Front. Comput. Sci., 2020, 14(1): 21–41
https://doi.org/10.1007/s11704-018-7319-2

RESEARCH ARTICLE

# Computer comparisons in the presence of performance variation

**Samuel IRVING**[1,2], **Bin LI**[1], **Shaoming CHEN**[1], **Lu PENG**[1], **Weihua ZHANG** (✉)[2,3,4], **Lide DUAN**[5]

1 Louisiana State University, Baton Rouge, LA 70803, USA
2 Shanghai Institute of Intelligent Electronics & Systems, Shanghai 201203, China
3 Software School, Fudan University, Shanghai 201203, China
4 Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 200433, China
5 University of Texas at San Antonio, San Antonio, TX 78249, USA

**Abstract** Performance variability, stemming from non-deterministic hardware and software behaviors or deterministic behaviors such as measurement bias, is a well-known phenomenon of computer systems which increases the difficulty of comparing computer performance metrics and is slated to become even more of a concern as interest in Big Data analytic increases. Conventional methods use various measures (such as geometric mean) to quantify the performance of different benchmarks to compare computers without considering this variability which may lead to wrong conclusions. In this paper, we propose three resampling methods for performance evaluation and comparison: a randomization test for a general performance comparison between two computers, bootstrapping confidence estimation, and an empirical distribution and five-number-summary for performance evaluation. The results show that for both PARSEC and high-variance BigDataBench benchmarks 1) the randomization test substantially improves our chance to identify the difference between performance comparisons when the difference is not large; 2) bootstrapping confidence estimation provides an accurate confidence interval for the performance comparison measure (e.g., ratio of geometric means); and 3) when the difference is very small, a single test is often not enough to reveal the nature of the computer performance due to the variability of computer systems. We further propose using empirical distribution to evaluate computer performance and a five-number-summary to summarize computer performance. We use published SPEC 2006 results to investigate the sources of performance variation by predicting performance and relative variation for 8,236 machines. We achieve a correlation of predicted performances of 0.992 and a correlation of predicted and measured relative variation of 0.5. Finally, we propose the utilization of a novel biplotting technique to visualize the effectiveness of benchmarks and cluster machines by behavior. We illustrate the results and conclusion through detailed Monte Carlo simulation studies and real examples.

**Keywords** performance of systems, variation, performance attributes, measurement, evaluation, modeling, simulation of multiple-processor systems, experimental design, Big Data

## 1 Introduction

Traditionally, computer researchers have used the geometric mean (GM) of performance ratios of two computers running a set of selected benchmarks to compare their relative performances. This approach, however, is limited by the variability of computer systems which stems from non-deterministic hardware and software behaviors [1, 2], or deterministic behaviors such as measurement bias [3]. The situation is exacerbated by increasingly complicated architectures and programs, both of which can negatively impact performance reproducibility [4]. Wrong conclusions could be drawn if variability is not handled correctly. Using a simple geometric

Received September 9, 2017; accepted January 15, 2018

E-mail: zhangweihua@fudan.edu.cn

mean cannot describe the performance variability of computers [5].

Recently, computer architects have been seeking advanced statistical inferential tools to address the problem of performance comparisons of computers. The two common statistical approaches of comparing two populations (e.g., two computers) are the hypothesis test and confidence interval estimation. As we know, most of the parametric tests such as *t*-tests require population distribution normality [6]. Unfortunately, computer performance measurements are often not normally distributed but either skewed or multimodal. Figure 1 shows 400 measurements of execution time from SPEC2006 benchmarks running on a commodity computer (Intel Core i7 CPU 960@3.20GHz, 1 processor with 4 cores, 10GB DDR3 RAM(1333 MHz)). We can see that the distributions of performance measures for the benchmarks are non-normal; benchmarks "gcc" and "mcf" are skewed to the right, while "bzip2" is multimodal. This non-normality observation was first observed by Chen et al. [5, 7] who tackled with a non-parametric statistics method named hierarchical performance testing (HPT).

In this paper, we propose three statistical resampling methods [8] to evaluate and compare computer performance. The first is a randomization test used to compare the performance between two computers; the second is a bootstrapping confidence interval method for estimating the comparative performance measurement, i.e., speedup, through a range; and the third is an empirical distribution method to evaluate the distributional properties of computer performance. The basic idea of resampling methods, as the name implies, is to resample the data iteratively, in a manner that is consistent with certain conditions (e.g., the general performance of two computers is equal.).

Specifically, we first resample the data according to the purpose of each method. Second, for each iteration, we calculate the statistic of interest, such as the ratio of geometric means between two computers. Third, we repeat the previous two steps a number of times. Then the distribution of the calculated statistic is used as an approximation of the underlying distribution of the statistic under the assumed condition. Hence, the resampling methods set us free from the need for normal data or large samples so that Central Limit Theorem can be applied [9]. Note that the proposed three methods all follow the three steps described above. However, the resampling and calculating steps within each iteration are different according to the individual purpose for each method.

In summary, the main contributions of this paper can be listed as follows:

First, we propose and implement a randomization test [10] for testing the performances of two computers, which provides an accurate estimate of the confidence of a comparison when the performances of two computers are close to each other.

Second, we propose and implement a bootstrapping-based confidence interval estimation method [11] to estimate the confidence interval of the ratio of geometric means between two computers.

Third, as a generic framework, the proposed method can directly be applied to arithmetic and harmonic means. We demonstrate that the arithmetic mean is very sensitive to outliers while geometric and harmonic means are much more stable.

Fourth, we point out that a single test is not enough to reveal the nature of the computer performance in some cases due to the variability of computer systems. Hence, we suggest using empirical distribution to evaluate computer performance and use five-number-summary to summarize the computer performance.

Fifth, we investigate the source of performance variation by predicting the performance and relative variation of machines running the SPEC 2006 benchmark suite using published hardware descriptions and environment variables.

Sixth, we demonstrate the effectiveness of the proposed sampling methods on Big Data benchmarks [12] which have more variation behaviors than traditional CPU benchmarks like SPEC or PARSEC.

Finally, we use a Biplot visualization tool [13] for computer performance comparisons which can visualize the

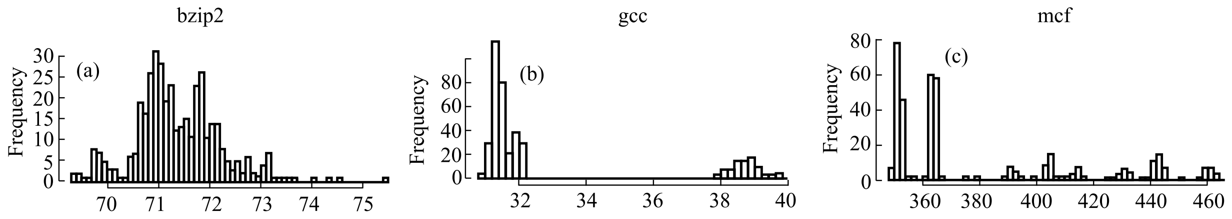


**Fig. 1** Histograms of execution times for three SPEC benchmarks from 400 repeated runs of each benchmark on the commodity computer. (a) Execution time of bzip2; (b) execution time of gcc; (c) execution time of mcf

projections of high-dimensional data onto a low-dimensional space through principal component.

## 2 Motivating example

In this section, we show an example of comparing two computers based on $t$-test and the proposed resampling methods. Table 1 lists the configurations of the computers. The data is available on SPEC website. Figure 2 shows the empirical distributions of geometric mean for two computers. The horizontal axis shows the SPEC ratio. The blue dash line is the empirical distribution of geometric means for the NovaScale computer, while the red solid line is the one from IBM. The vertical dash line shows the geometric mean from the raw data. The basic idea of using an empirical distribution is to see the distribution of a statistic (e.g., geometric mean of computer performance). We can see many useful distributional properties from the empirical distribution, such as the center, mode, variation, and range of the statistic. The details of empirical distribution are described in Section 5. From Fig. 2, although the two distributions overlap, the geometric mean of computer A (red solid curve) is well above that of computer B (blue dash curve). As shown in Table 2, the $t$-test does not detect the difference between two computers while the randomization test does. This implies that the randomization test is more powerful at detecting the difference even when there is an overlap between two distributions. The bootstrap interval also shows the ratio of geometric means is significantly below one (blue dashed curve against red solid curve) which implies that computer A runs faster than computer B.

**Table 1** Configurations of the two computers in Fig. 2

| | Configurations |
|---|---|
| Middle (blue dashed line) | NovaScale T860 F2 (Intel Xeon E5645, 2.40 GHz) |
| Middle (red solid line) | IBM System x3400 M3 (Intel Xefon E5649) |

**Table 2** Test results for the example in Fig. 2

| $T$ test $p$-value | Randomization test $p$-value | 95% Bootstrapping |
|---|---|---|
| 0.117 | 0.016 | [0.974, 0.997] |

## 3 Statistical performance comparison via randomization test

Statistical inference is based on the sampling distributions of sample statistics which answers the question: “if we recollect the data, what will the statistic be?” A sampling distribution of a statistic (e.g., geometric mean) can be well approximated by taking random samples from the population. Traditional parametric tests assume the sampling distribution has a particular form such as a normal distribution. If the distributional assumption is not satisfied, commonly there are no theoretical justifications or results available. On the other hand, the great advantage of resampling is that it often works even when there is no theoretical adjustment available. The basic idea of the randomization test [10] is as follows: in order to estimate the $p$-value (i.e., 1- confidence) for a test, we first estimate the sampling distribution of the test statistic given the null hypothesis is true. This is accomplished by resampling the data in a manner that is consistent with the null hypothesis. Therefore, after resampling many times, we can build up a distribution (called an empirical distribution) which approximates the sampling distribution of the statistic that we are interested in. Thus, we can estimate the $p$-value based on the empirical distribution.

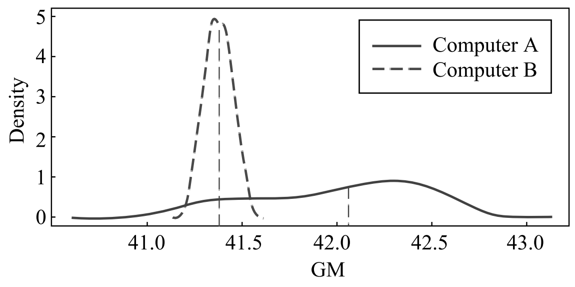


**Fig. 2** Density plots of the empirical distributions for the two computers. The dotted lines are the geometric means

Suppose we have two computers A and B to compare over a benchmark suite consisting of n benchmarks. For each computer, we ran the benchmarks m times and denote the performance scores of A and B at their $j$th runs of the $i$th benchmark as $a_{i,j}$ and $b_{i,j}$ respectively. The hypotheses are specified below.

**Null hypothesis**: the general performance of A and B over $n$ benchmarks are equivalent.

**Alternative hypothesis**: we will use only one of the following three as our alternative hypothesis.

$H_{1a}$: the general performance of A is better than that of B.

$H_{1b}$: the general performance of B is better than that of A.

$H_{1c}$: the general performance of A is not the same as that of B.

We proposed the randomization test as follows:

1) For each benchmark $i$ ($i = 1, 2, \ldots, n$), we combine all the $m$ performance scores from A and B into one list respectively.

2) We randomly permute the list, for each benchmark, and

assign the first m scores to computer A and the other m to B for the $i$th benchmark.

3) Calculate the ratio of the geometric mean of the performance scores for computer A and B over n benchmarks.

4) Repeat step 1–3 $M$ times (M is usually a large number, e.g., 500), so we have $M$ geometric mean ratios, denote as FM (i.e., the empirical distribution of geometric mean ratios under the null hypothesis) from $M$ repetitions.

