Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 2341973, 11 pages
https://doi.org/10.1155/2021/2341973

Research Article

Probability Analysis to Improve the Confidence in Profiling Accuracy

Yingying Wen,[1] **Guanjie Cheng**,[1] **Bo Lin**,[2] **and Jianwei Yin**[1]

[1]*Computer Science and Technology, Zhejiang University, Hangzhou, China*
[2]*Innovation Centre for Information, Binjiang Institute of Zhejiang University, Hangzhou, China*

Correspondence should be addressed to Yingying Wen; wingwingtwo@hotmail.com

Received 22 July 2021; Accepted 17 November 2021; Published 30 November 2021

Academic Editor: Yung-Chung Wang

Performance profiling for the system is necessary and has already been widely supported by hardware performance counters (HPC). HPC is based on the registers to count the number of events in a time interval and uses system interruption to read the number from registers to a recording file. The profiled result approximates the actual running states and is not accurate since the profiling technique uses sampling to capture the states. We do not know the actual running states before, which makes the validation on profiling results complex. Jianwei YinSome experiments-based analysis compared the running results of benchmarks running on different systems to improve the confidence of the profiling technique. But they have not explained why the sampling technique can represent the actual running states. We use the probability theory to prove that the expectation value of events profiled is an unbiased estimation of the actual states, and its variance is small enough. For knowing the actual running states, we design a simulation to generate the running states and get the profiled results. We refer to the applications running on production data centers to choose the parameters for our simulation settings. Comparing the actual running states and the profiled results shows they are similar, which proves our probability analysis is correct and improves our confidence in profiling accuracy.

1. Introduction

In data centers, performance is critical to improve the quality of service [1] and save costs [2]. Multiple tasks controlled by the operating system share computation resources at the same time to improve user experience and resource utilization. The whole system's performance representing the combination of multiple tasks is not enough for analysis of each task's performance. Many applications need to know the running states of specific tasks. These applications include anomaly detection on data centers [3, 4], compiler optimization using method stacks [5, 6], and hot spots detections [7, 8].

Modern processors have hardware support to monitor system performance. Hardware Performance Counters (HPC) [9] are register-based counters to count the number of events in a time interval. With the help of interruption, HPC can output the counted number to a recording file. For profiling each task's performance, only one extra information is in need—the instruction address. The instruction address indicates which task the processor is working for at the moment of interruption. The profiling technique then treats the counted events in the last time interval as all caused by this task indicated by the instruction address. It is not accurate to use the instant instruction address to represent the running states of a long sustaining time interval. But it has already been used to profile the performance of tasks.

Many widely used profiling tools have already adopted this approximation method, like PAPI [10, 11], perf [12], and VTune [13]. Profiling accuracy attracts research attentions. The experiment-based evaluation compares the profiling results across multiple system architectures to improve the confidence of the profiling technique [14, 15]. And CPU simulator gives detailed information, which makes the comparison more direct [16]. But these researches utilized simple benchmarks to check the accuracy. This kind of validations cannot deduce other workloads' conditions since the mechanism lacks proof and analysis—they have not explained why the sampling technique can represent the actual running states.

In this paper, we show the mechanism of the profiling technique with the help of probability theory. We model the profiling process with two main elements: the running granularity of a task and the sampling interval. We classify all possible conditions into three classes according to the rate between running granularity and sampling interval. For a constant rate, the sampling process is a kind of Bernoulli experiment [17]. We prove no matter what the rate value is, the expectation value is unbiased to the actual value, including the condition with a mixture of rate values that would still keep the unbiased property. And the variance is related to the number of samples, which is small enough and usually smaller than 0.25.

We further use the simulation experiments to validate our proof. The implementation of simulation includes the generation of actual running states and the sampling process. The settings of the simulation follow the characteristics of the workloads running on live production data centers. We simulate single tasks and mixed tasks running with multiple running granularities and under multiple resource utilization levels. All of these experiments show that the expectation value is an unbiased estimation. The variance is also included into consideration, whose effect does not influence the unbiased property.

We organize our paper as follows. Section 2 introduces the background of the profiling technique. We propose our analysis model in Section 3. Section 4 designs the simulation model. Section 5 shows the simulation results including the simulation's prerequisite. Section 6 reviews the related work. And we conclude in Section 7.

2. Profiling Technique

In this section, we introduce the profiling technique used for recording the running states of clusters. For example, Figure 1 is a profiled result of the Windows operating system. The green part at the bottom shows the value changes of CPU utilization in this 60-second observing window. The blue line shows the rate of current operating frequency to the highest frequency. These lines link the samples of every second. And each sample represents the averaged CPU utilization of the last 1-second time interval.

2.1. HPC Profiles. Hardware performance counters consist of two components—event detectors and event counters [18]. Users can configure performance event detectors to detect performance events as cache misses, cycles consumed, or branch mispredictions. Often, event detectors have an event mask filed that allows further qualifications of the event. According to the processors' privilege mode, for example, HPC can collect the kernel occurred events with the administrator mode.

The event counter would increase itself by one if this event happened once until a system interruption happened.

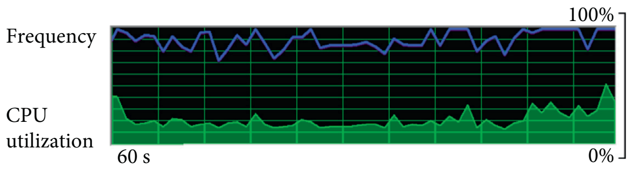

Figure 1: A CPU utilization record sustaining for 60 seconds.

It outputs its historical value and its value can be reset to zero or not according to whether it is on accumulative counting mode. The condition to cause this kind of system interruption can be separated into two types:

(i) Time-based sampling is implemented through interrupting tasks' execution at regular time intervals and recording the program counters. This approach is often used to show the relationships between profiled events to time dimensions.

(ii) Event-based sampling is implemented through interrupting after a specific number of performance events—when the number of events that happened reaches a threshold. Users can specify the threshold events.

