Use of a Levy Distribution for Modeling Best Case Execution Time Variation

Jonathan C. Beard and Roger D. Chamberlain*

Dept. of Computer Science and Engineering
Washington University in St. Louis, St. Louis, Missouri, USA
{jbeard,roger}@wustl.edu

Abstract. Minor variations in execution time can lead to out-sized effects on the behavior of an application as a whole. There are many sources of such variation within modern multi-core computer systems. For an otherwise deterministic application, we would expect the execution time variation to be non-existent (effectively zero). Unfortunately, this expectation is in error. For instance, variance in the realized execution time tends to increase as the number of processes per compute core increases. Recognizing that characterizing the exact variation or the maximal variation might be a futile task, we take a different approach, focusing instead on the best case variation. We propose a modified (truncated) Levy distribution to characterize this variation. Using empirical sampling we also derive a model to parametrize this distribution that doesn't require expensive distribution fitting, relying only on known parameters of the system. The distributional assumptions and parametrization model are evaluated on multi-core systems with the common Linux completely fair scheduler.

1 Introduction and Background

Understanding the performance of software systems is often accomplished with the help of stochastic queueing models. These models typically require knowledge of the distributions for inputs such as arrival rate and service rate for compute kernels within an application. Directly influencing the aforementioned values is the execution time distribution of each compute kernel. Complete knowledge of the distribution is generally futile for modern systems. Yet understanding it, however incompletely, is critical to selecting proper model formulations. When understanding complex phenomena, it is often the practice to find a useful bound. We contend that the minimal expected execution time variation of a system, or best case execution time variation (BCETV), is such a bound. By forecasting BCETV for a particular software and hardware combination, we hope to improve the a *priori* knowledge of a models' applicability. This paper introduces the use of a modified Levy distribution for characterizing the BCETV of short execution,

* This work was supported by Exegy, Inc. Washington Univ. and R. Chamberlain receive income based on the license of technology by the university to Exegy, Inc.

A. Horváth and K. Wolter (Eds.): EPEW 2014, LNCS 8721, pp. 74–88, 2014.

compute bound kernels. A closed form expression for the probability density function as well as it's first and second moments are derived. The distributional assumptions and model are evaluated via empirical evaluation.

Several references simply assume that the distribution of a series of execution times should be Gaussian [7]. Other works (e.g., [10]) have shown some examples of successive execution times that are not Gaussian with any high probability. Other phenomena such as worst case execution time have been modeled with the Gumbel distribution [5]. Empirically measured execution time noise for a minimal workload of "no-op" instructions (the difference between the nominal and measured execution times, plotted in Figure 1) exhibits a heavily skewed distribution. Simply assuming a Gaussian distribution (green line) overestimates the mass of one tail while underestimating the other. A Gumbel distribution (blue line) is arguably even worse. Some might posit that a Gamma distribution is a good fit, however the support exists only for $x \geq 0$ which fits neither reality or our use case as a noise model. A modified Levy distribution (red line, exact modifications to be discussed) is plotted against the same data, visually it is the best fit to the observed data.

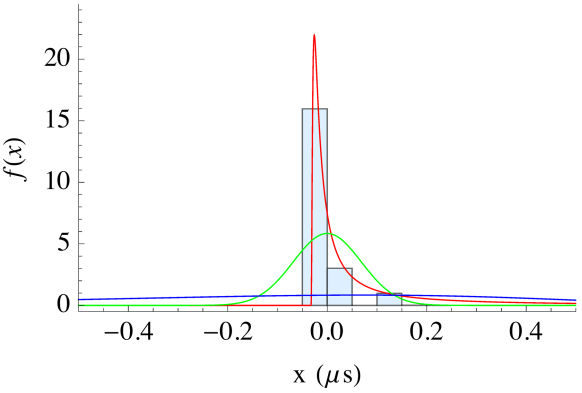
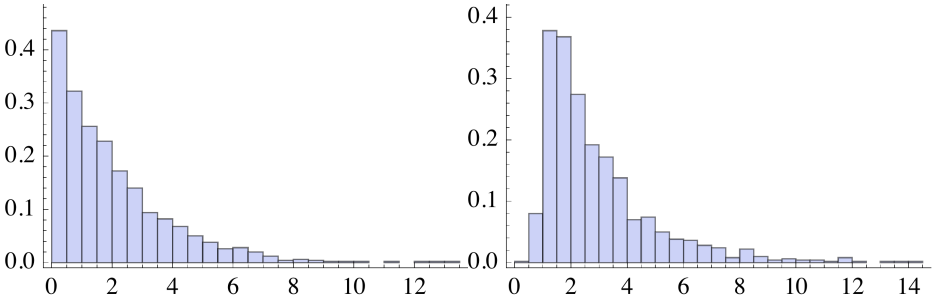


Fig. 1. Histogram of the discrete PDF for a simple "no-op" workload execution time absolute error (light blue bars) in μs plotted against the PDFs of a fitted Gaussian distribution (green line), a Gumbel distribution (blue line) and a modified Levy distribution (red line). Visually it is easy to see that the modified Levy distribution is the best fit for this data set.

