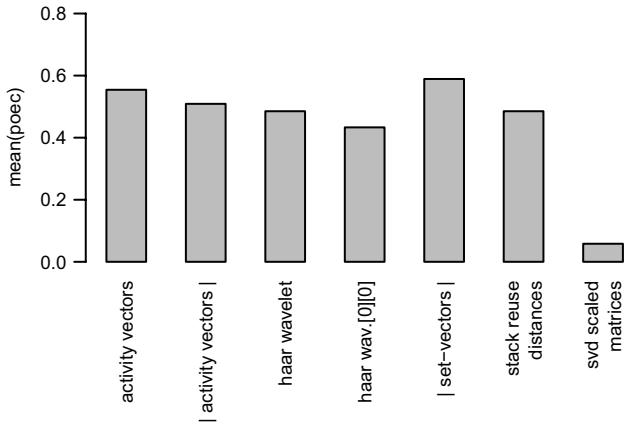

Figure 3: Mean PoEC classification performance

about $15ns$, the calculation of the kmeans algorithm in R was about $2.5ns$, including data exchange over the Ruby-R-bridge in both cases.

The *setvector* method suffered as the *wavelet* method from the slow Ruby interpreter and took about $3.97\mu s$ per instruction. The magnitude calculation and the clustering algorithm for the setvectors finished in less than $0.5ns$. The better clustering performance originates from using scalars (vector magnitues) rather than vectors of dimensionality $16 \cdot 16 = 256$, as it has been done in the *wavelet* technique.

The by far worst computational performance was demonstrated by the *stack reuse distance* method, that also took about $3.97\mu s$ of time per simulated instruction, but this time the method was implemented in C-code to finish the simulation in just some days instead of months.

Comparing the ruby wavelet implementation (about $14,0175\mu s$ per simulated instruction) to the execution time measued by [5] (about $10ns$ per simulated instruction), allows the *setvector* method to be estimated for about $5-10ns$ per instruction when implemented on a C code basis; i.e. the *setvector* method has about the same magnitude of calculation complexity as the other mentioned methods, except the *stack reuse distance method*. All values have been averaged over several executions or files.

6 Conclusions

In this paper we introduced a new method for phase classification called *setvectors*. The method is similar to the *activity vectors* method proposed by [11], but it differs in the way the vectors are obtained. While Settle et al. just count accesses to *super sets*, we calculate the number of accesses to a set that reference a different key. We show that the proposed method outperforms state of the art methods with respect to classification accuracy by approximately 6 to 25 percent, having about the

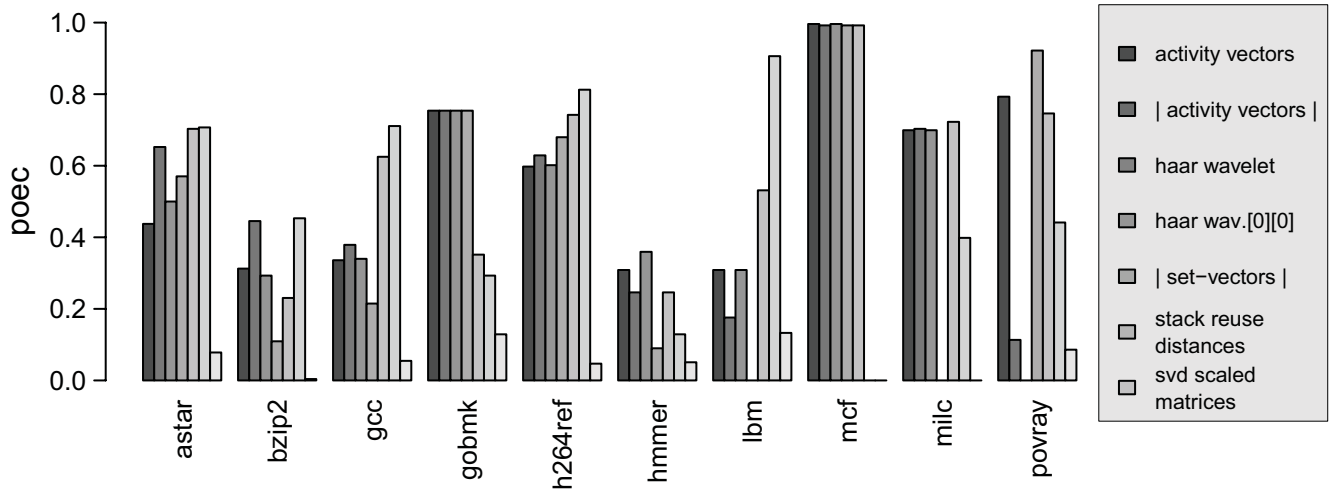

Figure 1: Accuracy of phase classification techniques measured by the PoEC metric

same computational constraints. As a second contribution, we introduced the *PoEC* metric that can be used to objectively evaluate phase classification methods in a more intuitive way than the known metrics CoV and $\sigma_{weighted}$. Although we proved the better performance of the *PoEC* method compared to the $CoV/\sigma_{weighted}$ method just qualitatively "by inspection", it obviously is a more reasonable approach.

References

[1] K. Beyls and E. H. D'Hollander. Reuse distance as a metric for cache behavior. 2004.

[2] A. S. Dhodapkar and J. E. Smith. Managing multi-configuration hardware via dynamic working set analysis. In *International Symposium on Computer Architecture (ISCA'02)*, May 2002.

[3] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and predicting program behavior and its variability. In *12th International Conference on Parallel Architectures and Compilation TEchniques (PACT'03)*, 2003.