The hardware performance counter method has distinct advantages. First, it profiles the system from the hardware level without any intrusion to applications, making applications and operating systems remain largely unmodified. Second, every modern processor has the support of performance counters. This method is a general solution. Third, this method profiles system on the fly as the applications executing to save the effort to reproduce the workloads, since some executions are prohibitively complex to be simulated or reproduced.

Though hardware performance counters reveal lots of information from the system view, this information mainly exposes the system's overall states, not a specific task's behaviours.

2.2. Task-Oriented Profiles. For profiling the running states of tasks, only one extra information is in need to be added. The information is the instruction address coming from context information [19, 20]. We would set periodic events to trigger the sampling, like for every 100 cache misses. At the sampling moment, the recorded sample contains the content of performance counters and instruction address. Then profiling technique uses this sample to represent the running of the last sampling interval.

Figure 2 illustrates the profiling method. The upper bar is the actual running state, and the lower bar is the state captured by profiling. The profiled state is different from the actual running state. The upper blue block shows the actual running of task XX. The upper orange blocks show the actual running condition of task YY. And the vertical lines represent the sampling moments that are triggered by the event threshold or time limit. The first sample regarding the last sampling interval events was all caused by task YY. The second sample would treat the last

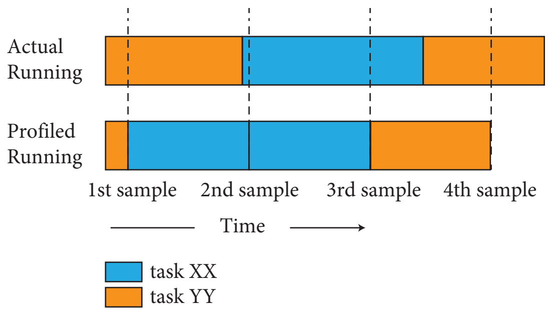

Figure 2: The illustration of sampling.

sampling interval events caused by task XX shown in the lower bar, though YY was running in most of the second sampling interval tasks. The profiling result shown in this figure is that task XX and task YY both consumed 2 units of the resource.

Our paper's task is an abstract presentation that can represent the threads, processes, programs, or applications according to the analysis granularity. If we intend to profile the performance of an application, then the task means the application. However, an application can be divided into multiple threads. All these threads are regarded as running for a single application—a single task.

2.3. Challenges. Checking the profiling method's accuracy has many challenges from the complex environment and the limitations of profiling techniques. In the following, we list the changes from two aspects.

First, we do not have the standard answers to validate the profiling results. In the data center, there are tens of thousands of applications running on thousands of computers [21]. These applications include online services that have high requirements on response time and off-line services that require high throughput. Sometimes the clusters can reach extremely high pressure, for example, the double 11 shopping festival. The complex environment makes every next moment different from the last one. We do not know the true proportions of the running applications or the true load of queries from users. Many production scenes appeared only once, which means these conditions could hardly be repeated.

Second, the profiling technique would unavoidably introduce overhead to the running system [22]. With the increasing sampling frequency, the overhead would increase, making it impossible to increase the sampling density too much to get detailed enough running states. And it is also impossible for current profiling techniques to separate the profiling workload from the original workload.

The experiments on benchmarks only prove some events' correctness under certain workloads, and these experiment-based researches have not covered all scenarios. An explanation of why profiling can be trusted would improve our confidence when profiling the system that has not been covered.

3. Analysis Model

3.1. Application Scenario. A representative scene using task-oriented profiling is the hot spot detection. Taking the hot methods detection as an example, it targets finding out the top hot methods that consumed the most resources (like CPU cycles) for further performance optimizations. Not every method can catch enough attention to be optimized further since there are too many methods running on a live environment to be optimized one by one. Thus for profiling how many CPU cycles are consumed by a method, we can set a sampling-based method to profile the running of methods.

For example, we set a sample of 0.1 seconds, which means every 0.1 seconds to interrupt the system running and record the current instruction address. This instruction address indicates the running method, for example, is "Sort()," and the number of cycles consumed is 200 million in this sampling interval. Then we count that the "Sort()" method consumed 200 million cycles. This interruption on the system is repeated to get an overview of the CPU cycle's consumptions of methods.

3.2. Model Components. We model the profiling process as two major elements to help us do further analysis. The main elements that need to be considered include the following.

(i) Running granularity: The averaged scheduling time of a task running continually until being switched out. Running granularity would be influenced by many factors like the property of this task, our observing level, system environment, etc. The length of a color block shown in Figure 3 is called the running granularity. The running granularity does not need to be a constant value.

(ii) Sampling interval: The distance between the last sampling to current sampling. Figure 3 shows an example. If the interruption is event-based rather than time-based, and the number of events is not proportional to time, the sampling interval's length would look nonuniform from the time dimension. But from the corresponding event dimension, it is still of uniform intervals.

In the following analysis, the base event is to denote the event dimension that causes the sampling interruption. For example, if it is time-based sampling, then the base event is time, and if it is CPU cycles based sampling (e.g., interrupt system every 250 million cycles), then the base event is CPU cycles. The number of base events that happened in a sampling interval is a constant value without variance. We call the constant number of events that happened in a sampling interval a unit of events.

About the nonbase events, the numbers of these events collected by samples may not be as steady as the base event. The number of nonbase events in a sampling interval would be different from the other sampling intervals. Their estimation variance would be a little higher than base events. We include the considerations on nonbase events by introducing variance to the constant unit of events when doing

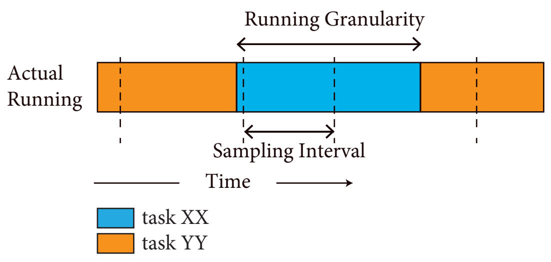

Figure 3: The main elements of the model.

experiment. To avoid introducing extra variable considerations into probability model, we first model our analysis focusing on base event, which can be further extended into nonbase events by adding extra considerations on the variance of the unit of events.

