January 2009

Statistical-Based WCET Estimation and Validation

Jeffery Hansen
Software Engineering Institute

Scott A. Hissam
Software Engineering Institute

Gabriel A. Moreno
Software Engineering Institute, gmoreno@sei.cmu.edu

Published In

Proceedings of the 9th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis.

STATISTICAL-BASED WCET ESTIMATION AND VALIDATION

Jeffery Hansen[1] Scott Hissam[1]
Gabriel A. Moreno[1]

Abstract
In this paper we present a measurement-based approach that produces both a WCET (Worst Case Execution Time) estimate, and a prediction of the probability that a future execution time will exceed our estimate. Our statistical-based approach uses extreme value theory to build a model of the tail behavior of the measured execution time value. We validate our approach using an industrial data set comprised of over 150 sampled components and nearly 200 million sample execution times. Each trace is divided into two segments, with one used to make the WCET estimate, and the second used check our prediction of the fraction of future execution time samples that exceed our WCET estimate. We show that compared to WCET estimates derived from the worst-case observed time, our WCET estimates significantly improve the ability to predict the probability that our WCET estimate is exceeded.

1. Introduction

In the analysis of many systems, it is necessary to know the Worst-Case Execution Time (WCET) of the tasks in the system. Traditionally, WCET estimation has been based on static-analysis techniques. The source code or disassembled binary executable of the task is analyzed to determine the time required for the longest path through the code. Modern processor techniques such as caching and predictive branching make this approach very difficult[3]. Efforts to overcome such difficulty require modeling and simulation of complex hardware architectures to augment these static-analysis code techniques[12, 7]. As a result, measurement-based approaches have become more popular.

In measurement-based approaches, estimates of the WCET of a task are made by running a series of sample executions of the task in isolation and directly measuring its execution time. This can be either the execution time of the whole task, or the execution time of individual basic blocks (segments of straight-line code with no intervening branches) of the task. In the simplest approaches, the task is executed and measured for a set number of times and the longest observed time, possibly scaled by an ad hoc "safety factor", is used as the WCET. While this approach is simple, there is no systematic way to determine the appropriate scale factor or predict how good the WCET estimate will be.

Some approaches use a combination of analytical and measurement-based methods. For example a tool called pWCET [14, 13] breaks a piece of code down into basic blocks, uses measurment to compute an execution time distribution, then uses a set of techniques to compute joint distributions based on the type of control structures in the code. The authors describe a technique which can produce

[1]Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.
{jhansen,shissam,gmoreno}@sei.cmu.edu

WCET predictions with no independence assumptions on the execution times of the individual basic blocks. The primary weakness of this approach is that the execution time is modeled directly by an empirical distribution function. Since WCET estimation is concerned primarily with the tail behavior of the distribution, a very large number of samples are required to obtain an accurate model.

While ideally we would like to find a WCET that is guaranteed never to be exceeded, this is not practical with a measurement-based approach. Instead the goal is to find WCET estimates that have very low and predictable probabilities of being exceeded (e.g., 10^{-5}).

1.1. Extreme Value Theory

In this paper, we present and evaluate a WCET estimation algorithm based on Extreme Value Theory (EVT). Extreme Value Theory[5, 1] is a branch of statistics for analyzing the tail behavior of a distribution. It allows us to reason about rare events, even though we may not have enough sample data to actually observe those rare events. Example applications include hydrology, finance and insurance.

1.2. WCET Estimation Using EVT

By using EVT, we can estimate WCET values that acheive a target exceedence probability (the probability that a future sample will exceed our WCET estimate). This is in contrast with the more primitive approach of simply using the highest observed execution time of a task over a set of trials for which we can not predict how well it will do in future invocations[10].