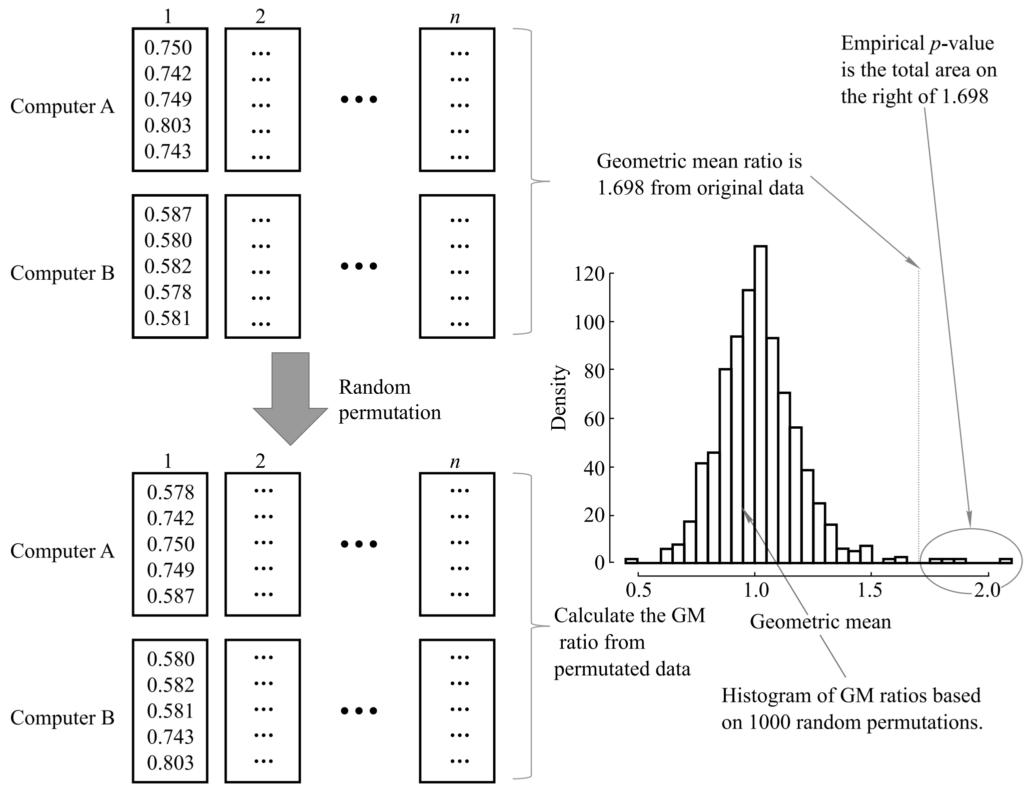
5) Calculate $g_{A|B}$, the ratio of the geometric mean of the performance scores for computer A and B over n benchmarks on the original data. Then we calculate an empirical $p$-value based on $F_M$ and the alternative hypothesis as follows. If we use $H_{1a}$, then the empirical $p$-value is the proportion of $F_M$ that is greater than or equal to $g_{A|B}$. If $H_{1b}$ is selected, then the empirical $p$-value is the proportion of $F_M$ that is less than or equal to $g_{A|B}$. If we use $H_{1c}$, then the empirical $p$-value is the twice of the smaller empirical $p$-value from $H_{1a}$ and $H_{1b}$.

Figure 3 illustrates the proposed randomization test under the alternative $H_{1a}$. Note that the randomization test described above uses the geometric mean to evaluate the computer performance. However, the proposed method can be easily modified to adopt other measures such as harmonic and arithmetic mean.

## 4 Confidence interval estimation by boostrapping

Due to the performance variability, the comparative performance measure, such as the ratio of geometric means and speedups, between two computers varies on different measurements. Hence, presenting a single numeric estimate cannot describe the amount of uncertainty due to the performance variability. The basic idea of a confidence interval (CI) is to provide an interval estimate (which consists of a lower limit and an upper limit) on the statistic with some predetermined confidence level, instead of giving a single estimate. The interpretation of a confidence interval is based on recollecting the data or repeating the experiment.

Bootstrapping [11] is a commonly used statistical technique which quantifies the variability of a statistic, e.g., estimate a 95% confidence interval of a statistic or its standard deviation, which are not yet available in theory [14]. The basic idea of bootstrapping is to use the sample as an approximation of the underlying population distribution, which is unknown, and resample the data with replacement (note that each observation can be sampled more than once). We proposed the following bootstrapping method to estimate the



1
2
n
Computer A
0.750
0.742
0.749
0.803
0.743
Computer B
0.587
0.580
0.582
0.578
0.581
Random permutation
Computer A
0.578
0.742
0.750
0.749
0.587
Computer B
0.580
0.582
0.581
0.743
0.803
Empirical $p$-value is the total area on the right of 1.698
Geometric mean ratio is 1.698 from original data
Density
Geometric mean
Calculate the GM ratio from permutated data
Histogram of GM ratios based on 1000 random permutations.

**Fig. 3** Illustration of the proposed randomization test

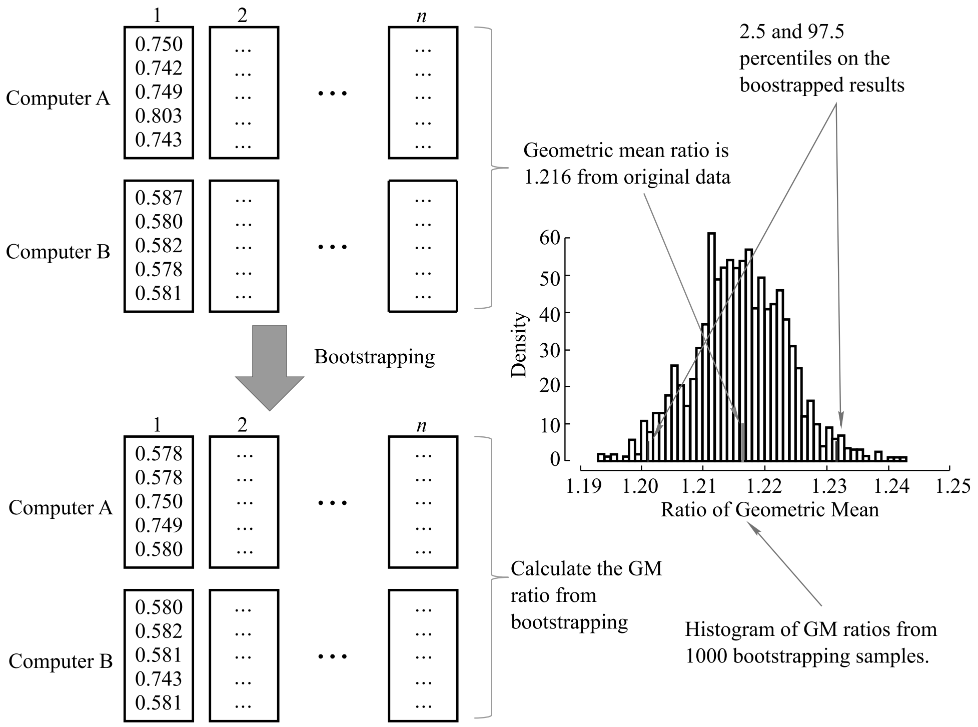


**Fig. 4** Illustration of proposed bootstrapping confidence interval estimation

ratio of the geometric mean of the performance scores from two computers.

1) For each benchmark $i$ ($i = 1, 2, \ldots, n$), we combine all the $m$ execution times from computer A and B into one list respectively.

2) We randomly sample the list with replacement for each benchmark, and assign the first m scores to computer A and the other m to B for the $i$th benchmark.

3) Calculate the ratio of the geometric mean of the execution times for computer A and B over n benchmarks.

4) Repeat step 1–3 $T$ times (T is usually a large number, e.g., 500), so we have $T$ geometric mean ratios, denote as $H_T$ from $T$ repetitions. Let $H_T^{\alpha/2}$and $H_T^{1-\alpha/2}$ be the $\alpha$ and 1-$\alpha/2$ percentiles of $H_T$ respectively. Then, a two-sided (1-$\alpha$)×100% bootstrap confidence interval is $\left[H_T^{\alpha/2}, H_T^{1-\alpha/2}\right]$. A one-sided (1-$\alpha$)×100% bootstrap confidence interval can be either $\left[H_T^{\alpha}, +\infty\right]$ or $\left[-\infty, H_T^{1-\alpha}\right]$. The former one-sided confidence interval is explained as the ratio of GMs between computer A and B is at least $H_T^{\alpha}$ with confidence (1-$\alpha$)×100%, while the latter as the ratio of GMs between computer A and B is at most $H_T^{1-\alpha}$ with confidence (1-$\alpha$)×100%. Figure 4 illustrates the proposed bootstrapping method using an example.

## 5 Empirical distribution and five-number summary

Although the proposed randomization test demonstrates more precise than conventional $t$-test, when two computers show overlapped distributions and close geometric mean, a single test such as $t$-test and randomization test cannot identify their differences. Figure 5 shows three pairs of computers listed in Table 3. The $p$-values of both $t$-test and randomization test for all the three pairs are close to 1.0. For example, the $p$-values are 0.941 and 0.856 for $t$-test and randomization test respectively for the two computers shown in Fig. 5(a). Similar situations also apply to the pairs in Figs. 5(b) and 5(c). This indicates no performance differences could be identified by a single test. On the other hand, an insignificant test result does not necessarily mean the two computers have the same performance. For example, in Fig. 5 we see that all three computers depicted by red solid lines have slightly higher geometric means than their competitors, but their performances are less consistent than the ones shown by blue dashed lines. Therefore in comparing performance, we need to consider the system variation effect especially when the means are close.

Hence, we suggest using the empirical distribution of the geometric mean and its five-number-summary to describe of performance for a computer as follows:

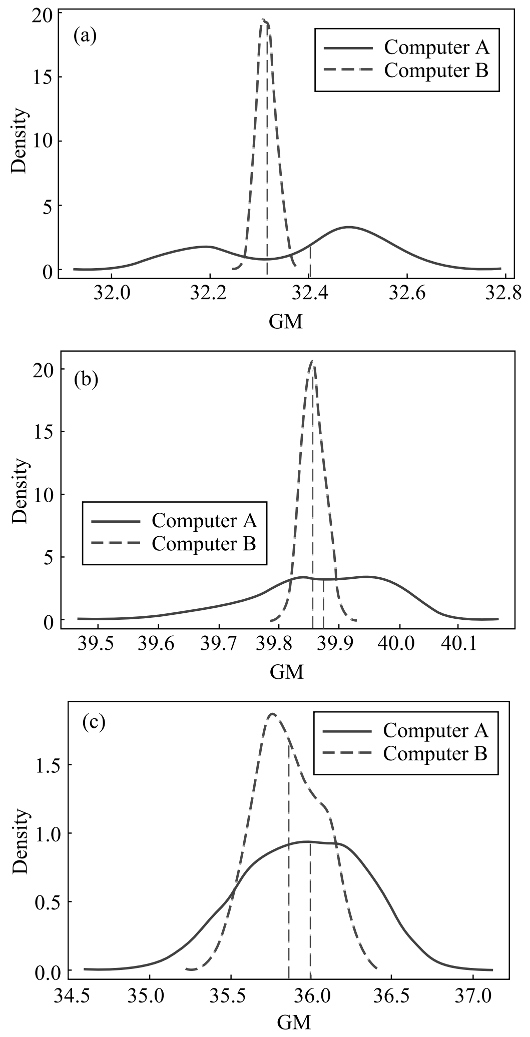


**Fig. 5** Density plots of the empirical distributions for three pairs of computers. The dot lines are the geometric means. (a) Density plots for pair one; (b) density plots for pair two; (c) density plots for pair three

**Table 3** Configurations of three pairs of computers in Fig. 5

| | Configurations |
|---|---|
| Figure 5(a) (blue dashed line) | PowerEdge R510 (Intel Xeon E5620, 2.40 GHz) |
| Figure 5(a) (red solid line) | IBM BladeCenter HS22 (Intel Xeon X5550) |
| Figure 5(b) (blue dashed line) | SuperServer 5017C-MF (X9SCL-F, Intel G850) |
| Figure 5(b) (red solid line) | Acer AW2000h-AW170h F1(Intel Xeon X5670) |
| Figure 5(c) (blue dashed line) | IBM System x3850 X5 (Intel Xeon E7-4820) |
| Figure 5(c) (red solid line) | IBM System x3690 X5 (Intel Xeon E7-2830) |

1) For each benchmark $i$ ($i = 1, 2, \ldots, n$), we randomly select one performance score.