3.3. Three Classes of Conditions. The accuracy of estimations on the base event would be mainly influenced by the tasks switches reflected by the rate between running granularity and sampling interval. When the sampling interval becomes smaller, and the running granularity keeps the same, the sampling's accuracy would increase, and the error bound would be smaller. Assuming an extreme condition that the sampling interval equals every clock cycle, the profiled result reflects the real running state accurately without any approximation.

We utilize the rate between the running granularity and sampling rate to define all possible conditions. We define a variable R to denote the rate as

$$R = \frac{\text{Running Granularity}}{\text{Sampling Interval}}. \tag{1}$$

According to the value of R, there are three kinds of conditions as shown in Figure 4. They are (a) $R = 1$, (b) $R < 1$, (c) $R > 1$.

(i) Figure 4(a) represents the $R = 1$ condition that the sampling interval and the running granularity are the same.

(ii) Figure 4(b) represents the $R < 1$ condition that the running granularity is smaller than the sampling interval, which means it is possible that the tasks already have been switched more than once within one sampling interval.

(iii) Figure 4(c) represents the $R > 1$ condition that the running granularity is larger than sampling interval.

We use these three kinds of conditions to help with our further analysis.

With a specific constant R value, the sampling process is a kind of Bernoulli experiment whose results would follow a binomial distribution. The Bernoulli experiment means running a task would be captured by a sample or would not be repeated independently. We first use cases with representative R values to illustrate the calculation of the profiling distribution's expectation value, conclude them with a

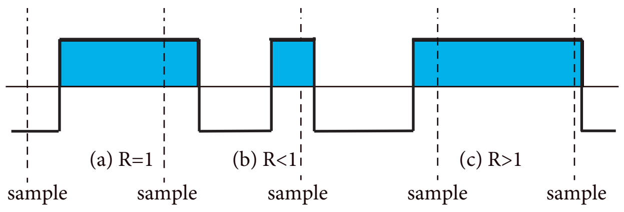

Figure 4: Three classes of conditions. (a)$R = 1$. (b)$R < 1$. (c)$R > 1$.

general representation method, and show its corresponding variance calculation method.

(1) For the first condition that $R = 1$, the sampling interval and the running granularity are of the same length. No matter where the sampling starts, one of two adjacent samples would capture this task and regard it caused by one unit of base events—one unit of base events means the constant number of base events that happened in a sampling interval. This is an accurate estimation without errors.

Additionally, when the R value is integer, like 2 or 3, the sampling result would keep the same condition as the $R = 1$ and give an accurate estimation.

(2) For the second condition that $R < 1$, we denote the time that the sample is captured as

$$T(i) = t_0 + i \cdot T_p. \tag{2}$$

The i value means this is the i^{th} sample, and the profiling starts from t_0 with a period of T_p. Every T_p would occur an interruption to get the sample. There is an assumption on the t_0. We regard the time starting to profile (t_0) as randomly chosen—t_0 is independent of T_p or other factors.

We assume r proportion ($r < 1$, e.g., 30%) of sampling interval is working for a task. The probability of being captured by a sampling point equals the running proportion of this task as $r\,(P(\text{captured}) = 30\%)$ in this sampling interval since the sampling point is independent of running this task. And the probability that this task is missed (not captured by the sampling point) is $1 - r\,(P(\text{missed}) = 70\%)$. This process is repeated, and we get a bunch of profiled samples. When we denote a unit of events as $\#(\text{events})$, the expectation value of events caused by this task in an interval can be deduced as

$$E(\text{events}) = P(\text{captured}) \cdot \#(\text{events}) + P(\text{missed}) \cdot 0 = r \cdot \#(\text{events}). \tag{3}$$

We can find that the expectation value equals the real running proportion.

(3) For the third condition that $R > 1$, we first analyze the case when the task's running granularity is less than 2 times of sampling interval and bigger than 1 time of sampling interval denoted as r times of sampling interval length. There are two possible conditions.

One is shown in Figure 4(c) that this task appears in three intervals and is captured by two samples. Another one is shown in Figure 5 that this task appears in two intervals and only is captured by one sample. The probability of the first condition captured by two samples is $P(\text{two}) = r - 1$, and the probability of the second condition captured by a single sample is the left probability $P(\text{one}) = 1 - P(\text{two}) = 2 - r$. The expectation value of $R > 1$ can be combined from these two probabilities as

$$\begin{aligned} E(\text{events}) &= P(\text{two}) \cdot 2 \cdot \#(\text{events}) + P(\text{one}) \\ &\quad \cdot \#(\text{events}) \\ &= r \cdot \#(\text{events}), \end{aligned} \tag{4}$$

where the $\#(\text{events})$ represents a unit of events. This condition would come to unbiased estimation too.

3.4. Piecewise Binomial Distribution. We conclude this deduction process to be a more general representation. The running granularity is r times of sampling interval. All possible sampling conditions are separated into two classes—when the running is captured by $\text{int}(r)$ samples and captured by $\text{int}(r) + 1$ samples. The probabilities of being captured by $\text{int}(r) + 1$ samples and $\text{int}(r)$ samples equal to $P(\text{int}(r) + 1) = \text{MOD}(r, 1)$ and $P(\text{int}(r)) = 1 - \text{MOD}(r, 1)$. And the expectation is

$$\begin{aligned} E &= P(\text{int}(r) + 1) \cdot (\text{int}(r) + 1) \cdot + P(\text{int}(r)) \\ &\quad \cdot \text{int}(r) \\ &= \text{MOD}(r, 1) + \text{int}(r) = r, \end{aligned} \tag{5}$$

where the $\text{int}(\cdot)$ is a function to get the integer part of this element and the $\text{MOD}(r, 1)$ is an operation to get the fractional part. For example, $\text{int}(3.14)$ equals 3 and $\text{MOD}(3.14, 1)$ equals 0.14. This expectation value shows profiling method would get an unbiased expectation value about the running proportion.