In one recent approach to using EVT for WCET estimation[2], execution time measurements are first fit to the Gumbel distribution[5] using an unbaised estimator. A WCET estimate is then made using a pertaining excess distribution function. The problem with this approach is that it incorrectly fits raw execution time data to the Gumbel distribution. The Gumbel and other EVT distributions are intended to model random variables that are the maximum (or minumum) of a large number of other random variables. In general, this is not the case for execution time measurements. An additional problem is that there does not appear to be any goodness-of-fit test to ensure the estimated parameters actually fit the measured data.

Our WCET estimation algorithm is also based on EVT. The most important change is that rather than attempting to fit the Gumbel distribution directly to the sample execution times, we use the method of block maxima[1], grouping the sample execution times into blocks and fitting the Gumbel only to the maximum values from each of the blocks. This ensures that the samples used to estimate the Gumbel distribution are samples from a maximum of random variables. In addition, we use a Chi-squared test to ensure that the sample data correctly matches the distribution we fit.

2. About the Data

In order to validate our WCET estimation approach, we use execution time traces from a set of 154 periodic tasks incorporating over 200 million combined execution time samples. These tasks make up a commercial device implemented as a PowerPC based embedded system running the VxWorks real-time operating system with many built-in monitoring and control interfaces. Two different instances of this device were stressed and measured by two different teams in two different laboratories using the same measurement procedure and device configuration over the course of a few days to obtain the observed execution times of these tasks[4]. Approximately 125 minutes of trace data was collected

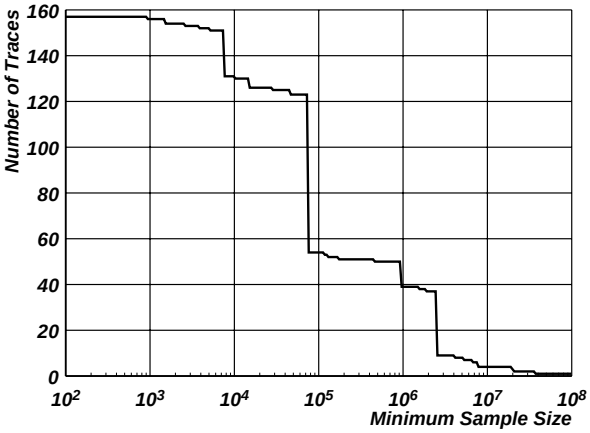

Figure 1. Number of Traces Meeting a Minimum Sample Size

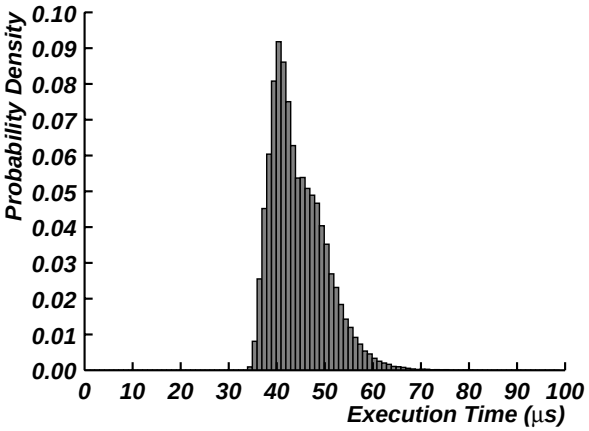

Figure 2. Execution Time Distribution for Task T_1

Property	Value
Trace Length	15 min.
Number of Samples	300,793
Execution Time Maximum	113.8 μs
Execution Time Mean	43.68 μs
Execution Time Std.	6.08 μs

Table 1. Execution Time Summary for Task T_1 Estimation Trace Data

for each task in 25 five-minute runs. Each trace is divided into two parts: an estimation part and a validation part. The estimation part is used for making WCET estimates and consists of the first 15 minutes of the trace data. The validation part is used for evaluating our WCET estimates and consists of the final 110 minutes of each trace. We used this partitioning between the estimation and validation parts because it gave us a good trade off between traces for which the WCET estimate converged while keeping the amount of estimation data low.