Many performance models require details of the inner workings of the target processor [6]. When empirical evaluation is performed, often the results obtained are still uncertain. How well did the empirical evaluation sample the distribution of execution times? Even when detailed knowledge is assumed, or empirical evaluation is performed, there is still uncertainty in the values obtained. Causes of this execution time uncertainty can include cache behavior, interrupts, scheduling uncertainty as well as countless other factors. Distributional uncertainty can lead to poor stochastic model performance. Instead of focusing on the worst or

even average case, our approach focuses on the best case and what this bound can do for the model decision making process. As an example, Figure 2(a) shows the distribution that a simple $M/M/k$ queueing model assumes for its inter-arrival distribution whereas Figure 2(b) might be closer to reality given a noisy system. One application of BCETV is to estimate how close a models' input assumptions will line up with reality assuming a best case variance. This could allow quick rejection of models whose assumptions are violated.



(a) Exponential distribution ($\lambda = .5$) (b) Exponential distribution ($\lambda = .5$) with additive Gaussian noise

Fig. 2. Stochastic models often make simplifying distributional assumptions about the modeled system. One common assumption is that of a Poisson arrival process (i.e., exponentially distributed inter-arrival times). This assumption is often violated by the "noise" that the hardware, operating system and environment impose upon the application. Figure 2(a) shows a nominal exponential distribution, while Figure 2(b) shows an example of a realized distribution.

BCETV is the minimum variation (error relative to the mean) which can be expected from any single observation of execution time. We assert that the minimal "no-op" workload can be used as a proxy for determining BCETV for short execution, compute bound kernels. In principle, these workloads should be quite deterministic in execution time, but clearly are not. We will show that the distribution of BCETV experienced by these workloads represents a reasonable lower bound "noise" model for nominal execution time. Utilizing empirical data, the modified Levy distribution is revised in terms of system parameters (i.e., processes per core, nominal execution time). Evidence is provided that the modified Levy distribution is a good match for BCETV, especially as the number of processes per core grows.

2 Methodology

The motivation to use a Levy distribution to model the best case execution time variation (BCETV) came from empirical observation. Ultimately we must justify that decision by comparing model predictions to experimental observations.

To that end we start by describing the process through which these data are collected. This is followed by the description of the modified Levy distribution that we propose to use, and how to parametrize it.

2.1 Synthetic Workload

Our focus is the uncertainty in execution time of a running process due to factors other than the process itself. As such, we use an intentionally simple nominal workload so that the observed variation is due not to the application itself, but to other system related factors (e.g., operating system, hardware, etc.). Our nominal workload is the execution of a fixed number of null operations or "no-op" instructions. Aside from no instructions at all, we assume that a null operation is the least taxing instruction. It follows from this logic that a series of null operations should present the most consistent execution time out of any real executable instruction sequence.

One aspect under study is how changing the nominal workload time changes the observed variation in actual execution times. In order to produce a workload of "no-op" instructions that is calibrated to a specific nominal execution time we use sequences of instructions of various lengths which are timed and then used as input for regression to produce an equation for the number of instructions to use for each nominal execution time. Calibration timing is performed while the timed process is assigned to a single core and executing with no other processes.

In theory any duration of workload could be created using this method, however in practice the file sizes become prohibitively large proportionate with the frequency of the processor and the desired running time (e.g., platorm A from Table 2 requires approximately 10 million"no-op" instructions for each second of execution time). Other approaches that reduce the file size could be used such as looping over a calibrated number of "no-op" instructions, however we've chosen to use the simpler aforementioned approach because it reduces the possibility of variation due to other factors, such as branching. Our method also assumes that cache pre-fetching will eliminate virtually all instruction cache misses which should then have no appreciable effect on the actual run time. One concern with huge numbers of instructions is that translation lookaside buffer (TLB) misses might increase the observed variation. With TLB misses we would expect an increase in the overall observed variation with longer duration executions with a random pattern (dependent on other processes operating on the same core, TLB algorithm, etc.). As we will show below, this is not the case; more variation is observed for short execution times.