We can find the sampling result distribution can be regarded as a binomial distribution under a specific constant R value. According to the binomial distribution, we get the variance (V) of sampling distribution under a specific r value as

$$V = \text{MOD}(r, 1) \cdot (1 - \text{MOD}(r, 1)). \tag{6}$$

This expectation value and variance value is about the distribution where samples are drawn. It is an ideal model, and enough number of samples can approach its distribution according to Chebyshev's theorem [23].

In actual running conditions, the running granularity would change, which makes the R value vary. We can regard the varying R value as the mixture of components with different R values. But no matter what kind of mixture proportion of these components, the expectations of components are all equal to r value. Then the combined expectation is equal to r. This can be denoted as

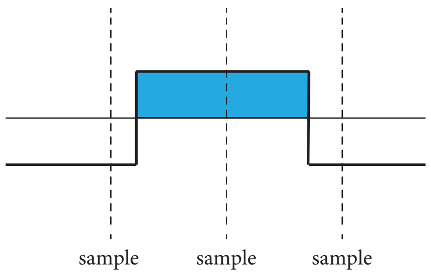

Figure 5: Another condition for possible sampling points.

$$E_{\text{combined}} = \sum_{i=1}^{C} \text{Pro}_i \cdot E_i, \tag{7}$$

where the Pro_i represents the proportion of i^{th} component, the E_i represents the expectation value of i^{th} component. The property that the expectation value is unbiased still exists when considering varying R values.

3.5. The Number of Samples. The number of samples would influence the approximation to the distribution. When the number of samples is n, then the mean value of these samples would keep the same to expectation value, and the variance of samples would be related with n:

$$E_{\text{Samples}} = x, \tag{8}$$

$$V_{\text{Samples}} = \frac{\text{MOD}(x, 1) \cdot (1 - \text{MOD}(x, 1))}{n}. \tag{9}$$

The maximum variance value appears as 0.25, when $\text{MOD}(x, 1) = 0.5$ and $n = 1$. The small variance can improve our confidence in the current profiling method. Taking the worst condition as an example, it still has good performance. The probability of misestimating the mean value by one more unit event is less than 0.022 8.

4. Simulation Model Design

We introduce how to implement our simulation process. Two parts need to be modelled: the actual running states and the profiling process.

It is a reflection of the condition that multiple tasks share the computing resources about the running state generation. For simplifying the resources, we will not detail computing resource into more specific types. We treat the resource such that it can only be occupied by one task at one moment, which is a simplified model to the true system. But model duplications to consider multiple types of resources are similar to the true system. For example, the multiprocessor CPU would run multiple processes simultaneously. At one moment, the CPU resource of the system can be shared by different tasks. But this true condition can be simulated by repeating this simplified model. A running model represents one processor resource that runs a single process at one moment. This simplification keeps the basic property of the system. If necessary, this model can be rebuilt into a complex system. For simulating the running states, we regard the smallest resource amount allocated to run tasks as a resource unit (e.g., a CPU cycle). Several parameters need to be specified, including the work amount of tasks (the number of resource units needed), their corresponding running granularity, and their scheduling and sharing behaviour.

Regarding the sampling method, the sampling interval is the key setting. By changing the sampling interval and running granularity, we simulate different R value conditions. The sampling interval would keep stable roughly with little variance. We add random noise into the sampling interval to approach the real profiling scene. The start time of the sampling t_0 can be generated randomly. We get the profiled samples based on the same true running condition, repeating the sampling with different randomly generated start time and different noise introduced. The statistics of profiled samples about the proportion of tasks are supposed to approach real task running proportion values.

5. Experimental Evaluation

This section shows the simulation results on various kinds of single task and mixed tasks, including the variance introduced to simulate nonbase events. We first introduce the simulation components and the experimental results are followed to prove the effectiveness of our model.

5.1. Simulation Components. Our experiments are conducted on real running environments and the analysis data is collected by the system profiling tool. We call it a simulation experiment since the workload that is running on the system is simulated without actual functions and is under control. There are three components that need to be clarified, including data collection method, the control of workloads' running granularity, and the method to introduce the nonbase event variance.

5.1.1. Data Collection Method. We use the "perf" to profile the system—the Linux kernel already contains this profiling tool. An application or a program serving for user requests consists of multiple methods, and these methods belong to their corresponding modules. The "perf" script would offer an automatic parsing function to map Instruction Pointer (IP) value to the corresponding method and module, record the hardware events, and index each sample with the sampling timestamp, CPU number, and event name. The dimensions collected by the "perf record" script related to our model and their meanings are shown in Table 1. We profiled the running states of these physical machines from CPU view—each recorded sample represents the state of a CPU in a sampling interval.

5.1.2. Running Granularity Scale. We regard the continuing samples with the same module name as the condition that this module has not been switched out—its running granularity value is calculated as the sum of the continuing sampling interval. The distribution of sampling interval from the time dimension is shown in Figure 6. The distribution of

Table 1: The dimensions of collected data.

Metrics	Meaning
Timestamp	The time to record the value of hardware counter
CPU	CPU number where the sample is collected
IP	Instruction pointer value at the sampling time
Method	The method name
Module	The module name
Event	Hardware event name, for example, cycles consumed
Value	The value of the corresponding hardware event counter

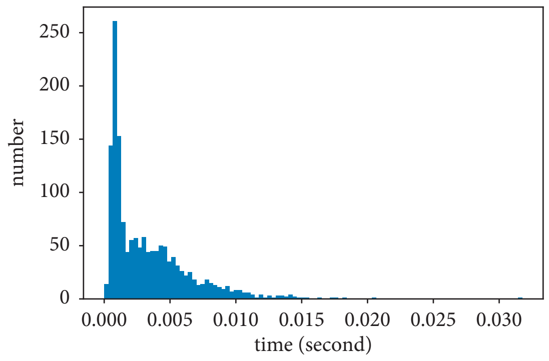

Figure 6: The distribution of sampling intervals when profiling the data center. This is a histogram whose x-axis represents the time length of sampling interval and y-axis represents the number of samples belonging to a specific value range.

running granularity deduced from the sampling results is shown in Figure 7. When the running granularity is smaller than the sampling interval, we treat it equal to the sampling interval, making running granularity here a little larger than its actual value. The running granularity is not always smaller than the sampling interval. It is about several times of sampling interval. Thus the R value scale that we experiment with, like 1 to 10, is good enough to cover the conditions rather than a thousand or million scale.