[4] C. Fang, S. Carr, S. Önder, and Z. Wang. Reuse-distance-based miss-rate prediction on a per instruction basis. In *Proceedings of the 2004 workshop on Memory system performance*, 2004.

[5] T. Huffmire and T. Sherwood. Wavelet-based phase classification. In *Parallel Architectures and Compilation Techniques (PACT'06)*, September 2006.

[6] C. Jacobs, A. Finkelstein, and D. Salesin. Fast multiresolution image querying. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 1995.

[7] J. Lau, J. Sampson, E. Perelman, G. Hamerly, and B. Calder. The strong correlation between code signatures and performance. In *IEEE International Symposium on Performance Analysis of Systems and Software*. March 2005.

[8] J. Lau, S. Schoenmackers, and B. Calder. Structures for phase classification. 2004.

[9] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klausner, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Programming Language Design and Implementation, ACM*, 2005.

[10] J. MacQueen. Some methods for classification and analysis of multivariate obervations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[11] A. Settle, J. L. Kihm, A. Janiszewski, and D. A. Connors. Architectural support for enhanced smt job scheduling. *Proceedings of the 13th International Conference of Parallel Architectures and Compilation Techniques*, September 2004.

[12] Sherwood, Perelman, Hamerly, and Calder. Automatically characterizing large scale program behavior. *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, 2002.

[13] E. Stollnitz, T. DeRose, and D. Salesin. Wavelets for computer graphics: A primer. In *IEEE Computer Graphics and Applications*.

[14] M. Zwick, M. Durkovic, F. Obermeier, W. Bamberger, and D. K. Mcccsim - a highly configurable multi core cache contention simulator. Technical report, Technische Universität München, https://mediatum2.ub.tum.de/doc/802638/802638.pdf, 2009.

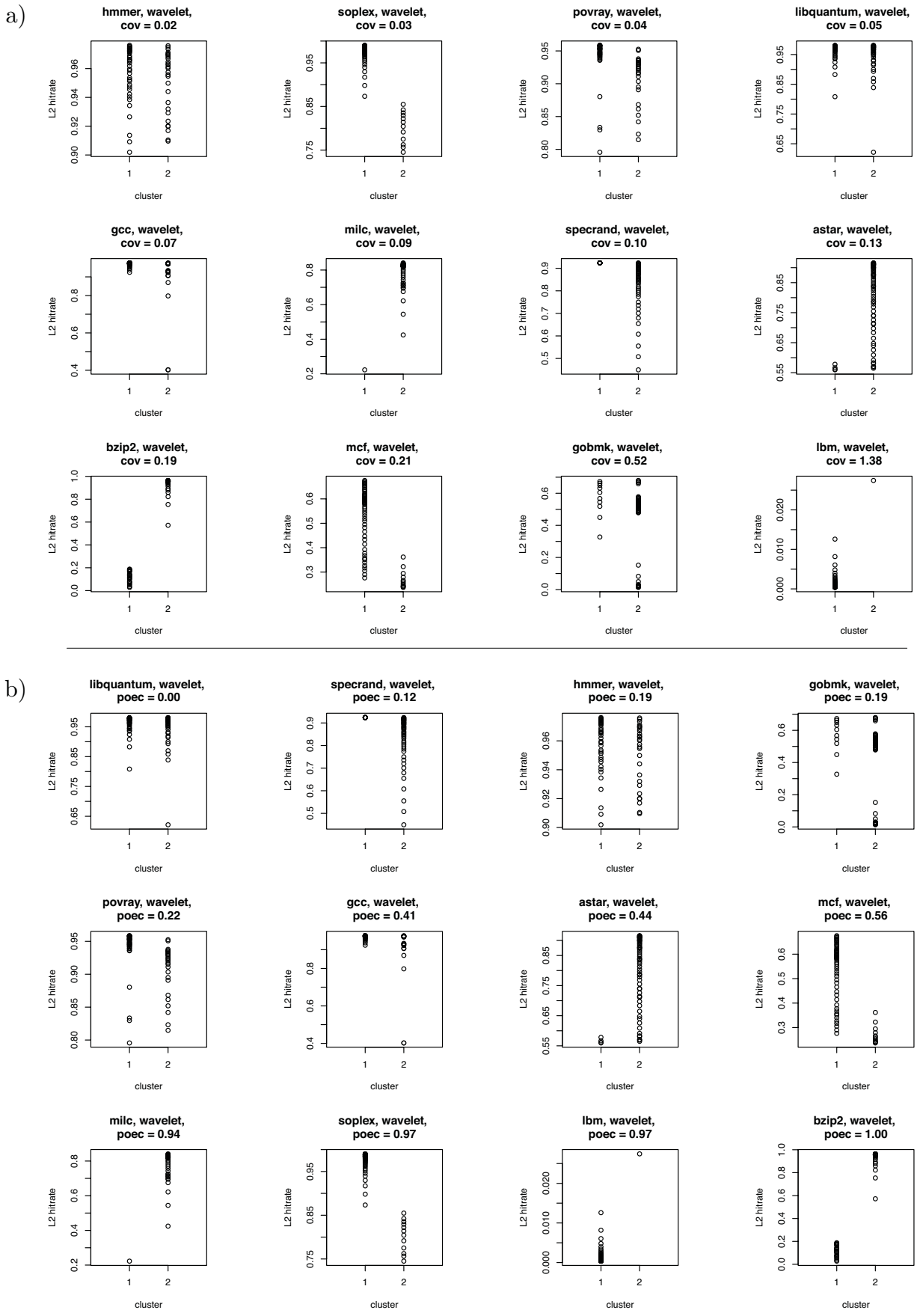

Figure 2: Comparison of CoV and $PoEC$ metric