2.2 Hardware, Software, and Data Collection

At the core of our efforts is empirical data collection. The distributional choice and subsequent verification depend upon it. To enable empirical data collection, a test harness was created that executes the synthetic "no-op" workloads while varying numbers of processes per core, nominal execution times, and execution platforms. As the synthetic workload processes are executing, the parameters in

Table 1 are collected. In order to reduce the possibility that results gleaned from this study might be an artifact of a particular hardware platform or operating system, two different platforms are used as shown in Table 2 (two of platform A and seven of platform B). All platforms support a version of the Linux completely fair scheduler [9] which will be exclusively used during data collection.

Table 1. Experimental Parameters

Parameter	Symbol
Nominal Execution Time	t_N
# Processes per Core	p
Voluntary Context Swaps	v
Non-Voluntary Context Swaps	nv
Actual Execution Time	t_A
Execution Time Noise ($t_A - t_N$)	Δ

Table 2. Hardware and Operating Systems

Label	Processor	Operating System
A	Intel E3 1220	Fedora 19, Linux Kernel v. 3.10.10
B	2 x AMD Opteron 2431	CentOS 5.9, Linux Kernel v. 3.0.27

Each data point collected consists of the dimensions outlined in Table 1. Nominal execution times vary from $0.25\mu s$ through $3.7ms$ with observations at an interval of $0.25\mu s$ throughout the range. The number of workload processes per core varies from 1 through 20 processes. Each sharing and nominal execution time pair is executed 1000 times to ensure a good distribution sample. The synthetic workloads are run on one of two of platform A or on one of seven of platform B from Table 2. In total 100+ million observations are made. Two factors limited the range of viable execution times: the lower bound on timer resolution (see below) and the memory needed to generate workloads of longer lengths (disk to store and physical memory to compile).

Generated data is divided into two sets. The first, a "training" set (of size 10^6) is segregated using uniform random sampling. The rest of the data is used for model evaluation and will be referred to as the "evaluation" set. We specifically want to judge the applicability of this noise model to multiple hardware types and operating systems using the same scheduler.

There has been much discussion about the best and most accurate way to time a section of code [3]. There are many methods including processor cycle counters and operating system "wall-clock" time. Given our reliance on empirical data for modeling and evaluation we feel it is important to cover how our timing measurements are made. In many cases, the use of a simple time stamp counter is effective assuming that the process will never migrate to another core. Another issue to consider is frequency scaling which can lead to wildly inaccurate timings

when utilizing the processor cycle counter. To alleviate some of those concerns and provide a relatively universal timing interface we developed a system timer thread that utilizes the x86 time stamp counter instruction on a single reference core to update a user space timer. When a process or thread requests the current time, an in-lined function copies the current time struct which has two time references and it compares the two times. If they are the same then the calling code can be sure that the time has been fully updated and the function returns, if not the code loops until the values match. Frequency scaling is turned off for the time update thread.

This timing method has several advantages: (1) it is entirely in user space, (2) it is lock-free, and (3) it is monotonic even when the timed thread is shifted to a new core. Two concerns with this approach stem from the copying operation. How long does it take to copy the timer struct on a target system and what happens when there are multiple Non-uniform Memory Access (NUMA) nodes? To test the latter of these concerns a benchmark was constructed to ascertain how long a copy operation takes when the copy is from the same NUMA node as the calling process and when the timer thread and requester thread are on differing NUMA nodes. The results of this are shown in Figure 3 for platform B from Table 2. What we've found is that reading memory allocated on a NUMA node other than the one closest to the time requesting process the access times can vary somewhat. To eliminate this issue, all subsequent experiments only use a single NUMA node.

A common problem with highly accurate timing via software is determining what is ground truth. Short of an external atomic clock, there are only varying degrees of truth. In order to determine the precision and accuracy of our measurements, a standardized workload is created with a series of "no-op" instructions of varying lengths. Each "no-op" length is timed using either the

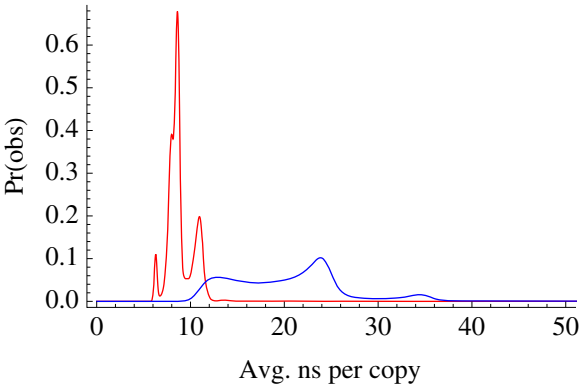


Fig. 3. Smooth histogram of 10^6 data points each representing timed averages of 500 copy operations, first on the same NUMA node (red line) and then across different NUMA nodes (blue line). The performance of a copy on the same NUMA node seems to be much more consistent.