5.1.3. Nonbase Event Variance. For showing the variance of nonbase event is small when profiling and our model about the variance of nonbase event is reasonable, we analyze the variance of "cycle" event when the base event is "time." The sampling intervals in the collected data are not the same; thus, we scale the cycle event value by dividing the number of cycles by the sampling interval's length. For example, the distribution of profiled cycle event in one physical machine that ran for 231 modules in 5 seconds is shown in Figure 8. The cycles consumed in a sampling interval tend to be fully utilized or in idle state, but this characteristic does not make the variance large for each module. For each module, we filter out its corresponding cycle event samples and calculate its variance. For these 213 modules that appeared in our 5-second observing window, not every module has a large number of samples. Thus we filter out 111 modules whose number of samples is larger than 10. We get the variances of these 111 modules shown in Figure 9. The mean value of the variances is 0.3171. The variance we introduce to simulate nonbase events between 0 to 2 is in a reasonable range.

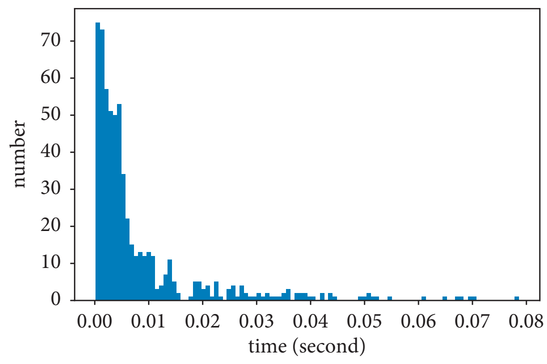

Figure 7: The distribution of running granularity deduced from sampling results. This is a histogram whose x-axis represents the time length of running granularity and y-axis represents the number of samples belonging to a specific value range.

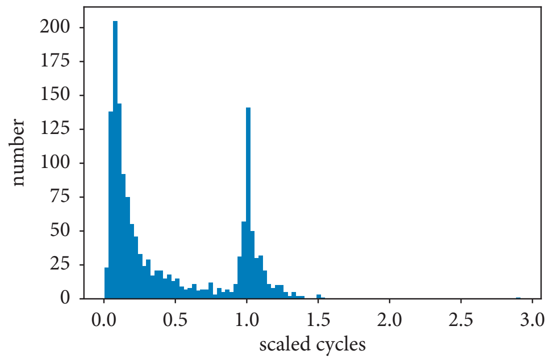

Figure 8: The distribution of cycles consumed event. The x-axis represents the proportion of consumed cycles to the total available cycles in a sampling interval.

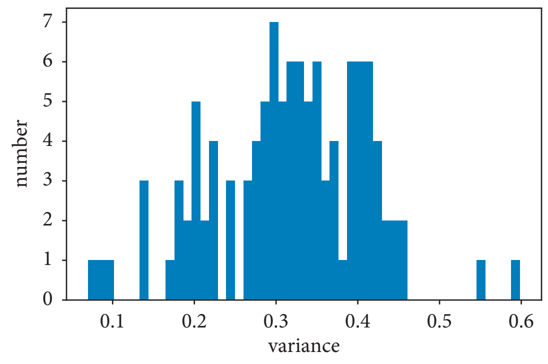

Figure 9: The distribution of variance values of 208 modules running on a physical machine. In this histogram, each sample means a variance value of a module, whose x-axis represents the variance value and y-axis represents the number of modules in a specific value range. The variance of module is calculated by its corresponding profiled samples.

5.2. Unbiased Estimation. In this section, we show that, under any R condition, the expected value of profiled samples is unbiased, and their variance is small. We first use multiple types of tasks respectively with different utilization levels and keep the sampling interval the same—which means different R values. And we also show the estimation of mixed tasks with different running granularity also has good performance.

5.2.1. Single Task. We set 1 million units to simulate running states, and each unit can work for a task or in idle. We set the utilization of units as 80%, 50%, and 30% to combine with the running granularity as 30, 50, 100, 150, 200, and 280 separately to cover a total of 18 types of running states. Then we profile these 18 types of running states by setting every 100 units to trigger interruption to get a sample—the sampling interval is 100, and repeat profiling on each running state 1000 times and get 1000 profiling results of each running state.

We do not introduce any extra variance first and randomly generate the running states for each unit according to the set running granularity. Taking the 80% utilization load level as an example as shown in Table 2, the mean value of utilization for these 1 million units approach to the set utilization value. But within each sampling interval, this task's utilization is not always 80%, as shown in Figure 10—we take 30 running granularity as an example to plot out the actual running states for each sampling interval.

The profiling process is conducted on these 18 types of running states. We find profiling results have little variance and are unbiased to the expectation value—the detailed result of 80% is shown in Table 2. The mean value distribution of 1000 times profiling result on 30 running granularity and 80% utilization condition is shown in Figure 11 whose minimum value is 0.786 9 and maximum value is 0.809 2. The profiling results of the other conditions with 50% and 30% utilizations are shown in Table 3 whose expectation values approach the actual utilization value and variance values are small too.

But we may doubt the low variance of these 1000 profiling results caused by the large number of samples collected by each profiling process—reaching 10 thousand samples. Of course, a large number of samples would guarantee a low variance. The fact is that the variance is still small enough even when the number of samples is only 1. We reduce the number of samples to 1. For example, the variance of 80% utilization when running granularity is 30 equals 0.004. Thus we can believe that, without extra variance introduced, the estimation is unbiased, and its variance is small.