2) Calculate the geometric mean of the performance score for this computer.

3) Repeat step 1–2 $M$ times ($M$ is usually a large number, e.g., 500), so that we have $M$ geometric means, denoted as FG R(i.e., the empirical distribution of geometric mean) from $M$ repetitions.

4) Then calculate the five elements of the five-number-summary of FG: minimum, first quartile (25th percentile, denoted as Q1), median, third quartile (75th percentile, denoted as Q3), and maximum.

Detailed results will be shown in Section 6.5.

## 6 Experimental results

### 6.1 Monte Carlo simulation study on statistical power and false discovery rates (FDRs)

In order to show the effectiveness of a testing method, we examine the statistical power (the ability to detect an effect, i.e., deviation from the null hypothesis) and the false discovery rate which is the probability of having type I error (i.e., rejecting the null hypothesis while the null hypothesis is true) of our proposed method, *t*-test, and a recent proposed HPT approach [3]. A common way to evaluate and compare the statistical powers and false discovery rates (FDRs), which are defined below, of the tests is through Monte Carlo simulation study.

• **Statistical power** The probability of rejecting the null hypothesis while the null hypothesis is, in fact, not true. Note that we denote power as statistical power in this paper.

• **False discovery rates** The probability of rejecting the null hypothesis while the null hypothesis is, in fact, true.

Hence, ideally we would like the statistical power to be as large as possible and the FDR as small as possible. In real examples, we usually do not know the underlying truth. In order to investigate the properties of HPT, *t*-test, and randomization test we applied a Monte Carlo simulation study where the truth is known. Below are the settings for the Monte Carlo simulation study on power and FDR for two imaginary computers $X$ and $Y$ that uses the following steps

a) For each benchmark running on computer $X$, we randomly select $m$ ($m = 5$ in this study) execution times without replacement (i.e., each execution time can be selected at most once) from the 1,000 execution times measured from that benchmark running on computer A shown in Table 4.

b) Then we randomly pick $L$ ($L$ is between 0 and 13) benchmarks and add a constant 1.0 to all the execution times for the selected $L$ benchmarks running on the real computer, and assign the sum to be the execution time of the benchmarks running on Computer $Y$. The reason that we use constant 1.0 in step b to make a difference between two computers is that the standard deviations of the performance from all 13 benchmarks range from 0.012 to 0.91. Hence, adding 1.0 to any benchmark can guarantee that there is at least one standard deviation difference between computer $X$ and $Y$.

c) The HPT test, $t$-test, and our proposed randomization test are carried out on the data generated through steps a & b.

d) Repeat steps a–c 100 times.

**Table 4** Configurations of the four commodity computers

| Computer | Configurations |
|---|---|
| A | AMD Opteron CPU 6172 @ 2.10GHz, 2 processors, each with 12 cores, with 12GB DDR3 RAM(1333 MHz) |
| B | Intel Core i7 CPU 960 @ 3.20GHz, 1 processor with 4 cores (Hyperthreading enabled), 10GB DDR3 RAM(1333 MHz) |
| C | Intel Xeon CPU X5355 @ 2.66GHz, 2 processors, each with 4 cores, 16GB DDR2 RAM (533MHz) |
| D | Intel Xeon CPU E5530 @ 2.40GHz, 2 processor, each with 4 cores,12GB DDR3 RAM (1333MHz) |

• **Remarks** Notice that the execution times in step a for computer $X$ and $Y$ are selected from the same population (from the selected commodity computer). In step b, if $L$ is greater than zero, then the truth is computer $X$ has better performance than computer $Y$ which has longer execution times for the $L$ benchmarks. It is ideal if the test can detect the difference by rejecting the null hypothesis (i.e., the general performance of $X$ is better than that of $Y$). Hence, $P$, the proportion of times (among 100 repetitions) a test rejects the null hypothesis, can be viewed as an approximate estimate of its power for nonzero $L$. On the other hand, when $L$ is zero, that proportion, $P$, becomes an estimate of its FDR.

In this study, we set the significance level at 0.05 and use the two-sided alternative hypothesis ($H_{1c}$). Figure 6(a) shows the Monte Carlo simulation results (i.e., $P$, the proportion of times the null hypothesis is rejected) on HPT, $t$-test (TT) and the proposed randomization test (RT) using the execution time measurements from the selected computer as the underlying population. Notice that the first point ($L = 0$), the value of $P$ is an estimate of the FDR, which should be close to the specified significance level (here it is 0.05) for a good test. For other points ($L = 1, 2, \ldots, 13$), the value of $P$ is an estimate of the power, which is supposed to be large for a good test. So we can see that our proposed randomization test has much higher power than the other two tests when $L$ is between one and seven. When $L$ is greater than seven, all tests achieve perfect power. When $L$ is zero, the FDRs for all tests are small and close to the specified significance level (here it is 0.05).

Without losing generality, we also repeat the above described Monte Carlo study by using the measurements from computer C shown in Table 4 running with PARSEC in step a. Figure 6(b) shows the Monte Carlo simulation results (i.e., the proportion of times the null hypothesis is rejected) on HPT, TT, and the proposed RT using execution time measured from another computer as the underlying population. From this figure, similar observations can be made. When $L$ is between 1 and 5, RT demonstrates stronger statistical power than HPT does. This is because, unlike our proposed RT, HPT is calculated using rank-based nonparametric tests (i.e., using Wilcoxon rank-sum test in Step 1 and Wilcoxon signed-rank test in Step 2). In statistics it is well known that the statistical power for the nonparametric tests based on ranks are usually less likely to detect the effects due to the loss of some information on magnitude by ranking [15]. Regarding the $t$-test, we see it starts to have positive power when $L$ is four and reaches the perfect power when $L$ becomes seven. In fact, $t$-test shows higher power than the HPT when $L$ is between four and seven. The reason is that the parametric tests are usually more efficient (i.e., higher power) than their nonparametric rank-based counterparts which was used in the HPT method [16].

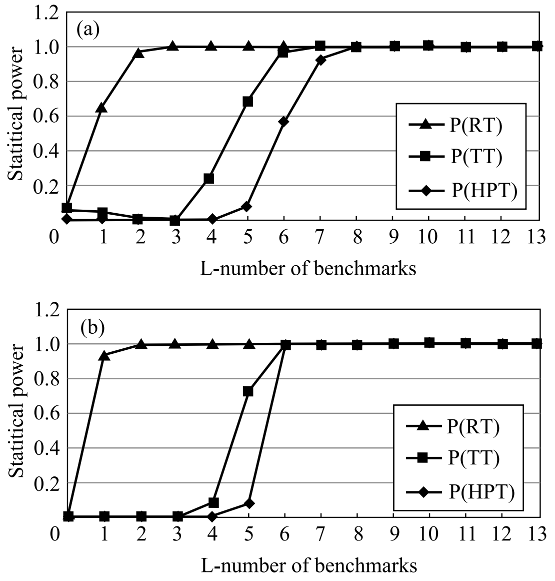


**Fig. 6** Results of Monte Carlo simulation study 1 (part (a)) and study 2 (part (b)) on statistical power and FDR

Thanks to high performance computers, the proposed randomization test (with $M = 500$) takes an average CPU timing of 0.41 seconds running on a regular Dell workstation with an Intel Xeon 2.66 GHz processor for the above experiment. The algorithm is implemented as R language functions.

## 6.2 Monte Carlo simulation study on confidence interval

Like the Monte Carlo simulation in Section 6.1, we also investigate the property of the proposed bootstrapping confidence interval and HPT speedup-under-test estimate from a simulation with known data generation mechanism. Below are the settings for the Monte Carlo simulation study on two imaginary computers $X$ and $Y$.

a) For each benchmark running on computer $X$, we randomly select $m$ ($m = 5$ in this study) execution times without replacement from the 1,000 execution times measured from that benchmark running on computer A shown in Table 4.

b) Then we multiply all the execution times (all n benchmarks) of computer $X$ by a constant 2.0. We assign the new values as execution times for computer $Y$.

c) The 95% speedups from HPT test and the proposed 95% bootstrapping confidence intervals are carried out on the data generated through step a & b.

d) Repeat step a–c 100 times.

Figure 7 shows the one hundred 0.95-Speedups from HPT test (red curves) and the proposed 95% bootstrapping confidence intervals (blue curves on the boundaries with the grey region in the middle). The black dashed line is the true ratio, 2, and the solid black line is the measured ratio of geometric mean. Note that the $t$-test confidence interval ($t$-interval), which is not shown in Fig. 7, is much wider than the bootstrapping confidence interval and outside the range of the plot. This implies our bootstrapping confidence interval is more accurate than $t$-interval. Based on Fig. 7, we have the following remarks.

1) Among all 100 bootstrapping confidence intervals, there are ninety-five intervals holding the true value, 2, which follows the pre-specified confidence level, 95%.

2) We see that the 0.95-Speedups from HPT test are consistently below the true value and the bootstrapping confidence intervals (lower than most of the lower limits of the bootstrapping CIs). This is because of the low power for the rank-based nonparametric tests.

3) The measured ratio of geometric mean varies around the true value 2 and falls within the bootstrapping CIs. This implies the ratio of geometric means is still a good estimate of comparative performance between two computers.

We also performed the above experiment on other commodity computers (listed in Table 4). The results are similar to Fig. 7. The Bootstrapping method also runs fast in R. It takes an average time of 0.51 seconds running on a Dell workstation equipped with an Intel Xeon 2.66 GHz processor for the above experiment.

## 6.3 Pairwise comparison of four commodity computers

Here, we applied our methods, $t$-test and HPT on pairwise comparison of four computers denoted as A, B, C and D which are specified in Table 4. For each computer, we run 1,000 times for each benchmark in PARSEC [17] and SPLASH-2 and then measure the execution time. All benchmarks are using their 8-thread version. In order to mimic the reality and have a full evaluation, we randomly select 5 out of 1,000 execution times (without replacement) for each benchmark and computer. Then we applied HPT, $t$-test, and our methods (RT) on the selected sample which is a subset of the whole dataset. To avoid sampling bias, we repeat the experiment 100 times.

Table 5 shows the Monte Carlo results (i.e., the number of times the null hypothesis is rejected based on 100 random repetitions) on $t$-test, HPT and proposed randomization test on all six pairwise comparisons among four computers.