x86 `rdtsc` instruction or the POSIX.1-2001 `clock_gettime()` function. Figure 4 shows the inter-quartile range (25^{th} to 75^{th} percentiles) difference of each timing measurement as a function of the length of the "no-op" instruction sequence, This plot informs us about the stability of the two timing methods. The system call to `clock_gettime()` is more stable than the `rdtsc` instruction, especially for these small workloads. A hypothesis as to why it is more stable is that the measurement of actual workload time is small relative to the time it takes to perform a system call. To test this theory the timing methods themselves are timed by executing five hundred of each method (either the `rdtsc` insn. or the `clock_gettime()` function) and using the average execution time of all five hundred to extrapolate the time to execute a single instruction. In this experiment the `rdtsc` instruction is used as the reference timer on platform A from Table 2. As expected (and shown in Figure 5) the system call to `clock_gettime()` takes almost 3× as long on average compared to the x86 `rdtsc` instruction. For this reason, we exclusively use the `rdtsc` instruction for all empirical timing measurements in this work.

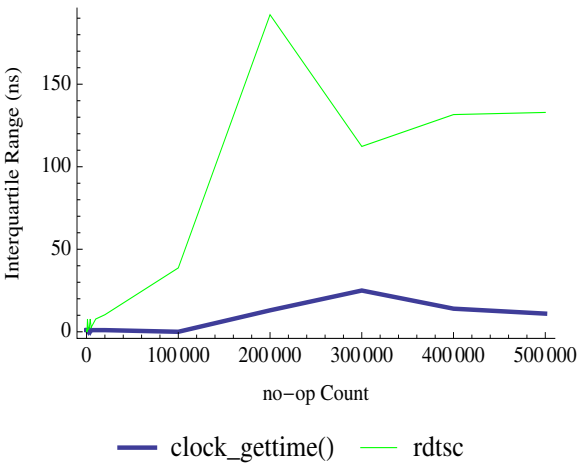


Fig. 4. Interquartile range difference (IQRD = $75^{th} - 25^{th}$) in nanoseconds for the times measured for each set of "no-op" instructions (number instructions listed on x-axis). Each instruction length was executed $10^6\times$ for each method. The IQR gives a visual representation to the stability of measurements for these two timing methods.

2.3 Distribution

Figure 1 provides a qualitative indication that a Levy distribution makes a good choice for modeling the noise present in execution times of a nominally fixed minimal workload (the proxy for BCETV). Quantitatively Table 3 summarizes the p-values for each distribution (higher is better), the table shows the minimum, maximum and mean values. The Levy distribution is the only distribution with

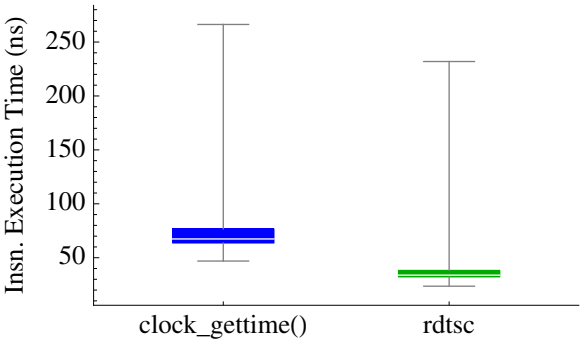


Fig. 5. Box and whisker plot showing a speed comparison of the `rdtsc` x86 assembly instruction compared to a `clock_gettime()` call to the Linux real-time clock. The `rdtsc` instruction's $25^{th} - 75^{th}$ percentiles are almost identical at the nanosecond scale. The `clock_gettime()` function overall takes much more time (approximately 3×).

Table 3. Summary of Anderson-Darling Goodness of Fit Test

Distribution	Min	10^{th}	50^{th}	90^{th}	Max	Mean
Gaussian Distribution	0	1.17×10^{-15}	1.68×10^{-14}	3.0×10^{-5}	.719	.002
Levy Distribution	0	2.11×10^{-15}	2.15×10^{-14}	.038	.803	.025
Gumbel Distribution	0	8.93×10^{-16}	1.97×10^{-14}	6.39×10^{-06}	.357	.002
Cauchy Distribution	0	3.89×10^{-16}	1.34×10^{-14}	.002	.771	.009

greater than 10% of the data having a p-value $\geq .01$. Next, we will quantitatively describe the modified Levy distribution that we use.

Realized execution time is the sum of a nominal (mean) execution time and a noise term. If the nominal execution time is represented by a random variable N, and the noise is represented by a random variable V, then the realized execution time $R \sim N + V$. The goal of this work is to find a distribution to represent a lower bound for V which we term BCETV.