5.2.2. Mixed Tasks. Except for a single task running on a system, it is more usual that multiple types of tasks (with different running granularity) share a system simultaneously. We simulate this condition by mixing tasks with a specific proportion. We show the result to make the system with 80% utilization composed by 30% task 1 with 30 running granularity, 20% task 2 with 50 running granularity, 20% task 3 with 150 running granularity, and 10% task 4 with 280 running granularity. We observe on 1 million units, and the sampling granularity is still 100. The generated running states show that each task's mean value is unbiased, as shown in Table 4 with small variance. We can find the mixed running still keeps the unbiased property.

Table 2: Different running granularity under 80% utilization.

Running granularity	R	Mean of running states	Mean of profiling	Standard deviation of profiling
30	0.3	0.798 4	0.798 3	3.34e-05
50	0.5	0.797 9	0.797 8	2.77e-05
100	1.0	0.803 2	0.803 1	5.75e-07
150	1.5	0.800 1	0.800 1	3.73e-05
200	2.0	0.798 2	0.798 2	4.19e-07
280	2.8	0.800 8	0.800 7	2.60e-05

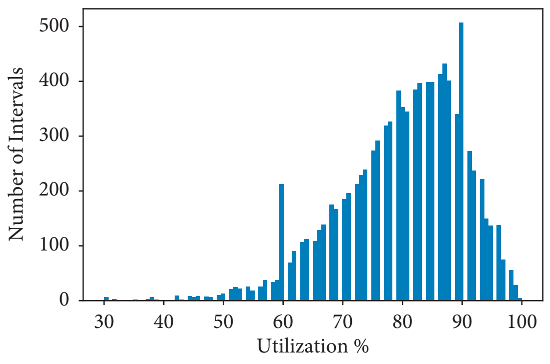

Figure 10: The actual running state in each sampling interval is not always equal to 80% utilization.

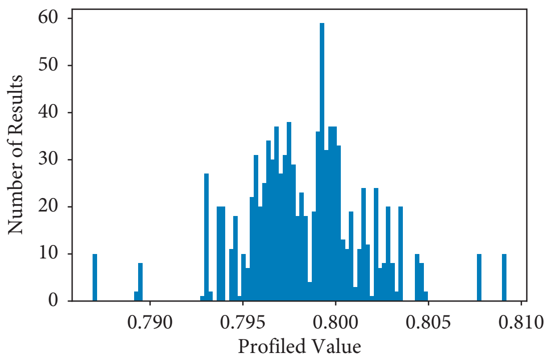

Figure 11: The distribution of profiled results. The x-axis is the profiled proportion of this task. There are 1000 samples. One profiling on system means one sample here. Any profiled result is around the correct answer with good performance.

Table 3: The profiling results of conditions with 50% and 30% utilization.

Running granularity	Utilization = 50%		Utilization = 30%	
	Mean	SD	Mean	SD
30	0.500 8	$4.69e-03$	0.294 5	$3.27e-03$
50	0.500 3	$4.79e-03$	0.297 8	$3.84e-03$
100	0.494 9	$0.04e-03$	0.299 9	$0.03e-03$
150	0.497 4	$1.38e-03$	0.295 3	$2.89e-03$
200	0.504 5	$0.07e-03$	0.295 7	$0.13e-03$
280	0.490 9	$2.43e-03$	0.298 0	$1.33e-03$

5.3. Variance Introduced to Simulate Nonbase Events. The sampling interval and unit of events are constant values in the former experiments—having no variance. But the real condition would always have some variations. Thus we explore the impact of the variance of sampling interval and unit of events in this section. Some nonbase events would not be as accurate as base events. Thus we introduce variance to sampling interval or unit of events to simulate the performance of nonbase events. The inaccuracy of the sampling interval would influence the specific number of the units of events. Thus the variance introduced to sampling interval or unit of events has the same effect to simulate.

Table 4: The simulation results of mixed tasks.

Task	Running granularity	Proportion (%)	Mean of running states	Mean of profiling	SD of profiling
Idle	1	20	0.198 3	0.198 1	$3.27e-03$
1	30	30	0.302 5	0.302 6	$4.24e-03$
2	50	20	0.203 9	0.203 8	$3.15e-03$
3	150	20	0.195 3	0.195 4	$1.20e-03$
4	280	10	0.099 9	0.100 0	$0.41e-03$

We introduce the noise to a unit of events—the number of events counted by a sample varies by a noise value generated randomly from a normal distribution. We denote the normal distribution by $N\,(\text{mean, standard deviation})$ function. We profile the same running states of mixed tasks as the former section and introduce noise to a unit of events whose value is regarded as 100 before—since the sampling interval is set to 100. The noise for each sample is drawn from the normal distributions $N(0,0.5)$, $N(0,1)$, and $N(0,2)$, respectively, and the unit of events profiled for each sample is calculated by

$$U_{\text{noise}} = U \cdot (1 + \text{noise}). \tag{10}$$

The mean and standard deviation (SD) value of 1000 times profiling—each profiling gets about 10 thousand samples—with noise introduced following three normal distributions, respectively, are shown in Table 5.

We also reduce the number of samples collected by each profiling process. When the number of samples is reduced to 1 thousand, it is shown in Table 6. The estimation is still unbiased, and the variance is influenced by injected noise but not too much to influence the mean value. The different actual running states cause the differences between task 1 and task 4 when $n = 500$ and noise following $N(0,1)$ distribution—the real proportions of task 1 and task 4 are 0.343 2 and 0.556 0. This means they are still unbiased estimations on the real proportions.

Table 5: Introduce noise into the unit of events.