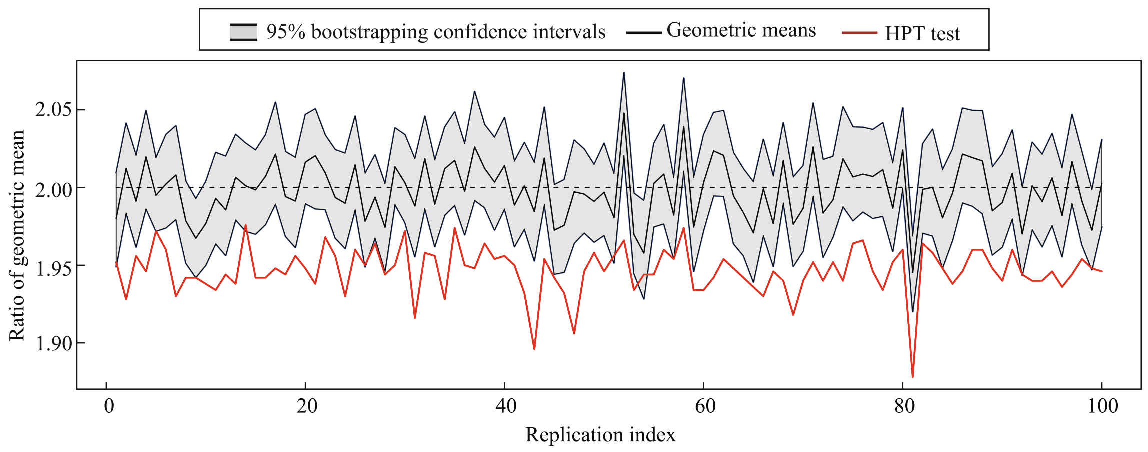


**Fig. 7** The 95% bootstrapping confidence intervals (boundaries of shaded region), measured ratios of geometric means performance speedups (solid line within the confidence interval) and 0.95-speedups from HPT test (red lines) based on 100 random replications

Based on Table 5, we have the following observations:

**Table 5** Results of pairwise comparison among four computers based on 100 random replications. The numbers shown in the table are the number of times the null hypothesis is rejected at the significance level 0.01 (the numbers in the parenthesis are for the significance level at 0.05)

| Comparison | B vs. A | D vs. A | C vs. A | D vs. B | C vs. B | D vs. C |
|---|---|---|---|---|---|---|
| HPT | 100 | 100 | 5 | 90 | 100 | 99 |
| | (100) | (100) | (91) | (99) | (100) | (100) |
| T-test | 100 | 100 | 91 | 100 | 100 | 100 |
| | (100) | (100) | (100) | (100) | (100) | (100) |
| RT | 100 | 100 | 100 | 100 | 100 | 100 |
| | (100) | (100) | (100) | (100) | (100) | (100) |

1) In four pairwise comparisons (i.e., B vs. A, D vs. A, C vs. B and D vs. C), all methods have the same conclusions (i.e., reject the null hypothesis and conclude two computers have significantly different performance.)

2) For comparing computer A and C, we see that HPT rejects the null hypothesis only 5 out of 100 times while our methods rejects the null in all 100 trials at significance level 0.01. When we change the significance level to 0.05, the number of times the null hypothesis is rejected for HPT increases to 91. *t*-test performs similar to randomization test, except it fails to reject the null hypothesis nine times at significance level 0.01.

3) For comparing computer B and D, we see that HPT rejects the null hypothesis 90 out of 100 times while both randomization test and *t*-test reject the null in all 100 trials at significance level 0.01. When we change the significance level to 0.05, the number of times the null hypothesis is rejected for HPT increases to 99.

For this experiment, we conclude that when the performance difference between two computers is large, all three tests will have the same significant conclusion. However, when performance gap between two computers is small, then the randomization test has the highest chance to detect the difference.

Figure 8 shows the one hundred 0.95-Speedups from HPT test (red curves), the proposed 95% bootstrapping confidence intervals (blue curves on the boundaries with the grey region in the middle), and 95% *t*-confidence interval (gray lines). We see that the speed-up estimates from HPT approach are smaller than the bootstrapping estimates most of the time, which concurs with the Monte Carlo simulation results in Fig. 7. This confirms that the speed-up estimates of HPT are relatively conservative than the bootstrapping estimates. Regarding the *t*-confidence interval, it is much wider than its bootstrapping counterpart, indicating that the bootstrapping method estimate is more precise than *t*-test. One interesting thing we found is that the HPT 0.95 speedup is very close to the lower bound of the 95% *t*-confidence interval. This

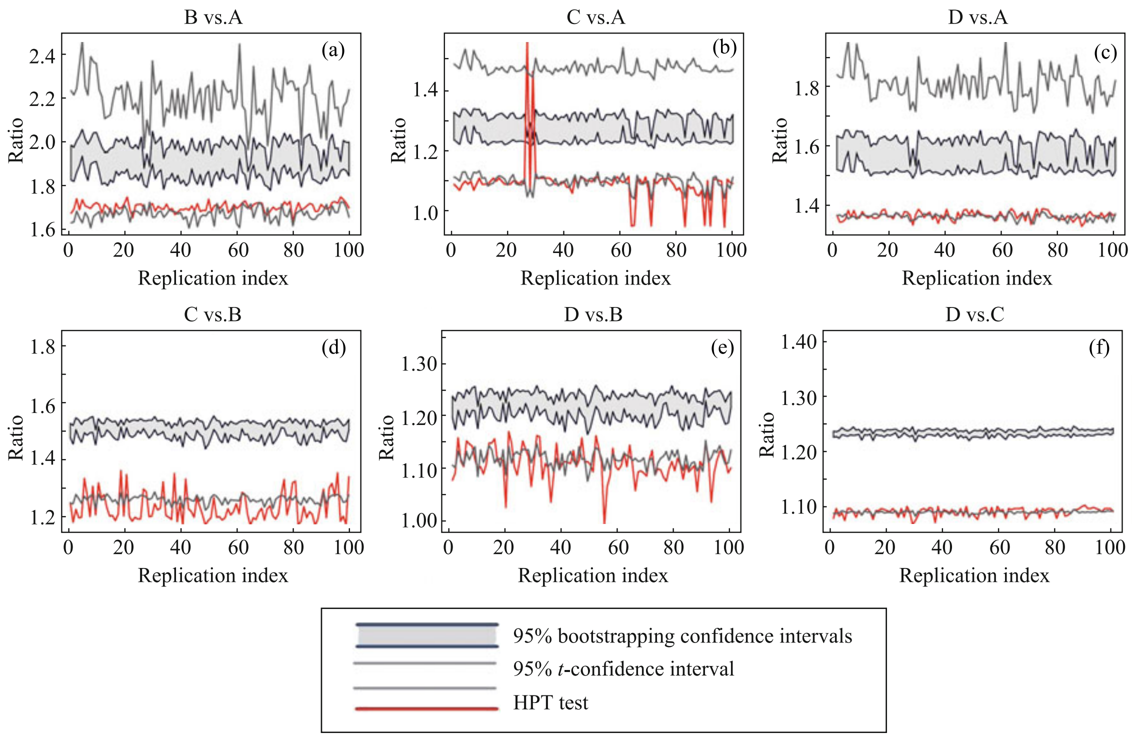


**Fig. 8** The 95% bootstrapping confidence intervals (boundaries of shaded region), 0.95-speedups from HPT test (red lines) and 95% *t*-confidence interval (grey lines) on six pairwise comparisons among Computer A, B, C and D from 100 replications. (a) B vs. A; (b) C vs. A; (c) D vs. A; (d) C vs. B; (e) D vs. B; (f) D vs. C

**Table 6** Quantitative comparisons of 0.95-performance speedups obtained by HPT, the 95% confidence intervals obtained from *t*-test, and bootstrapping method

| | A1-A2 | B1-B2 | C1-C2 | D1-D2 | E1-E2 | F1-F2 | G1-G2 |
|---|---|---|---|---|---|---|---|
| GM Speedup | 3.339 | 3.495 | 1.698 | 3.259 | 1.984 | 1.675 | 1.27 |
| HPT Speedup | 2.64 | 2.24 | 1.39 | 2.45 | 1.76 | 1.546 | 1.15 |
| T-interval | [2.626,4.245] | [2.364,5.167] | [1.417,2.035] | [2.540,4.182] | [1.733,2.272] | [1.429,1.964] | [1.139,1.417] |
| Bootstrap CI | [3.326,3.352] | [3.476,3.513] | [1.696,1.700] | [3.257,3.262] | [1.983,1.986] | [1.674,1.676] | [1.268,1.273] |

implies that the HPT speedup estimate is conservative and tends to underestimate the true speedup value.

### 6.4 SPEC CPU2006 results

Now we carry out another experiment using the data collected from SPEC.org and have been used in Chen et al. [5]. Table 6 shows the comparative results of the 0.95-performance speedups obtained by HPT, 95% *t*-intervals, and the 95% bootstrapping confidence intervals of the ratio of geometric means performance speedups. The first row shows the ratio of geometric means performance speedups from the data. Interestingly, we see that the bootstrapping CI holds the ratio of geometric means performance speedups from the data. The 0.95-performance speedups obtained by HPT are all below the bootstrapping CIs. The 95% *t*-intervals are much wider than the ones from bootstrapping method, indicating its relatively low precision for estimation compared with bootstrapping method. In addition, the HPT 0.95 speedups are close to the lower limits of the *t*-intervals.

The above experiment shows that the HPT and our methods can identify the difference between each pair of computers, although the absolute Speedup numbers are different. Now we select another seven pairs of computers from SPEC.org listed in Table 7 and perform the same experiment.

The results are listed in Table 8. We see that HPT shows low confidence and conservative estimate of Speedups in all cases while our proposed RT method demonstrates high confidence (>0.999). Similar as above results in Table 6, the 95% *t*-intervals are wider than the ones from bootstrapping method. Again, the GM Speedup is in the range of bootstrapping confidence intervals.

**Table 7** Configurations of another seven pairs of computers

| Computer 1 | Computer 2 |
|---|---|
| H1: Fujitsu, CELSIUS R570 Intel Xeon E5506 | H2: Fujitsu Siemens Computers CELSIUS M460 Intel Core 2 Quad Q9550 |
| I1: Fujitsu, CELSIUS R570 Intel Xeon E5506 | I2: Sun Microsystems Sun Fire X4450 |
| J1: Supermicro A+Server 2042G-6RF AMD Opteron 6136 | J2: Supermicro, Motherboard H8QI6-F AMD Opteron 8435 |
| K1: Huawei RH2285 Intel Xeon E5645 | K2: Fujitsu CELSIUS W380 Intel Core i5-660 |
| L1: Tyan YR190-B8228 AMD Opteron 4238 | L2: Fujitsu CELSIUS W380 Intel Core i5-660 |
| M1: Tyan YR190-B8228 AMD Opteron 4180 | M2: Fujitsu Siemens Computers CELSIUS M460 Intel Core 2 Quad Q9550 |
| N1: Fujitsu, CELSIUS M470 Intel Xeon W3503 | N2: Sun Microsystems Sun Fire X4150 |

### 6.5 Five-number-summary results

As we shown in Fig. 5, the empirical distribution described above fully embraces the variability of computer systems which stems from non-deterministic hardware and software behaviors. However, sometimes it is desired to summarize the results through a few numbers instead of the empirical distribution, which usually contains hundreds of numbers. This can be achieved through the five-number-summary of the empirical distribution. Figure 9 illustrates the five-number-summary on the IBM BladeCenter HS22. We know that the total area under the density curve is 100%. The first quartile (Q1), median, and the third quartile (Q3) cut the total

**Table 8** Comparative summary results on comparing another seven pairs of computers

| | H1-H2 | I1-I2 | J1-J2 | K1-K2 | L1-L2 | M1-M2 | N1-N2 |
|---|---|---|---|---|---|---|---|
| GM Speedup | 1.122 | 1.135 | 1.127 | 1.318 | 1.11 | 1.13 | 1.167 |
| HPT confidence | 0.732 | 0.868 | 0.576 | 0.885 | 0.753 | 0.804 | 0.825 |
| HPT Speedup | 0.950 | 0.928 | 0.944 | 0.962 | 0.94 | 0.908 | 0.932 |
| $T$ confidence | 0.849 | 0.896 | 0.878 | 0.975 | 0.814 | 0.872 | 0.891 |
| $T$-test CI | [0.956,1.316] | [0.973,1.325] | [0.967,1.314] | [1.037,1.675] | [0.948,1.298] | [0.963,1.325] | [0.964,1.413] |
| RT confidence | >0.999 | >0.999 | >0.999 | >0.99 | >0.99 | >0.99 | >0.999 |
| Bootstrap CI | [1.117,1.126] | [1.13, 1.14] | [1.117,1.138] | [1.31,1.325] | [1.109, 1.11] | [1.127,1.132] | [1.166,1.168] |

area into four equal areas, which has 25% under curve area. Hence, five-number-summary is a compact way to summarize the distribution of a random variable and it shows the following characteristics of the distribution: 1) the range of data; 2) the range of the middle 50% of the data is Q3−Q1, which is called the Interquartile range (IQR) in the statistics community; 3) the center of the distribution. Both the range and IQR are often used as measuring the variation of a random variable. Figure 10 shows the boxplots, which are the graphic presentation of five-number-summary, of the computers listed in Table 3. Note that in boxplot, the bottom and the top of the boxplot are the minimum and maximum. The bottom and top of the box are the Q1 and Q3, respectively. The line inside the box is the median.