The Levy distribution [11] has a closed form probability density function (PDF, shown in Equation (1)), however in general it has no defined moments. Observations from the empirical data lead to a solution. Whereas the tail of the Levy distribution is infinite, the noise present within the real execution times has a limit. The limit, not surprisingly, is correlated with both the nominal length of execution and the number of processes assigned to a single compute core. This leads to the consideration of a variation of the Levy distribution that is truncated at a point represented by a new parameter Ω. The modified Levy distribution is defined using the truncation method of Equation (2) as Equation (3) where $F(\cdot)$ is the CDF of the PDF denoted by $f(\cdot)$. (Note: erfc(x) is the compliment of the Error Function, $1 - \text{erf}(x)$, and E_i is the exponential integral function [1].) In order to make the equations more concise, $w = \frac{\beta}{2(\alpha-\Omega)}$ and $z = \frac{\beta}{2(x-\alpha)}$.

$$f_L(x;\alpha,\beta) = \frac{e^{-z}\,(2z)^{3/2}}{\sqrt{2\pi}\beta} \tag{1}$$

$$f_{mL}(x;\alpha,\beta,\Omega) = \frac{f_L(x;\alpha,\beta)}{F_L(\Omega;\alpha,\beta) - F_L(-\infty;\alpha,\beta)} \tag{2}$$

$$f_{mL}(x;\alpha,\beta,\Omega) = \frac{\sqrt{\beta}e^{-z}}{\sqrt{2\pi}(x-\alpha)^{3/2}\left(\mathrm{erfc}\sqrt{-w}\right)} \tag{3}$$

Restricting the use of the modified Levy distribution, mL, to $x \leq \Omega$ and $x > \alpha$ leads to a closed form expression of the mean as shown in Equation (4). Lastly, a variance is also defined as Equation (5).

$$\mu_{mL}[\alpha,\beta,\Omega] = \frac{\beta\Gamma\left(-\frac{1}{2},-w\right)}{2\sqrt{\pi}\mathrm{erfc}\left(\sqrt{-w}\right)} + \alpha \tag{4}$$

$$\sigma^2_{mL}[\alpha,\beta,\Omega] = \frac{(\alpha-\Omega)^2\left(\frac{E_{\frac{5}{2}}(-w)\left(\sqrt{2\pi}\beta^{3/2}\mathrm{erfc}\left(\sqrt{-w}\right)+3(\Omega-\alpha)^{3/2}\left(4e^w-3E_{\frac{5}{2}}(-w)\right)\right)}{\sqrt{\Omega-\alpha}} + 4(\alpha-\Omega)e^{2w}\right)}{2\pi\beta\mathrm{erfc}^2\left(\sqrt{-w}\right)} \tag{5}$$

Our next task is to determine an appropriate parametrization of the modified Levy distribution. We accomplish this task by fitting a model to empirical measurements. The "training" data are sorted into groups W_{p,t_N} which are indexed by the number of processes sharing a core, p, and the nominal execution time, t_N (see Equation (6a)). Within each group, the execution time noise is computed for each observation as in Equation (6b).

$$W_{p,t_N} = \bigcup_i obs_i \in p, t_N \tag{6a}$$

$$\Delta = t_A - t_N \tag{6b}$$

Separately for each group W, Maximum Likelihood (ML) techniques are used to find the best parameters for a number of distributions, including the modified Levy distribution that we are proposing. The quality of the distributions' fit to the empirical data is judged via an Anderson-Darling [2] goodness of fit test as shown in Table 3 (chosen because of the weight given to the tails of the distributions compared to other tests such as the Kolmogorov-Smirnov test [4]).

2.4 Parameterization

While the ML techniques used above can yield a parametrization for the modified Levy distribution that is well matched to the data, in general ML techniques are quite computationally expensive and also require substantial support to be effective. An alternative is to redefine the modified Levy distribution parameters, α, β, and Ω, in terms of a subset of the parameters in Table 4.

The selection of parameters from Table 4 is reduced based on the intuition that the nominal execution time and number of processes sharing a core will have the largest impact on the true execution time. Given the design of the minimal compute kernel, it is expected (and confirmed) that there are zero voluntary context swaps allowing the variable to be discarded. A Pearson correlation coefficient between the target variables and the training data (Table 4) quantifies the intuition about the remaining parameters.

Table 4. Correlation Between Target Predictors

	nv	t_N	p
Δ	.508	.771	-.0056

Table 4 summarizes the correlations within the training set between the execution time noise, Δ, and the other parameters. For the entire training set there is a weak correlation between the number of processes sharing a core and the execution time noise. There is a strong correlation between the nominal execution time and the noise. Not shown is the co-variance between the non-voluntary context swaps and the number of processes per core which implies a lack of independence. The models considered therefore consist only of the two independent parameters p and t_N.