Since the periods of the tasks are different, the number of sample execution times collected in each trace is different. In order to ensure a sufficient number of samples for estimation and validation, we exclude those traces with less than 75,000 samples. We chose this value because it represents a large break in the number of traces exceeding this threshold (see Figure 1), and empirical evidence suggest that the 9,000 samples in the estimation part of the trace are usually sufficient to make a good estimate. A total of 122 tasks met this threshold.

2.1. Trace Characteristics

There are no data dependent loops in any of the tasks we measured. Because of this, we expect the execution times to be dependent only on factors such as the state of the cache, predictive branching and other processor effects. To generalize our results to tasks with data-dependent loops it would be necessary to combine our approach, (using it to estimate the execution time of basic blocks) with code analysis.

Table 1 lists basic statistics of the 15 minute estimation part of a representative trace which we will call T_1. The execution time distribution for this trace is shown in Figure 2. As can be seen from the graph, the distribution peaks near the mean and falls of with rapidly decreasing probability density

above 60μs.

2.2. Collection Methodology

Execution time measurements were collected from an embedded system (excluding the operating system itself) with over one million source lines of C and C++ code. Each task in the system, when running in steady state, was performed by an identifiable task body in the source code. Each task body was tagged by inserting WindView[11] user event markers in the source to identify when a task started working on a job and when that job was complete. As such, these markers delineated in the observed trace when a task was actually running in the CPU and when that task body had yielded. With these markers, the actual periods and execution times for each job the task performed could be measured.

Once the source code was tagged, the measurement procedure was to bring the device into steady state (post-startup phase) and collect measurement samples from the device using the same industrial testing methods and standards adopted by the device's manufacturer. This meant that the device was subjected to the maximum allowable data rates on all input channels for a fixed interval of time. After the sample data was collected, the traces from the jobs performed by the tasks were analyzed to calculate each jobs' execution time (the time occurring between each user event marker taking into account any preemption or blocking that may have occurred during the jobs execution). A more detailed discussion of the measurement approach can be found in [4].

3. Extreme Value Theory

In principle, we could collect enough sample execution time data to construct an empirical distribution function $\hat{F}(x)$ and then simply use the inverse CDF to compute an estimated WCET as $\hat{\omega} = \hat{F}^{-1}(1 - p_e)$ where p_e is the desired probability of exceeding our estimate. The problem with this approach is that in order to have a good model of the right tail behavior we would need a vast amount of data. Furthermore, we could never generate an estimate higher than the highest observed value. Extreme Value Theory (EVT)[5] gives us a way to reason about this tail behavior without the vast amount of data required by a brute-force approach.

Given a random variable $Y = \max X_1, ..., X_n$ formed from the maximum of a set of n i.i.d. (independent identically distributed) random variables X_i, EVT tells us the distribution of Y will converge to the Generalized Extreme Value (GEV) distribution as n goes to infinity. This is analogous to the role the central limit theorem plays in the sum of a large number of random variables. Properties of the distribution of the base random variables X_i determine to which of three forms the distribution of Y will converge. The three forms are: Type I (Gumbel) – When the underlying distribution has a non-heavy upper tail (e.g., Normal), Type II (Frechet) – When the underlying distributions has a heavy upper tail (e.g., Pareto), and Type III (Weibull) – When the underlying distributions has a bounded upper tail (e.g., Uniform).

In applying EVT to the WCET estimation problem, we assume that the execution time distribution (e.g., Figure 2) has a non-heavy tail for most tasks. This implies the GEV distribution will converge to the Type I or Gumbel form which we assume for the reminder of this paper. We test and confirm this hypothesis in Section 5..