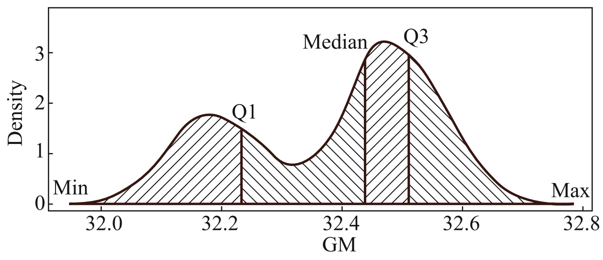


**Fig. 9** Illustration of five-number-summary on IBM BladeCenter HS22

## 7 Investigating the source of variance

For this investigation, we predict the performance variation of a hardware configuration using only a description of the hardware and the flags used for compilation and execution. To simplify this prediction, we first predict the performance of a given hardware configuration and then predict the relative variation (standard deviation of performance divided by performance) which can then be used to calculate the variation.

We use 8,236 hardware configurations running SPEC INT 2006 available from SPEC as the dataset. The reported SPEC ratio is used as the performance metric for each machine. Performance and normalized variance histograms are shown in Fig. 11.

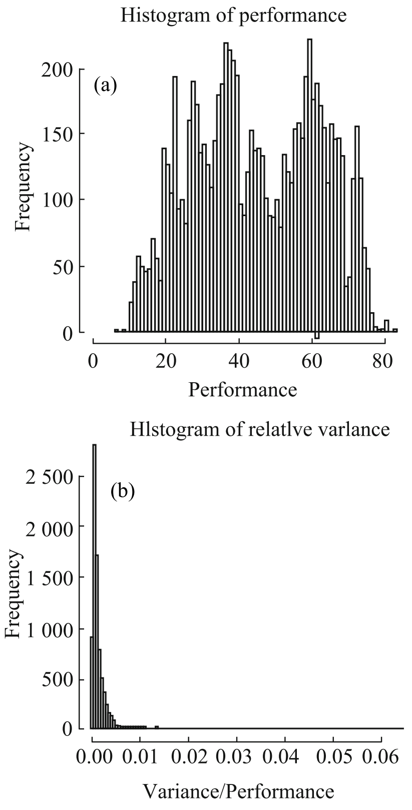


**Fig. 11** A histogram of the SPEC ratios (a) and relative SPEC ratio variance (b) for 8,236 hardware configurations running SPEC INT 2006 published between 2006 and Q2, 2017

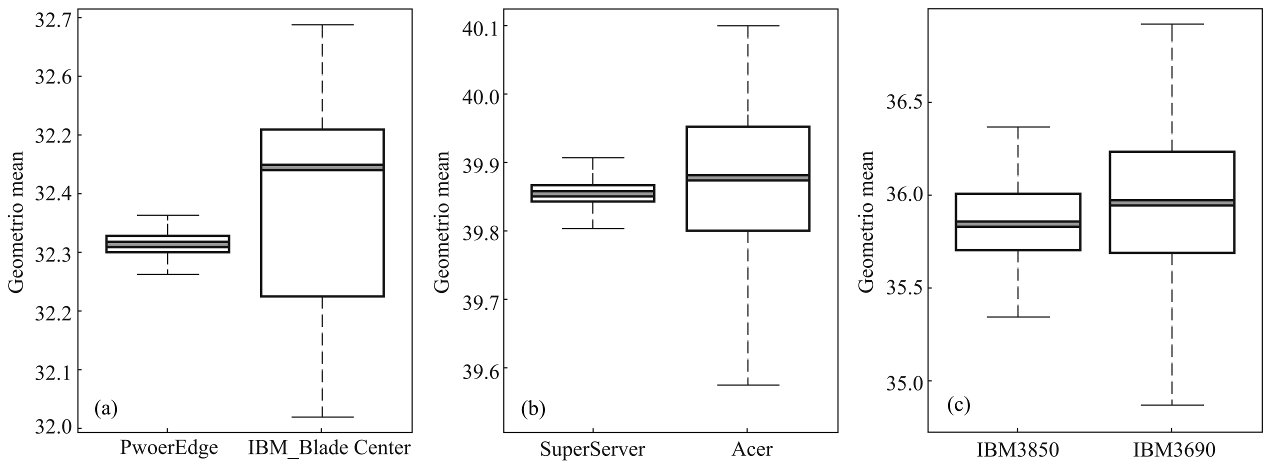


**Fig. 10** Graphic representation of five-number-summaries corresponding to the computers in Fig. 5. (a) Graphic representation 1; (b) graphic representation 2; (c) graphic representation 3

We use the published hardware configurations to train performance and relative variation predictors. For this experiment, we consider only the "base" configuration and performance results from the SPEC dataset.

For each hardware configuration, we have 24 variables describing the basic the hardware and software environment including CPU Model, Frequency, number of cores, cache sizes, etc. These variables are a mixture of integer variables (e.g., number of threads, hard disk speed) and string variables (e.g., operating system, compiler). In addition to the hardware/software environment variables, we use Boolean variables to indicate whether or not a certain flag was used during compilation or execution on this machine. Only the 100 most commonly used flags are considered during prediction. In total, we utilize 132 variables for predicting performance and relative variation.

The dataset of 8,236 machines is split into a training set and a testing set using 70% and 30% of the total dataset, respectively. The response variables are the performance and relative variation. The performance is the geometric mean of the median measure from 12 benchmarks. Note that each benchmark has three measurements. The relative variation is the ratio of the standard deviation of the geometric mean and the performance. Note that the standard deviation is estimated based on 500 bootstrap samples.

For both performance and relative variation, the boosting regression tree algorithm is used to fit predictive models using 24 environment variables as well as all 124 variables. The models are trained on training set and the prediction performance is evaluated on testing set.

The correlation of predicted and measured performance using only environment variables on test samples is 0.982. The top ten variables with the highest relative variable importance when predicting performance using only environment variables are shown in Table 9.

**Table 9** Environment variables with the highest relative influence when predicting performance

| Variable | Relative influence |
|---|---|
| File system | 40.567 |
| CPU frequency | 21.502 |
| L3 cache size | 17.014 |
| RAM stick size | 10.772 |
| L2 cache size | 1.801 |
| Disk size | 1.446 |
| Auto-parallel enabled | 1.218 |
| RAM stick count | 1.054 |
| CPU cores per chip | 0.829 |
| L1 cache size | 0.819 |

When predicting performance using only environment variables, the most influential variable is the File System type (e.g., NTFS, ext4, ReiserFS, etc.), which controls the way the operating system stores and retrieves data, followed by the CPU Clock Frequency. Variables relating to memory size are highly influential including: L1, L2, and L3 cache sizes as well as the amount of RAM (number of sticks * stick count) and the hard disk size. Variables relating to parallelism rank slightly lower: "Auto-Parallel Enabled", which allows benchmarks to use multithreading (which may improve performance but also cause inter-thread interference increasing performance variation), and the number of CPU Cores per Chip. SPEC CPU 2006 benchmarks are a mix of memory bound applications (strongly influenced by memory variables) and compute-bound applications (strongly influenced by parallelism).

The correlation is increased to 0.992 when both environment and flag variables are used to predict performance; the top ten variables are shown in Table 10.

**Table 10** Environment and flag variables with the highest relative influence when predicting performance. Flag variables are shown in bold

| Variable | Relative influence |
|---|---|
| AVX2 | 25.207 |
| File system | 20.325 |
| CPU frequency | 19.745 |
| L3 cache size | 12.589 |
| **Auto-p32** | 8.766 |
| **ParNumThreads=1** | 2.702 |
| RAM stick size | 1.289 |
| **SmartHeap64** | 1.19 |
| Auto-parallel enabled | 0.974 |
| CPU cores per chip | 0.928 |

Four flag variables are amount the top ten most influential variables when predicting performance. The most influential variable is the "AVX2" compiler flag which enables the use of "advanced vector extensions 2" instructions, which can combine multiple arithmetic or memory operations into a single vector instruction and thereby reduce the total number of instructions. The second most influential variable is the "Auto-p32" compiler flag which automatically converts 64 bit pointers to 32 bits when possible, improving performance. The "ParNumThreads" flag is used to specify the number of threads to use in a parallel region.

In the dataset, ParNumThreads is used primarily to disable parallelism by setting the number of threads to 1, which prevents variation caused by inter-thread interference. The "SmartHeap64" compiler flag enables the use of the 64-bit MicroQuill SmartHeap library which controls memory allocations in multi-threaded applications and can improve performance in heap-intensive applications.

Since the relative variation is highly skewed with some extremely large outliers, logarithm is applied to make it less skewed. Using on environment variables, the correlation of predicted and measured relative variations is 0.498. The top ten variables with the highest relative information are shown in Table 11.

**Table 11** Environment variables with the highest relative influence when predicting relative variation

| Variable | Relative influence |
|---|---|
| L2 cache size | 25.851 |
| File system | 13.958 |
| CPU chip count | 10.414 |
| Total RAM size | 8.788 |
| System state | 6.638 |
| CPU core count | 5.152 |
| L3 cache size | 4.556 |
| Threads per core | 4.011 |
| RAM stick count | 3.995 |
| Disk size | 3.494 |

The top ten variables for predicting relative variation can be broken down into two key groups. Firstly, variables related to the total number of threads, including: CPU Chip Count, System State, CPU Core Count, and Threads per Core. More threads running in parallel creates more opportunities for interference, which can act as a source of randomness and thus increase variation. The System State variable indicates the runlevel of the operating system; runlevel influences the number of OS background threads that may interfere with benchmark performance.

Secondly, variables related to memory, including: L2 Cache Size, File System, Memory Size, L3 Cache Size, RAM Stick Count, and Disk Size. Lower memory tiers are shared by more competing threads and thus larger sizes can increase the impact of thread interference. Similarly, the file system type will influence the quality of service for parallel disk accesses.

Combining the environment and flag variables, the correlation of predicted and measured relative variations is increased to 0.534. The top ten variables with the highest relative information are shown in Table 12.

When using all variables for predicting relative variation, only two flag variables appear in the top ten. "Par Num Threads = 1" disables parallelism when used, preventing variation caused by inter-thread interference by limiting the number of threads to 1. The "HugeTLBFS-link=BDT" flag instructs Linux's RAM-based filesystem to store data into huge pages. Huge pages decrease the time required to find where memory is mapped by increasing page file size from ~4 KB to ~4 MB (sizes vary by platform). Increasing the page file size reduces the total number of page files required to manage virtual memory, decreasing the time required to find a specific memory address.

**Table 12** Environment and flag variables with the highest relative influence when predicting relative variation. Flag variables are shown in bold

| Variable | Relative influence |
|---|---|
| L2 cache size | 20.976 |
| CPU chip count | 8.647 |
| File system | 7.818 |
| Total memory size | 6.724 |
| **Par Num Threads = 1** | 4.427 |
| CPU core count | 4.364 |
| System state | 3.751 |
| Threads per core | 3.688 |
| **HugeTLBFS-link=BDT** | 3.228 |
| Memory stick count | 3.066 |

From this investigation, we see that while performance can be explained almost completely by the environment and flag variables used – relative variation can only be explained in part. Our results suggest that the primary source of variation is intra-thread interference given that significant environment variables relate to the number of active threads and the size of shared memory. Flag variables were found to be less significant than environment variables when prediction variation, with the most significant flag variable being disabling parallelism for some benchmarks. We do not have variables relating to the number of OS threads running in the background or certainty that the SPEC 2006 was run as the only application, which could explain the remainder of the variation.