Using simple linear regression to find coefficients for p and t_N that best fit the parameters for α, β and Ω found by ML, the relationships in Equation (7) are found with the following assumptions: $p \in \mathbb{Z} \wedge p > 1$ and $t_N \in \mathbb{R} \wedge t_N > 0$. To parametrize the modified Levy distribution as defined in Equation 3, several other constraints must be added, namely: Equation (7a) is expected to have a negative range for the entire domain, Equation (7b) is positive for the entire domain and Equation (7c) is greater than α for the entire domain. A limitation of these equations is the range of data used to create them. It is expected that α will not continue to decrease as $t_N \to \infty$ and the β, Ω parameters probably have limitations as well; however these equations are supported through the range of data specified in Section 2.2.

$$\alpha = 4.75 \times 10^{-9} p - 0.220 t_N \tag{7a}$$

$$\beta = 4.19 \times 10^{-10} p + 0.007 t_N \tag{7b}$$

$$\Omega = 3.19 \times 10^{-6} p + 0.742 t_N \tag{7c}$$

Using Equations (7), which predict α, β and Ω based on p and t_N, the PDF and mean of the modified Levy distribution can now be described in terms of p and

t_N as shown in Equations (8) and (9), respectively. The variance of Equation (8) is a straightforward algebraic manipulation of Equation (5).

$$f_{mL}(x;p,t_N) = \frac{8.2\times10^{-6}(\sqrt{1p+1.6\times10^{7}t_N})e^{\frac{0.044p+6.98\times10^{5}t_N}{p-4.6\times10^{7}t_N-2.1\times10^{8}x}}}{(-4.8\times10^{-9}p+0.22t_N+x)^{3/2}\text{erfc}\left(\sqrt{0.003-\frac{0.003p}{p+3.02\times10^{5}t_N}}\right)} \tag{8}$$

$$\mu_{mL}[p,t_N] = \frac{(1.2\times10^{-10}p+0.002t_N)\Gamma\left(-\frac{1}{2},0.003-\frac{0.003p}{p+3.02\times10^{5}t_N}\right)}{\text{erfc}\left(\sqrt{0.003,-\frac{0.003p}{p+3.02\times10^{5}t_N}}\right)} + 4.8\times10^{-9}p-0.22t_N \tag{9}$$

3 Results

How well does the modified Levy distribution approximate the actual BCETV observed while executing a nominally deterministic compute bound kernel? We will focus our evaluation on the PDF expressed in terms of processor sharing, p, and nominal execution time, t_N, presented above as Equation (8).

The Anderson-Darling (AD) goodness of fit test of Table 3 is, frankly, not very promising. Yet, we already know from Table 3 that the modified Levy is the best out of the listed distributions used to model the training data. It is not at all surprising that our overall p-value when using AD is not very high ranging from 0 to 0.73. What is welcome news is that AD is not the only metric available, as it is relatively ineffective at identifying portions of the parameter space that have a good vs. a poor fit.

A second measure of how well the modified Levy distribution fits empirical data is how well the moments match. When comparing the mean of the empirical data sets to that predicted by Equation (9), the differences are effectively below our ability to differentiate based on the techniques described in Section 2.2 (i.e., the difference is $\ll 10^{-12}s$). Comparing the variance for the modified Levy vs. the empirical measurements results in an r-squared value of 0.69, which indicates a reasonable degree of correlation between model and data, but the alignment between the two is clearly not perfect.

While the above quantitative assessments of the modified Levy distribution's match with empirical measurements make it clear that the model is far from perfect, we must keep in mind the fact that modelers can often exploit individual models that are far from perfect, and given prior use of models that are much more divergent from reality than our proposed modified Levy distribution there is the real potential for benefit from the ability to use a distribution that more closely matches empirical measurement than previous models and also has relatively simple expressions of its first two moments.

We continue the assessment of how well the modified Levy distribution characterizes the noise in execution times by presenting QQ-plots for three distributions

relative to the empirical data (see Figure 6). The first column of plots is the modified Levy distribution of Equation (8), the second column is a Gaussian distribution, and the third column is a Cauchy distribution. The latter two distributions are parametrized by fitting to the data using ML techniques. For each distribution, 4 distinct QQ-plots are shown, separating the processor sharing variable, p, into quartiles. The first (top) row represents the range $1 \leq p \leq 5$, the second row represents the range $6 \leq p \leq 10$, the third row represents the range $11 \leq p \leq 15$, and the fourth (bottom) row represents the range $16 \leq p \leq 20$.

First consider the results in Figure 6(g) and (j), which include the modified Levy distribution and significant processor sharing. Here, we see quite nice alignment between the model and the empirical data, the best evidence yet that the modified Levy is a good execution time noise model. Next consider the results in Figure 6(a) and (d), which include the modified Levy distribution and little processor sharing. In this case, there is reasonably good alignment at the low end of the range, but the empirical data has slightly less variation than the model at the high end of the range. Finally, note that the alignment between model and empirical data is noticeably worse for both the Gaussian and the Cauchy distributions across the entire range of p.