The Gumbel distribution has two parameters: a location parameter μ and a scale parameter β. It has

1. *Set the initial block size b to 100.*
2. *If the number of blocks $\lfloor N/b \rfloor$ is less than 30, then stop (not enough samples to generate an estimate).*
3. *Segment execution times $x_1, ..., x_N$ into blocks of b measurements.*
4. *For each of the $\lfloor N/b \rfloor$ blocks find the maximum values $y_1, ...y_{\lfloor N/b \rfloor}$ where $y_i = \max(x_{(i-1)b+1}, x_{(i-1)b+2}, ..., x_{ib})$.*
5. *Estimate the best-fit Gumbel parameters μ and β to the block maximum values $y_1, ...y_{\lfloor N/b \rfloor}$.*
6. *If the Chi Squared fit between the block maximum values $y_1, ...y_{\lfloor N/b \rfloor}$ and the Gumbel parameters μ and β does not exceed a 0.05 confidence value, then double the value of b and go back to Step 2.*
7. *Return WCET estimation $\omega = F_G^{-1}((1-p_e)^b)$ where F_G^{-1} is the percent-point function (Equation 1) for a Gumbel with parameters μ and β, b is the block size and p_e is target sample exceedance probability.*

Figure 3. WCET Estimation Algorithm

the cumulative distribution function (CDF) $F_G(y) = e^{-e^{-\frac{y-\mu}{\beta}}}$ and the percent-point function:

$$F_G^{-1}(q) = \mu - \beta \log(-\log(q)) \tag{1}$$

4. WCET Estimation

In order to apply these EVT techniques to execution time samples, we must construct sample data from a random variable that is formed from the maximum of another set of random variables. We group the execution time samples into size b blocks of consecutive samples, then choose the maximum value from each block to construct a new set of sample "block maximum" values. That is, given a set of raw execution time measurements $x_1, ..., x_N$, we construct the set of block maximums $y_1, ..., y_{\lfloor N/b \rfloor}$ where y_i is the maximum x_j value between $x_{(i-1)b+1}$ and x_{ib}. Left over samples not filling a block are discarded.

By selecting only the block maximum values, the algorithm presented here focuses on the samples of most interest to us. In general the larger the block size b, the better fit we will get to the Gumbel distribution. However, there is a tradeoff since a increasing b will result in fewer blocks and thus fewer sample values.

Our algorithm for making WCET estimates takes as input a set of N execution time samples, $x_1, ..., x_N$, and a target exceedance probability p_e, and returns a WCET estimate for which we predict that future invocations of that task will exceed this estimate with probability p_e. The smaller the value we choose for p_e, the larger the estimated WCET value will be. Since the underlying algorithm assumes that execution times come from an unbounded distribution (but with rapidly decreasing probability density for large values), we can not generate a WCET value for $p_e = 0$.

Figure 3 shows an outline of our WCET estimation algorithm. This algorithm is discussed in greater

detail in the following sections. We will use the 15 minute estimation part of the trace for Task T_1 mentioned in Section 2.1. as an example.

4.1. Steps 1-4: Blocking of Samples

The goal of Steps 1 through 4 is to select a subset of the original data points that represent the upper tail of the behavior. This is done by grouping the data into blocks of b samples and discarding all but the maximum value from each block. Samples at the end of the trace that do not completely fill a block are discarded. We make the assumption that all execution time samples are independent and thus the blocking method chosen will not affect the statistical properties of the blocks.

The selection of b is a trade-off between the quality of fit to the Gumbel distribution, and the number of samples that can be used in making that fit. Generally, the larger the value we choose for b, the more likely the block maximum values will follow a Gumbel distribution, but the fewer samples we will have available to use in the estimation of the Gumbel parameters. For example, if we have 50,000 execution time samples with which to make an estimate, there will be 500 block maximum values when the block size is 100. But if we use a block size of 1,000, then there will be only 50 block maximum values to use in the estimation of the Gumbel parameters.

We use the generally accepted value of 30 as the minimum number of samples (block maximum values) to use in estimating a distribution. This limit also bounds the size to which b can grow. Since b is doubled until the goodness-of-fit test is passed (at Step 6), it is possible that $\lfloor N/b \rfloor$ can fall below 30 and thus we must stop the algorithm without generating a WCET estimate. In this case, the only remedy is to either collect more execution time samples or use another WCET estimation technique.