## 8 The sampling size

Due to the performance variability, we usually measure the performance score more than once for each benchmark. Hence, it remains a question that how many measurements (performance scores) for each benchmark, $m$, we should take. Generally, the size of $m$ depends on two factors:

1) The size of the performance variability. If there is no performance variability, then measuring once, $m = 1$, gives an accurate performance score. On the other hand, if the performance variability is large, then we need $m$ be large to have a good estimation of performance.

2) The quality of the statistical inference. Hypothesis testing and estimation are the two major branches of statistical inference. A good test procedure should have a high probabil-

ity to detect the deviation from the specified null hypothesis (i.e., high statistical power) when the null hypothesis is not true. On the other hand, the width of the confidence interval and the mean squared error (MSE) of an estimated parameter (e.g., speedup), gives us some idea about how uncertain we are about the unknown parameter. The smaller the width of a confidence interval (with fixed confidence level, e.g., 95%) and MSE, the more precise the estimate is. Hence, the statistical power, MSE and the width of confidence interval are widely used to examine the quality of statistical inference.

Here, we redo the Monte Carlo simulation study on power, described in Section 6.1, with $L = 1$ on the commodity computer (AMD Opteron CPU 6172 @ 2.10GHz, 2 processors, each with 12 cores, with 12GB DDR3 RAM(1333 MHz)) using different sizes of $m$, $m = 3, 5, 7, 10, 15, 20, 30, 50, 100$. The top panel of the proposed bootstrap estimate with different sizes of $m$. The vertical grey bar indicates the standard deviation of MSE. We see that the size of MSE (the smaller the MSE, the more accurate the estimate is) and its standard deviation decreases with the increase of $m$. Sometimes we may constrain the width of the confidence intervals. For example, we want to have a 95% confidence interval with width (i.e., upper limit – lower limit) no greater than 0.03. Notice that the smaller the width, the more consistency the estimate has. The bottom panel of Fig. 12 shows the width of 95% confidence interval with different size of $m$. The vertical grey bar indicates the standard deviation of width. Similar to MSE, we see that the width of confidence interval decreases as $m$ increases.

The above study shows the statistical properties of the proposed methods by increasing the size $m$. However, in practice we usually do not know the truth. Hence, the power of the test and MSE are unknown. A common way to determine the size of $m$ is by setting the width of the confidence interval in advance. Figure 13 shows the flowchart of selecting the size of $m$ in practice based on the predetermined width of confidence interval $\Delta$. Basically, we need specify an initial value of $m$, usually a small value like 3, and a threshold for the width of confidence interval $\Delta$. Then we sample $m$ measurements for each benchmark and computer. We calculate a bootstrapping confidence interval based on the sample data. If the width of confidence interval is greater than the threshold $\Delta$, then we increase the size of $m$ and sample more measurements for each benchmark and computer. Then we recalculate the confidence interval. We stop sampling when the width of confidence interval is no greater than the predetermined threshold $\Delta$.

For the example below, we use two computers: A and C described in Section 6.3. We would like to find the size of $m$ by restricting the width of the bootstrapping confidence interval of the ratio of geometric means performance speedups to be no greater than 0.015. Table 13 shows the bootstrapping confidence intervals and corresponding width with various sizes of $m$. We see that the sample size of $m$ should be at least 16 under the restriction.

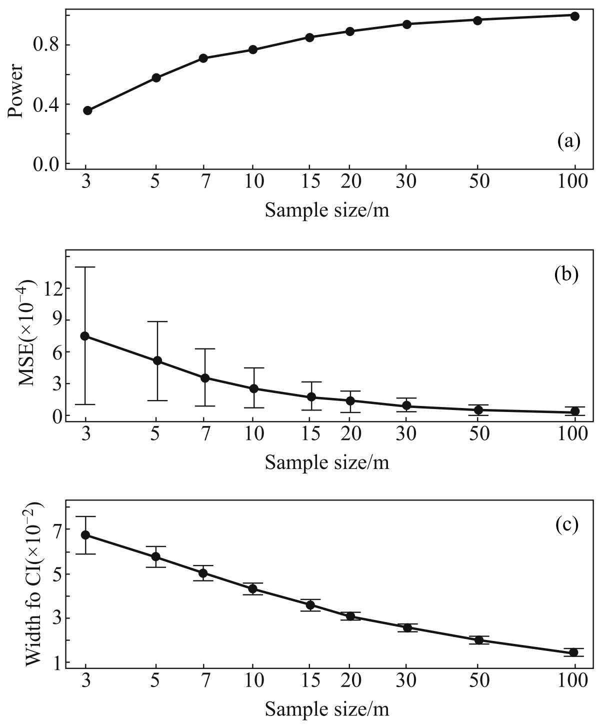


**Fig. 12** The sample size effect on the statistical power, MSE and the width of confidence interval under various sizes of $m$. (a) The sample size effect on the statistical power; (b) the sample size effect on the MSE; (c) the sample size effect on the width of confidence interval

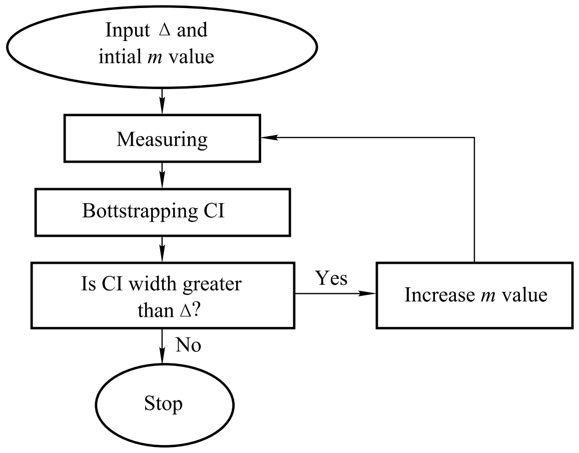


**Fig. 13** Flowchart of choosing the sample size based on the width of confidence interval

**Table 13** An illustration of choosing the sample size (m) based on the width of confidence interval

| *m* | 3 | 5 | 7 | 10 | 13 | 15 | 16 |
|---|---|---|---|---|---|---|---|
| Bootstrap CI | [1.203, 1.228] | [1.204, 1.223] | [1.207, 1.227] | [1.212, 1.228] | [1.216, 1.231] | [1.216 1.232] | [1.217, 1.232] |
| CI Width | 0.0256 | 0.0198 | 0.0194 | 0.0166 | 0.0153 | 0.0155 | 0.0149 |

## 9 Applicability to other means

As a generic framework, our proposed methods can be directly applied to arithmetic and harmonic means while the HPT framework cannot apply since it uses rank instead of any performance metric. We applied the propose methods using these three means on an example in which we compare SPEC scores of two machines: IBM System x3500 M3 with Intel Xeon E5530, and CELSIUS R570 with Intel Xeon X5560, which are obtained from SPEC website. Table 14 shows the confidences and confidence intervals using three metrics on the example. We see that both harmonic mean and geometric mean identify the difference between two computers while arithmetic mean cannot. This is because the arithmetic mean is subject to extreme values. For example, among 29 benchmarks, CELSIUS R570 has 25 benchmarks with a larger mean performance score than their counterparts for IBM System x3500 M3. However, IBM System x3500 M3 has much higher performance scores in the libquantum and bwaves benchmarks than their counterparts in CELSIUS R570. If the two benchmarks are eliminated from the data, then changes in the confidence and confidence interval using the arithmetic mean are much larger than the ones using the geometric and harmonic means.

**Table 14** Summary of comparing geometric, harmonic and arithmetic means on confidence and confidence interval (CI)

| | G-mean | H-mean | A-mean |
|---|---|---|---|
| Confidence | >0.99 | >0.99 | 0.492 |
| CI | [0.913, 0.920] | [0.887, 0.892] | [1.019, 1.031] |
| Confidence* | >0.99 | >0.99 | >0.99 |
| CI* | 0.882, 0.889] | [0.881, 0.886] | [0.880, 0.889] |

## 10 Applicability to Big Data benchmarks

In this section we study the effectiveness of the proposed sampling methods on Big Data benchmarks [12], which have been demonstrated to be different from traditional CPU benchmarks like SPEC or PARSEC. Big Data analytics is an emerging field that is driven by the need to find trends in increasingly large data sets. Applications include search engines, social networking, e-commerce, multimedia analytics, and bioinformatics. Big Data applications require extra layers in the software stack due to the use of distributed storage and processing frameworks, such as Apache Hadoop, thus creating additional opportunities for variance. We find the execution-time-variance of Big Data applications (calculated as the standard deviation divided by the mean) to be about twice as large as that of spec benchmarks; this is due to these additional virtualization layers used by the Big Data Bench (i.e., Hadoop, Spark, Java).

As listed in Table 15, a set of seven Big Data benchmarks were chosen from the spark implementation of the BigDataBench version 3.1.1 and executed on five separate machines listed in Table 16. Each benchmark was executed 1,000 to 2,000 times on each machine and the execution time was measured. The larger variance of Big Data application performance makes naïve comparisons of machine performances impractical and mandates a sampling method such as the one proposed.

We ran three studies using the big data described above. Study 1 and 2 are both based on the random sampling of Machine 3 and Machine 4. Namely, for each benchmark from each computer, five execution times are randomly selected without replacement. Then we (1) compare the two computers using HPT, *t*-test and proposed randomization test; (2) estimated the ratio of the geometric means through the proposed bootstrapping confidence interval, *t*-test confidence interval and HPT speedup-under-test estimate based on the randomly selected subset of data. Both studies were repeated 100 times. (3) For Study 3, we applied a new visualization tool

**Table 15** Summary of selected Big Data workloads

| ID | Domain | Operations or algorithm | Types | Data sets |
|---|---|---|---|---|
| a | Social networks | Connected components | Offline analytics | Facebook social network |
| b | Social networks | Kmeans | Offline analytics | Facebook social network |
| c | Search engine | Sort | Offline analytics | Wikipedia entries |
| d | Search engine | Grep | Offline analytics | Wikipedia entries |
| e | Search engine | Word count | Offline analytics | Wikipedia entries |
| f | E-commerce | NaiveBayes | Interactive analytics | Amazon movie reviews |
| g | Search engine | Page rank | Offline analytics | Google Web graph |

**Table 16** Summary of selected computers

| ID | Configurations |
|---|---|
| 1 | Intel Xeon CPU E5-2630 @ 2.6 GHz, 2 processors, each with 12 cores, 192GB DDR3 RAM (1600 MHz) |
| 2 | 2,Intel Xeon CPU X5530 @ 2.40GHz, 2 processors, each with 4 cores, 12GB DDR3 RAM (1333MHz) |
| 3 | Intel Core i7 CPU 3820 @ 3.6 GHz, 1 processor with 8 cores, 24GB DDR3 RAM (1600 MHz) |
| 4 | Intel Core i7 CPU 960 @ 3.20 GHz, 1 processor with 4 cores (Hyperthreading enabled), 10GB DDR3 RAM(1333MHz) |
| 5 | AMD Opteron CPU 6172 @ 2.1GHz, 2 processors, each with 12 cores, with 12GB DDR3 RAM(1333 MHz) |

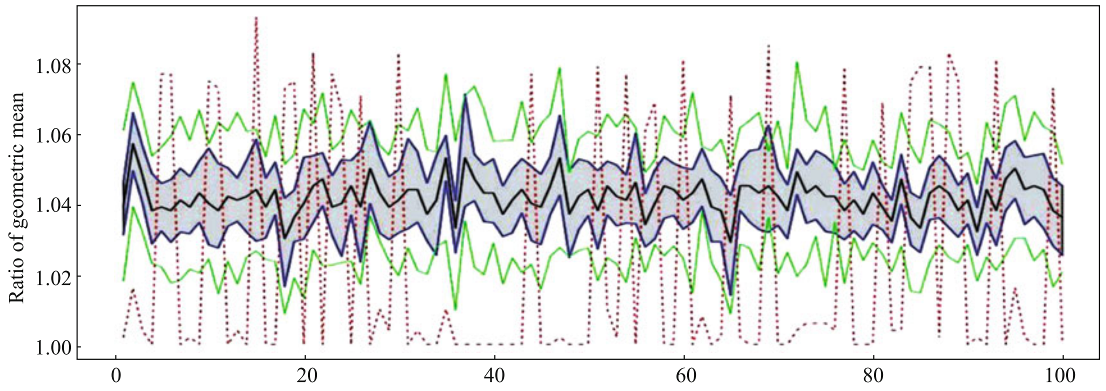


**Fig. 14** The 95% bootstrapping confidence intervals (solid blue lines), measured ratios of geometric means (solid black line within the confidence interval), 95% $t$-test confidence intervals (solid green lines) and 0.95-speedups from HPT test (red dash lines) based on 100 random replications

called a Biplot [13] to visually examine the performance of many computers and benchmarks simultaneously.