From the above we conclude that the modified Levy distribution is a relatively good proxy for BCETV. The distribution of BCETV can in turn be used in many ways. To demonstrate the utility of BCETV, we explore the mean queue occupancy (MQO) of a single queue system when noise is added as in Figure 2. The single queue system operates as two threads with one way communication that is designed to have an exponentially distributed inter-arrival and service time distribution (i.e., workload is dependent upon an exponential random number source). A simple model for MQO is the $M/M/1$ queueing model, it expects the inter-arrival times to be exponentially distributed. We posit that the farther from this distribution the actual system is, the greater the model's predictions will differ from empirical reality. The Kullback-Leibler (KL) divergence [8] is a measure of the divergence between two distributions (zero being a perfect match). We are interested in how far the distributional lower bound as predicted by the convolution of the exponential distribution and Equation 8 differs from that expected by the $M/M/1$ MQO model. With a divergence of zero we should expect to find a very close match between modeled and experimental MQO. At higher divergences (the exact amount is an open question) we don't expect the $M/M/1$ model to be very accurate. Figure 7 is a summary of median KL divergences (y-axis) separated by percent model accuracy (calculated as $\frac{|\text{modeled MQO} - \text{measured MQO}|}{\text{measured MQO}} \times 100$, x-axis) for 6000+ separate executions of the single queue system described above on the platforms shown in Table 2. It shows that lower KL divergence (green bar) between the expected exponential and that convolved with the BCETV distribution, is associated with more accurate MQO predictions. This implies that BCETV can be used as a predictor for model choice (at least with a Markovian arrival process).

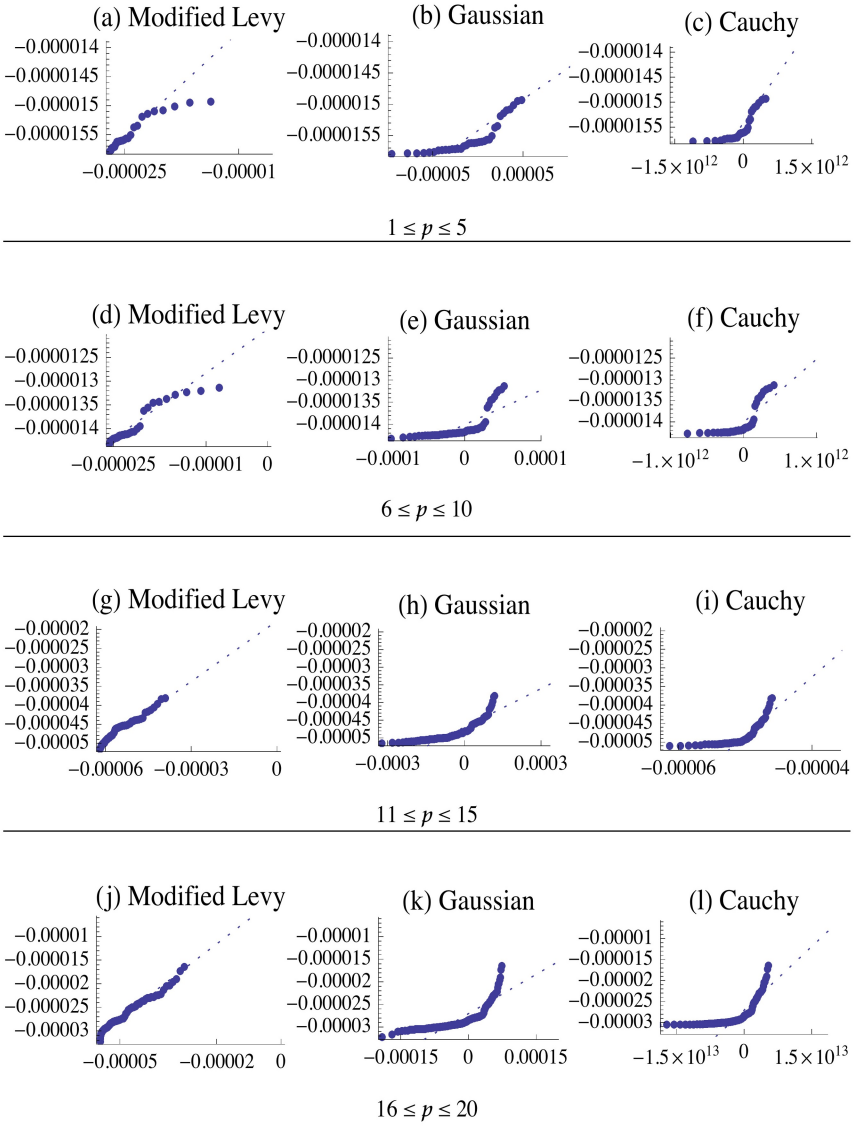


Fig. 6. QQ-plots comparing empirical data (vertical axis) to the analytic distributions (horizontal axis). The dashed line shows the ideal response.