Task	$N(0,0.5)$ Mean(SD)	$N(0,1)$ Mean(SD)	$N(0,2)$ Mean(SD)
Idle	0.198 4(4.04e – 03)	0.198 3(5.38e – 03)	0.198 4(9.74e – 03)
1	0.302 3(5.01e – 03)	0.302 3(7.02e – 03)	0.303 2(11.52e – 03)
2	0.203 8(3.83e – 3)	0.203 9(5.69e – 03)	0.203 5(9.73e – 03)
3	0.195 5(2.56e – 03)	0.195 5(4.56e – 03)	0.195 0(9.14e – 03)
4	0.099 9(1.67e – 03)	0.099 9(3.24e – 03)	0.100 4(5.97e – 03)

Table 6: The profiling result of reducing the number of samples.

Task	$N(0,2)$ $n = 1000$ Mean(SD)	$N(0,1)$ $n = 1000$ Mean(SD)	$N(0,1)$ $n = 500$ Mean(SD)
Idle	0.193 0(29.89e – 03)	0.194 4(18.69e – 03)	0.207 8(26.29e – 03)
1	0.317 9(36.98e – 03)	0.316 5(20.39e – 03)	0.343 6(31.10e – 03)
2	0.192 4(30.17e – 03)	0.191 8(17.86e – 03)	0.199 0(23.96e – 03)
3	0.197 1(29.02e – 03)	0.197 6(13.98e – 03)	0.193 9(21.03e – 03)
4	0.102 4(20.15e – 03)	0.101 0(9.72e – 03)	0.056 4(10.93e – 03)

6. Related Work

Except for the hardware-based performance profiling, there are another two representative methods. One is an intrusive method [19, 24, 25] that needs to modify the application source code to add instrumentation code for collecting data. This method requires the authority to access the source code, rebuild the application source code, and redeploy this version into the system. These requirements are impractical. Moreover, these intrusive methods can disturb the application's behaviour, bringing other questions about the collected data's validity.

Another one is the simulator-based method [26, 27] using the processor simulator that models the real processor's architecture. It collects processor performance data by using the simulator to execute the application. This method can yield detailed data on a processor like the pipeline stalls and cache line behaviours. However, not every processor would have its corresponding simulator that is provided by its manufacturers. The simulator would also be tens of times slower than running on real processors, making performance profiling costly.

The task granularity profiling is useful in code profiling and hot execution path detection [28]. Identifying program hot spots can support runtime optimization [29, 30]. The application anomaly [4] or stragglers detection [31, 32] also needs the information from the task-level.

For more accuracy to count the events into specific tasks, there are instruction-oriented profiling techniques [33]. The profiling interruption is triggered by an instruction related dimension. A detailed record of interesting events and pipeline stage latencies in the out-of-order processor is collected. Trace-based profiling [34, 35] has a similar design to follow a running pipeline to record the running states. But they are not useful in improving the confidence in the hardware counter accuracy.

Many pieces of literature analyze accuracy from the probability theory. Chen [36] proposed the ProbPP method for analyzing the probabilities on the execution paths of the multithreaded programs. Yan and Ling [37] used the probability model on the memory level parallel analysis to estimate the maximum number of cache misses. But they did not use the probability model to prove the accuracy of hardware-based profiling technique.

7. Conclusion

In this paper, we analyze the hardware-based profiling technique's mechanism using the probability theory and design an analysis model to simulate the profiling process. The setting of the simulated model follows the characteristics of workloads running in a live environment. The simulation results validate our probability deduction result and show that the expectation value has nonbiased property, and the variance is small. It is expected that this work can improve confidence in the profiling accuracy and broaden the relevant research directions.

Data Availability

The simulated running state data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] J. Leverich and C. Kozyrakis, "Reconciling high server utilization and sub-millisecond quality-of-service," in *Proceedings of the Ninth European Conference on Computer Systems, ser. EuroSys '14*, April 2014.

[2] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: improving resource efficiency at scale," in *Proceedings of the 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 450–462, Portland, Oregon, June 2015.

[3] A. Li, L. Gu, and K. Xu, "Fast anomaly detection for large data centers," in *Proceedings of the 2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pp. 1–6, Miami, FL, USA, December 2010.

[4] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni, "Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change," in *Proceedings of the 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pp. 452–461, Anchorage, AK, USA, June 2008.

[5] B. Gregg, "The flame graph," *Communications of the ACM*, vol. 59, no. 6, pp. 48–57, 2016.

[6] K. Ishizaki, "Analyzing and optimizing java code generation for Apache spark query plan," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, ser. ICPE '19*, pp. 91–102, Association for Computing Machinery, Mumbai India, April 2019.

[7] M. C. Merten, A. R. Trick, C. N. George, J. C. Gyllenhaal, and W.-m. W. Hwu, "A hardware-driven profiling scheme for identifying program hot spots to support runtime optimization," in *Proceedings of the 26th Annual International Symposium on Computer Architecture, ser. ISCA '99*, pp. 136–147, IEEE Computer Society, Atlanta Georgia USA, May 1999.

[8] P. Gralka, C. Schulz, G. Reina, D. Weiskopf, and T. Ertl, "Visual exploration of memory traces and call stacks," in *Proceedings of the 2017 IEEE Working Conference on Software Visualization (VISSOFT)*, pp. 54–63, Herrsching am Ammersee, Germany, September 2017.

[9] P. Krishnamurthy, R. Karri, and F. Khorrami, "Anomaly detection in real-time multi-threaded processes using hardware performance counters," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 666–680, 2020.

[10] V. M. Weaver, D. Terpstra, H. McCraw et al., "Papi 5: measuring power, energy, and the cloud," in *Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 124-125, Austin, TX, USA, April 2013.

[11] P. J. Mucci, S. Browne, C. Deane, and G. Ho, "Papi: a portable interface to hardware performance counters," in *Proceedings of the Department of Defense HPCMP Users Group Conference*, pp. 7–10, 1999.

[12] B. Gregg, "Perf examples," 2020, http://www.brendangregg.com/perf.html.

[13] A. Pandey, D. Tesfay, and E. Jarso, "Performance analysis of intel ivy bridge and intel broadwell microarchitectures using intel vtune amplifier software," in *Proceedings of the 2018 2nd International Conference on Inventive Systems and Control (ICISC)*, pp. 423–426, Coimbatore, India, Jan 2018.