In Study 1, for a significance level of 0.05, HPT fails to reject null hypothesis as the two machines generally have the same performance in terms of the geometric mean, 69% of times, while $t$-test and our randomization test both are 0% (i.e., reject all 100 times). When the significance level is 0.01, since HPT uses nonparametric test, their $p$-value in this case cannot go below 0.01. The $t$-test fails to reject the null hypothesis 4% of the time, while our test still rejects all 100 times.

## 11 Biplots for the visualization of benchmark effectiveness

Figure 14 shows the results of Study 2. The black solid line in the center is the observed geometric means based on 100 simulations. The blue solid lines show the 95% bootstrapping confidence intervals. The green solid lines show the 95% $t$-test confidence intervals. The red dash line shows the HPT speed-up estimates. Based on the figure, we can see that the $t$-test confidence interval is consistently wider than the bootstrapping confidence interval and that the HPT speedup estimates are highly variable bouncing up and below and far away from the observed Geometric means.

Finally, we use a Biplot visualization tool [13] for computer performance comparisons. Biplot is a useful tool to visualize the projections of high-dimensional data onto a low dimensional space through principal component analysis. In this section, we will first briefly describe the principal component analysis technique and introduce the Biplot method through an illustrative example. Then we will apply the Biplot method to the performance results of all five machines used in section IX with seven Big Data benchmarks and explain the results that may shed new insights on comparing computer performance.

Principal component analysis is a time-honored method for dimension reduction and data visualization. Figure 15 shows a randomly generated dataset with 1,000 points from a bivariate Gaussian distribution. Figure 15(a) shows the raw data with the two principal components. The first principal component (PC1), shown as the red arrow in the plot, is the direction in feature space (e.g., $X_1$ and $X_2$ in this case) along which projections have the largest variance. The second PC (PC2), shown as the blue arrow in the plot, is the direction which maximizes variance among all directions orthogonal to the first PC. The principal components are the linear combination of all the features. The value of the coefficients for the PC is called the loading vector of the corresponding PC. The value for the sample point of the PC is called the score for the corresponding PC. For example, PC1 is equal to $0.996X_1+0.258X_2$; hence the loading vector for PC1 is (0.996, 0.258). For a sample point with $X_1=1$ and $X_2=0$, the PC1 score is equal to $0.996\times1+0.258\times0=0.996$.

Instead of plotting the data on its raw scales, an alternative way to visualize the data is to project the data onto PC1 and PC2. In this example, since the data contains only two variables, $X_1$ and $X_2$, projecting onto PC1 and PC2 is equivalent to rotating the data to use PC1 and PC2 as the horizontal and vertical axes. This is shown in Fig. 15(b). For each point, the projected value on the horizontal axis is its PC1 score, while the projected value on the vertical axis is its PC2 score. A Biplot graph, which is shown in Fig. 15(c), presents not only the PC scores but also the loading vectors in a single display. The red arrow shows the coefficient values for $X_1$ on the PC1 and PC2 loading vectors. As can be seen, the coefficient value for $X_1$ in PC1 (i.e., 0.966) is larger than its counterpart in PC2 (i.e., 0.258) and the coefficient value for $X_2$ in PC2 is negative (i.e., –0.966), with its absolute value being larger than its counterpart in PC1. Hence, we can see PC1 reflects mainly the variation in the $X_1$ direction, and PC2 mainly reflects variation in the $X_2$ direction.

Figure 15(d) shows the proportion of variance that is explained by each PC. Since the data has only two variables, there are at most two PCs. The first PC explains about 95% of the total variance of the data, while PC2 explains the remaining 5%.

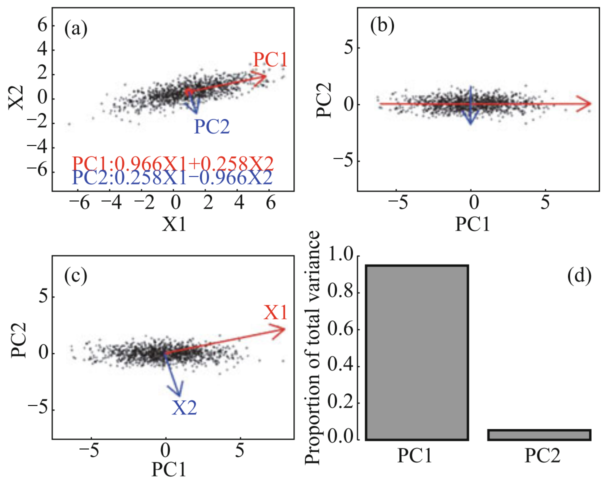


**Fig. 15** Illustrative example for principal component analysis and biplot. (a) Raw data with PC1 and PC2; (b) PC scores on the PC1 and PC2; (c) lower left: biplot of the data; (d) proportion of total variance explained by PC1 and PC2

Figure 16 shows the Biplot of the performances of all five machines used in Section 9 with all seven Big Data benchmarks. Note that for each machine and each benchmark, we have measured about 1,000 times. To create the Biplot in Fig. 16, we use the median value of the performance measure for each benchmark and machine. The median values for all five machines and all seven Big Data benchmarks are listed in the Table 17. Since we have five machines and seven benchmarks, there are up to five PCs. The right panel of Fig. 16 shows the proportion of total variance explained by each PC. As we can see, the first two PCs explained more than 99.7% of the total variance. Hence, using the leading two PCs in the Biplot keeps almost all the information in the data. Based on the Biplot, which is shown on the left panel, we have the following remarks.

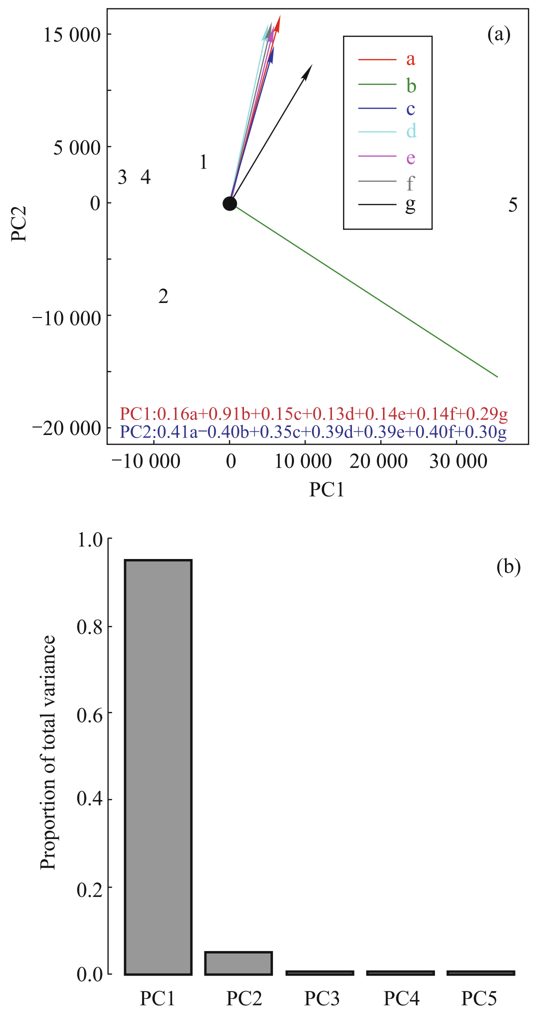


**Fig. 16** Biplot on big data benchmark example. (a) Biplot on PC1 and PC2 together with the loading values for seven benchmarks; (b) proportion of total variance explained by five PCs

1) We see that the benchmark b has the largest impact (i.e., coefficient value) on the PC1. This indicates that PC1 roughly reflects the performance measure on benchmark b. This can be verified by the dominant value of the loading coefficient for benchmark b in PC1 (i.e., equal to 0.91).

2) For PC2, the remaining six benchmarks measures are clustered together and have about the same impact (i.e.,

coefficient value). This indicates that these six measures (from benchmark a, c, d, e, f, g) are highly correlated to each other and PC2 mainly reflects the average performance on these six benchmarks. Table 18 shows the pairwise correlation among all seven benchmarks. We see that most of the pairwise correlations among benchmarks a, c, d, e, f, g are over 0.95 (shown in bold type).

3) The PC1 score for machine 5 is far greater than the other four machines. This is due to its higher performance on all seven benchmarks and particularly on benchmark b (i.e., 92291).

4) The PC2 score for machine 2 is the smallest among all. This is due to its lower performance on benchmark a, c, d, e, f, g, for which is has the lowest values among all five machines, and relatively large value on benchmark b, which for which it has the third largest value among all.

5) Overall, machines 1, 3 and 4 have similar performance over all seven benchmarks. Machine 5 has the highest overall performance, while machine 2 has the lowest overall performance.

**Table 17** Median values of all five machines on seven big data benchmarks

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| 1 | 12746 | 53600 | 14182 | 12785 | 14473 | 13292 | 19774 |
| 2 | 7265 | 53581 | 9157 | 7427 | 9154 | 7718 | 14602 |
| 3 | 10945 | 44492 | 12101 | 10894 | 12379 | 11184 | 16028 |
| 4 | 11499 | 47084 | 12318 | 11205 | 13062 | 11444 | 16997 |
| 5 | 18429 | 92291 | 18867 | 16448 | 18915 | 17429 | 29271 |

**Table 18** Pairwise correlation among all seven big data benchmarks

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| a | 1.00 | 0.82 | 1.00 | 0.99 | 1.00 | 1.00 | 0.96 |
| b | 0.82 | 1.00 | 0.83 | 0.76 | 0.80 | 0.78 | 0.94 |
| c | **1.00** | 0.83 | 1.00 | 0.99 | 1.00 | 1.00 | 0.97 |
| d | **0.99** | 0.76 | **0.99** | 1.00 | 1.00 | 1.00 | 0.93 |
| e | **1.00** | 0.80 | **1.00** | **1.00** | 1.00 | 1.00 | 0.95 |
| f | **1.00** | 0.78 | **1.00** | **1.00** | **1.00** | 1.00 | 0.95 |
| g | **0.96** | 0.94 | **0.97** | **0.93** | **0.95** | **0.95** | 1.00 |

## 12 Related work

Over decades, the debate over the method and metrics for computer performance evaluation has never ended [18–20]. Fleming and Wallace [15] argued that using geometric mean to summarize normalized benchmark measurements is a correct approach while arithmetic mean will lead to wrong conclusions in this situation. Smith, however, claimed that geometric mean cannot be used to describe computer performance as a rate (such as mflops) or a time by showing counter examples. Furthermore, Johnson [16] advocated using weighted arithmetic mean or harmonic mean instead of geometric mean to summarize computer performance over a set of benchmarks. Hennessy and Patterson [21] described the pros and cons of geometrics mean, arithmetic mean, and harmonic mean. Eeckhout [22] summarized that arithmetic and harmonic means can clearly describe a set of benchmarks but cannot apply the performance number to a full workload space, while geometric mean might be extrapolated to a full benchmark space but the theoretic assumption cannot be proven.