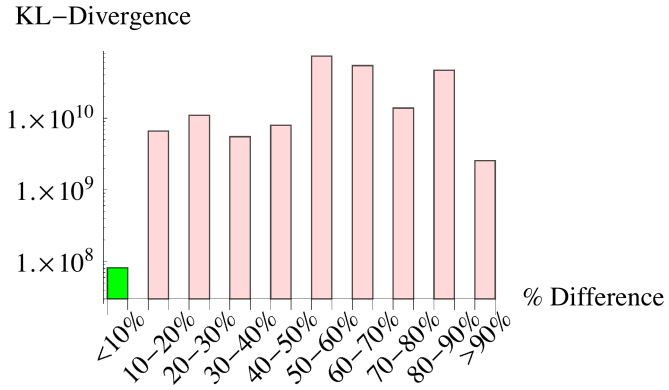


Fig. 7. The y-axis shows the median KL divergence between the $M/M/1$'s expected exponential inter-arrival distribution and the lower bound predicted by convolving the exponential distribution with Equation 8. The x-axis is the percent difference between the mean queue occupancy predicted by an $M/M/1$ model and the actual measurements from a single queue system designed to have a perfectly exponential workload. The lowest KL divergence (green bar) is associated with more accurate predictions.

4 Conclusions and Future Work

We've demonstrated a noise model that appears to work far better than a simple Gaussian assumption, in fact far better than multiple other distributions. It has also been shown to work for at least two differing platform types (see Table 2) using the same fair scheduling algorithm. First, we've shown expressions for the PDF and first two moments of a Levy distribution that has been modified to have bounded moments. Through empirical data collection, a model is derived that can be used to parametrize the modified Levy distribution relatively well without resorting to computationally expensive parameter fitting.

In Figure 6 we showed how well the quantiles of the modified Levy distribution match to the quantiles of the empirical data. We've also noted that the fit between the model and empirical data gets better as more processes are added per core. This is in keeping with our original assumption that a single process on a single core should exhibit its native distribution, in our case purely deterministic, or close to the nominal mean, t_N. The models demonstrated here are only validated over the range of empirical data that we've collected. For future work, we would like to extend the parameter estimators for $p > 20$ and higher nominal execution times t_N.

One concern with our approach is also one of its strengths, that it is based on wide empirical sampling. Is this noise model really applicable to multiple hardware types, or were our choices simply judicious? Could other parameters in addition to nominal execution time, t_N, and the number of processes per core, p, provide a better estimate on other platforms (e.g., alternative instruction sets). One potential application of this noise model is as a minimal expected noise for

all workloads. We are also interested in investigating variability in execution time due to the nature of the application itself, e.g., including the effects of caching, branching, etc.). We will investigate these options in future work.

References

[1] Abramowitz, M., Stegun, I.A.: Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables. Courier Dover Publications (2012)
[2] Anderson, T.W., Darling, D.A.: Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes. The Annals of Mathematical Statistics, 193–212 (1952)
[3] Bryant, R., O'Hallaron, D.R.: Computer Systems: A Programmer's Perspective. Prentice Hall (2003)
[4] Chakravarty, I., Roy, J., Laha, R.: Handbook of Methods of Applied Statistics. McGraw-Hill (1967)
[5] Edgar, S., Burns, A.: Statistical analysis of WCET for scheduling. In: Proc. of 22nd IEEE Real-Time Systems Symposium, pp. 215–224 (2001)
[6] Engblom, J., Ermedahl, A.: Pipeline timing analysis using a trace-driven simulator. In: Proc. of 6th Int'l Conf. on Real-Time Computing Systems and Applications, pp. 88–95 (1999)
[7] Jain, R.: The Art of Computer Systems Performance Analysis. John Wiley & Sons (1991)
[8] Kullback, S., Leibler, R.A.: On information and sufficiency. The Annals of Mathematical Statistics, 79–86 (1951)
[9] Li, T., Baumberger, D., Hahn, S.: Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin. ACM SIGPLAN Notices 44(4), 65 (2009)
[10] Mazouz, A., Touati, S.A.A., Barthou, D.: Study of variations of native program execution times on multi-core architectures. In: Proc. of Int'l Conf. on Complex, Intelligent and Software Intensive Systems, pp. 919–924 (2010)
[11] Nolan, J.: Stable Distributions: Models for Heavy-tailed Data. Birkhäuser (2003)