[14] D. Zaparanuks, M. Jovic, and M. Hauswirth, "Accuracy of performance counter measurements," in *Proceedings of the 2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 23–32, Boston, MA, USA, April 2009.

[15] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Producing wrong data without doing anything obviously wrong," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS XIV*, pp. 265–276, Association for Computing Machinery, Washington DC USA, March 2009.

[16] W. Korn, P. J. Teller, and G. Castillo, "Just how accurate are performance counters?" in *Proceedings of the Conference 2001 IEEE International Performance, Computing, and Communications Conference (Cat. No.01CH37210)*, pp. 303–310, Phoenix, AZ, USA, April 2001.

[17] Wikipedia, "Bernoulli trial," 2020, https://en.wikipedia.org/wiki/Bernoulli_trial.

[18] P.-H. Chen, C.-T. King, Y.-Y. Chang, and S.-T. Tseng, "Multiprocessor system-on-chip profiling architecture: design and implementation," in *Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems*, pp. 519–526, Shenzhen, China, 2009.

[19] K. Shankar and R. Lysecky, "Non-intrusive dynamic application profiling for multitasked applications," in *Proceedings of the 2009 46th ACM/IEEE Design Automation Conference*, pp. 130–135, San Francisco California, July 2009.

[20] M. Maxwell, S. Moore, and P. Teller, "Efficiency and accuracy issues for sampling vs. counting modes of performance monitoring hardware," in *Proceedings of the DoD High Performance Computing Modernization Program's User Group Conference*, 2002.

[21] J. Guo, Z. Chang, S. Wang et al., "Who limits the resource efficiency of my datacenter: an analysis of alibaba datacenter traces," in *Proceedings of the 2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*, pp. 1–10, Phoenix, Arizona, USA, June 2019.

[22] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, and R. Hundt, "Google-wide profiling: a continuous profiling infrastructure for data centers," *IEEE Micro*, vol. 30, no. 4, pp. 65–79, 2010.

[23] B. G. Amidan, T. A. Ferryman, and S. K. Cooley, "Data outlier detection using the Chebyshev theorem," in *Proceedings of the 2005 IEEE Aerospace Conference*, pp. 3814–3819, IEEE, Big Sky, MT, USA, March 2005.

[24] H. K. Cho, T. Moseley, R. Hank, D. Bruening, and S. Mahlke, "Instant profiling: instrumentation sampling for profiling datacenter applications," in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 1–10, Shenzhen, China, February 2013.

[25] S. Horovitz, Y. Arian, M. Vaisbrot, and N. Peretz, "Non-intrusive cloud application transaction pattern discovery," in *Proceedings of the 2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 311–320, Milan, Italy, July 2019.

[26] Z. Du, B. Xia, F. Qiao, and H. Yang, "System-level evaluation of video processing system using simplescalar-based multicore processor simulator," in *Proceedings of the 2011 Tenth International Symposium on Autonomous Decentralized Systems*, pp. 256–259, Tokyo, Japan, March 2011.

[27] H. Ham, C. Park, J. Kim, J. Kim, and J. Cho, "Processor power simulator for low power code generation using look up table," in *Proceedings of the 2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, pp. 550–553, Seogwipo, South Korea, November 2011.

[28] M. C. Merten, A. R. Trick, E. M. Nystrom, R. D. Barnes, and W. W. Hwu, "A hardware mechanism for dynamic extraction and relayout of program hot spots," in *Proceedings of the 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, pp. 59–70, Vancouver, BC, Canada, June 2000.

[29] M. C. Merten, A. R. Trick, C. N. George, J. C. Gyllenhaal, and W. W. Hwu, "A hardware-driven profiling scheme for identifying program hot spots to support runtime optimization," in *Proceedings of the 26th International Symposium on*

Computer Architecture (Cat. No.99CB36367), pp. 136–148, Atlanta, GA, USA, May 1999.

[30] N. P. K. Gorti and A. K. Somani, "Runtime optimization utilizing program structure," in *Proceedings of the 2012 18th International Conference on Advanced Computing and Communications (ADCOM)*, pp. 46–53, Bangalore, India, December 2012.

[31] M. Solaimani, M. Iftekhar, L. Khan, and B. Thuraisingham, "Statistical technique for online anomaly detection using spark over heterogeneous data from multi-source vmware performance data," in *Proceedings of the 2014 IEEE International Conference on Big Data (Big Data)*, pp. 1086–1094, Washington, DC, USA, October 2014.

[32] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, "Statistical techniques for online anomaly detection in data centers," in *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pp. 385–392, Dublin, Ireland, May 2011.

[33] J. Dean, J. E. Hicks, C. A. Waldspurger, W. E. Weihl, and G. Chrysos, "Profileme: hardware support for instruction-level profiling on out-of-order processors," in *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pp. 292–302, Research Triangle Park, North Carolina, USA, December 1997.

[34] H. Pirzadeh, S. Shanian, A. Hamou-Lhadj, and A. Mehrabian, "The concept of stratified sampling of execution traces," in *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, pp. 225-226, Kingston, ON, Canada, June 2011.

[35] P. Crowley and J. . Baer, "On the use of trace sampling for architectural studies of desktop applications," in *Proceedings of the Workload Characterization: Methodology and Case Studies. Based on the First Workshop on Workload Characterization*, pp. 15–24, Dallas, TX, USA, November 1998.

[36] Y. Chen, "Platform independent analysis of probabilities on execution paths of multithreaded programs," in *Proceedings of the 2013 IEEE/ACIS 12th International Conference on Computer and Information Science (ICIS)*, pp. 397–404, IEEE, Niigata, Japan, June 2013.

[37] Y. Yan and M. Ling, "Accelerating the analytical modeling of memory level parallelism by the probability analysis," in *Proceedings of the 2019 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pp. 1–6, Victoria, B.C., Canada, August 2019.