Relying on only a single number is difficult to describe system variability stemming from complex hardware and software behaviors. Therefore, parametric statistic methods such as confidence interval and *t*-test have been introduced to evaluate performance [1, 23]. Nevertheless, Chen et al. [7] demonstrated that these parametric methods in practice require a normal distribution of the measured population which is not the case for computer performance. In addition, the number of regular benchmark measurements from SPEC or PARSEC is usually not sufficient to maintain a normal distribution for the sample mean. Therefore, Chen et al. [7] proposed a non-parametric statistic hypothesis tests to compare computer performance. As demonstrated in the paper, our proposed resampling methods can identify smaller differences between two computers even in a situation where a single test is not enough to reveal it.

Oliveira et al. [24] applied quantile regression to the non-normal data set and gained insights in computer performance evaluation that analysis of variance (ANOVA) would have failed to provide. Our approach considers different variation sources (non-deterministic or deterministic behaviors) for the fixed computer configurations and handles the non-normality by using resampling technique such as bootstrapping and permutation.

Patil and Lilja [25] demonstrated the usage of resampling and Jackknife in estimating the harmonic mean of an entire dataset. Unlike their approach, we applied resampling methods on a more complicated situation-comparing two computers on multiple benchmarks with multiple measurements. Hence, the bootstrapping method in our paper is different from the one in [25]. Namely, we bootstrap the samples within each benchmark instead of on the entire dataset.

Much research has been conducted in an effort to identify and remove sources of performance variation (and thus increase quality of service) in cloud computing systems performing many concurrent tasks. Iosup et al. [26] study the impact of workload on performance variability in cloud services via program trace analysis. Similarly, Leitner and Cito [27] profile infrastructure-as-a-service (IaaS) cloud systems seeking the cause of performance variation, especially inter-task interference. In contrast, our Big Data benchmark performance study uses only one active task.

Previous work has been conducted on profiling applications to predict performance variation in multi-threaded systems. Zhang et al. [28] propose VarCatcher for measuring the performance variation of individual execution paths within an application; execution patterns are then clustered in an effort to explain performance variability. Pusukuri et al. [29] use runtime performance metrics (i.e., cache misses, thread context-switches) to throttle inter-thread interference and thereby reduce performance variation. Jimenez et al. [30] predict the performance variation bounds for compute intensive applications and propose to limit variation by reducing bandwidth at the cost of reduced performance. Our research similarly studies the relationship between hardware and OS-level events, but we data collected from many different hardware configurations to predict variation in advance. This work is an extension of our prior ISPASS publication [16] which was limited in scope to statistical resampling methods for measuring computer performance on SPEC benchmarks without the use of Biplot visualization tools.

## 13 Conclusion

We propose a randomization test framework for achieving a both accurate and practical comparison of computer architectures performance. We also propose a bootstrapping confidence interval estimation framework for estimating a confidence interval on a quantitative measurement of comparative performance between two computers. We illustrate the proposed methods through both Monte Carlo simulations where the truth is known and real applications.

Interestingly, even though geometric mean as a single number cannot describe the performance variability, we find that the ratio of geometric means between two computers always falls into the range of Boosted confidence Intervals in our experiments. In cases where two computers have very close performance metrics, we propose using empirical distribution to evaluate computer performance and using five-number-summary to summarize the computer performance.

We investigate the source of performance variation by using hardware and environment descriptions to predict performance and relative variation with a predicted and measured correlation of 0.992 and 0.5 respectively. The best predictors of relative variation are found to be the degree of parallelism and the size of amount memory space, suggesting performance variation comes in large part from thread interference.

We demonstrate that the proposed sampling method is effective at differentiating the performances of machines running Big Data benchmarks, which have higher variance than traditional CPU benchmarks. Our analysis of Big Data benchmark variance was extended using a Biplot to visualize machine performance similarities and benchmark correlation.

**Acknowledgements** This work was supported in part by the National High Technology Research and Development Program of China (2015AA015303), the National Natural Science Foundation of China (Grant No. 61672160), and Shanghai Science and Technology Development Funds (17511102200), National Science Foundation (NSF) (CCF-1017961, CCF-1422408, and CNS-1527318). We acknowledge the computing resources provided by the Louisiana Optical Network Initiative (LONI) HPC team. Finally, we appreciate invaluable comments from anonymous reviewers.

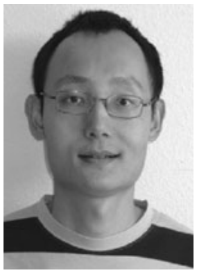
## References

1. Alameldeen A R, Wood D A. Variability in architectural simulations of multi-threaded workloads. In: Proceedings of the 9th IEEE International Symposium on High Performance Computer Architecture. 2003, 7–18
2. George A, Buytaer D, Eeckhout L. Statistically rigorous java performance evaluation. ACM SIGPLAN Notices, 2007, 42(10): 57–76
3. Mytkowicz T, Diwan A, Hauswirth M, Sweeney P F. Producing wrong data without doing anything obviously wrong. In: Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2009, 265–276
4. Krishnamurthi S, Vitek J. The real software crisis: repeatability as a core value. Communications of ACM, 2015, 58(3): 34–36
5. Chen T, Guo Q, Temam O, Wu Y, Bao Y, Xu Z, Chen Y. Statistical performance comparisons of computers. IEEE Transactions on Computers, 2015, 64(5): 1442–1455
6. Freund R J, Mohr D, Wilson W J. Statistical Methods. 3rd ed. London: Academic Press, 2010
7. Chen T, Chen Y, Guo Q, Temam O, Wu Y, Hu W. Statistical performance comparisons of computers. In: Proceedings of the 18th IEEE International Symposium On High Performance Computer Architecture. 2012, 1–12
8. Hollander M, Wolfe D A. Nonparametric Statistical Methods. 2nd ed. New York: John Wiley & Sons, 1999
9. Moore D, McCabe G P, Craig B. Introduction to the Practice of Statistics. 7th ed. New York: W. H. Freeman Press, 2010
10. Edgington E S. Randomization Tests. 3rd ed. New York: Marcel-Dekker, 1995
11. Davison A C, Hinkley D V. Bootstrap Methods and Their Application. New York: Cambridge University Press, 1997
12. Wang L, Zhan J, Luo C, Zhu Y, Yang Q, He Y. Bigdatabench: a big data benchmark suite from internet services. In: Proceedings of the

20th IEEE International Symposium on High-Performance Computer Architecture. 2014, 488–499

13. Gower J C, Lubbe S G, Roux N L. Understanding Biplots. Hoboken: John Wiley & Sons, 2011
14. Efron B, Tibshirani R J. An Introduction to the Bootstrap. New York: Chapman and Hall/CRC, 1994
15. Fleming P J, Wallace J J. How not to lie with statistics: the correct way to summarize benchmark results. Communications of the ACM, 1986, 29(3): 218–221
16. Johnson R A. Statistics: Principles and Methods. 6th ed. New York: John Wiley & Sons, 2009
17. Bienia C, Kumar S, Singh J P, Li K. The PARSEC benchmark suite: characterization and architectural implications. In: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques. 2008, 72–81
18. Citron D, Hurani A, Gnadrey A. The harmonic or geometric mean: does it really matter? ACM SIGARCH Computer Architecture News, 2006, 34(4): 18–25
19. Iqbal M F, John L K. Confusion by all means. In: Proceedings of the 6th International Workshop on Unique Chips and Systems. 2010, 1–6
20. Mashey J R. War of the benchmark means: time for a truce. ACM SIGARCH Computer Architecture News, 2004, 32(4): 1–14
21. Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Approach. 4th ed. Walthan: Morgan Kaufmann, 2007
22. Eeckhout L. Computer Architecture Performance Evaluation Methods. California: Morgan & Claypool Press, 2010
23. Lilja D J. Measuring Computer Performance: A Practitioner's Guide. New York: Cambridge University Press, 2000
24. Oliveira A, Fischmeister S, Diwan A, Hauswirth M, Sweeney P F. Why you should care about quantile regression. In: Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2013, 207–218
25. Patil S, Lilja D J. Using resampling techniques to compute confidence intervals for the harmonic mean of rate-based performance metrics. IEEE Computer Architecture Letters, 2010, 9(1): 1–4
26. Iosup A, Yigitbasi N, Epema D H J. On the performance variability of production cloud services. In: Proceedings of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Newport Beach. 2011, 104–113
27. Leitner P, Cito J. Patterns in the chaos—a study of performance variation and predictability in public IaaS clouds. ACM Transactions on Internet Technology, 2016, 16(3): 15
28. Zhang W, Ji X, Song B, Yu S, Chen H, Li T, Yew P, Zhao W. Varcatcher: a pramework for tackling performance variability of parallel workloads on multi-core. IEEE Transactions on Parallel and Distributed Systems, 2016, 28: 1215–1228
29. Pusukuri K K, Gupta R, Bhuyan A N. Thread tranquilizer: dynamically reducing performance variation. ACM Transactions on Architecture and Code Optimization, 2012, 8(4): 46–66
30. Jimenez I, Maltzahn C, Lofstead J, Moody A, Mohror K, Arpaci-Dusseau R, Arpaci-Dusseau A. Characterizing and reducing cross-platform performance variability using OS-level virtualization. In: Proceedings of the 1st IEEE International Workshop on Variability in Parallel and Distributed Systems. 2016, 1077–1080

Samuel Irving received the bachelor's degrees in both computer science and electrical engineering from Louisiana State University (LSU), USA in December 2011. He is currently enrolled in the Computer Engineering PhD program and received the Donald W. Clayton PhD Assistantship at LSU. His research interests include machine learning, big data analytics, and heterogeneous architecture design.



Bin Li received his Bachelor's degree in Biophysics from Fudan University, China. He obtained his Master's degree in Biometrics (08/2002) and PhD degree in Statistics (08/2006) from The Ohio State University, USA. He is an associate professor with the Experimental Statistics department at Louisiana State University, USA. His research interests include statistical learning & data mining, statistical modeling on massive and complex data, and Bayesian statistics. He received the Ransom Marian Whitney Research Award in 2006 and a Student Paper Competition Award from ASA on Bayesian Statistical Science in 2005. He is a member of the Institute of Mathematical Statistics (IMS) and American Statistical Association (ASA).

Shaoming Chen received the bachelor's and master's degrees in electronics and information engineering from the Huazhong University of Science and Technology, China in 2008 and 2011, respectively. He is currently working in AMD after receiving the PhD degree in electrical and computer engineering at Louisiana State University, USA in August 2016. His research interests include sub-memory system design and cost optimization of data centers.

Lu Peng received the bachelor's and master's degrees in computer science and engineering from Shanghai Jiao Tong University, China, and the PhD degree in computer engineering from the University of Florida in Gainesville in April 2005. He is currently Gerard L. "Jerry" Rispone professor with the Division of Electrical and Computer Engineering at Louisiana State University, USA. His research focus on memory hierarchy system, reliability, power efficiency and other issues in processor design. He received an ORAU Ralph E. Power Junior Faculty Enhancement Awards in 2007 and the Best Paper Award (processor architecture track) from IEEE International Conference on Computer Design in 2001. He is on the editor board of Microprocessors and Microsystems.

Weihua Zhang received the PhD degree in computer science from Fudan University in 2007. He is currently an associate professor of Parallel Processing Institute, Fudan University, China. His research interests are in compilers, computer architecture, parallelization and systems software.

Lide Duan is currently an assistant professor in the Department of Electrical and Computer Engineering at The University of Texas at San Antonio, USA. Prior to joining UTSA, he worked as a senior CPU design engineer at AMD, working on future x86 based high performance and low power CPU microarchitecture design and performance modeling. He received a PhD in Computer Engineering from Louisiana State University, USA in 2011. His PhD research focused on soft error reliability analysis and prediction for processors at computer architecture level. He also received a bachelor's degree in Computer Science from Shanghai Jiao Tong University, China in 2006.