4.2. Step 5: Parameter Estimation

In Step 5, we compute the parameters of the Gumbel distribution of the block maximum values. We do this by applying linear regression to the QQ-plot of these values. A QQ-plot[1], or quantile plot, is a plot of the empirical quantile values of sample data against the quantiles of the standard form of a target distribution. For Gumbel quantiles, this is done by plotting the points:

$$(-\log(-\log(1 - \frac{i}{\lfloor N/b + 1 \rfloor})), y_i^{[s]}), \quad i = 1..\lfloor N/b \rfloor$$

where $\{y_1^{[s]}, ..., y_{\lfloor N/b \rfloor}^{[s]}\}$ are the block maximum values sorted from lowest to highest. If the block maximum values follow a Gumbel distribution, then the points on the QQ-plot will form a straight or nearly straight line. The slope and intercept of the best-fit line through these points can be used as estimators for the Gumbel μ and β parameters, respectively. While using linear regression to compute a best fit line is not the only method to estimate the Gumbel parameters, it is quicker and easier than methods such as maximum likelihood estimation.

An example QQ-plot for Task T_1 is shown in Figure 4. The dots represent the block maximum values, and the line represents the Gumbel fit to the data. Notice that we get a good linear fit to the data and we can find the Gumbel location parameter value $\mu = 61.72$ from the y-intercept, and the Gumbel scale parameter value $\beta = 5.95$ from the slope.

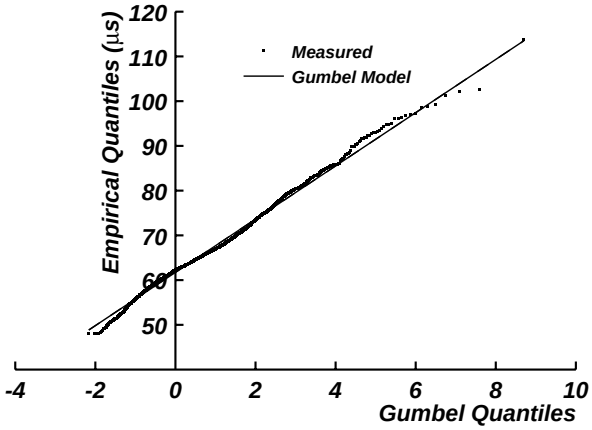

Figure 4. QQ-Plot of Block Maximum Values with Gumbel Quantiles for Task T_1

4.3. Step 6: Verifying Goodness-of-Fit

In Step 6, we verify the goodness of fit between the sampled block maximum values and the estimated Gumbel parameters using a Chi Squared test. The test is performed by grouping the sample values into M bins and then comparing the number of samples that fall into each bin O_i with the expected number of samples that fall into bin E_i (computed by multiplying the total number of samples by the difference in the Gumbel CDF at the edges of the bin) using the Chi Squared formula:

$$\Xi^2 = \sum_{i=1}^{M} \frac{(O_i - E_i)^2}{E_i} \tag{2}$$

The resulting value, along with the number of degrees of freedom is then checked against a Chi Squared Table to see if it is significant. The number of degrees of freedom in this case is given by $M - 3$. This includes two for the number of Gumbel parameters we are estimating, and one for the last bin who's value is fixed after all other bin values have been determined. From the Chi Square table we can find the critical Chi Square value for a level of significance p. p represents the probability that a Chi Square distributed random variable will exceed that critical value. Typically a match at the $p = 0.05$ is considered acceptable.

In our algorithm, we create the bins by dividing the range between the smallest and largest block maximum value into $N_b/30$ equally spaced bins, where $N_b = \lfloor N/b \rfloor$ is the number of blocks. This will create an initial set of bins for which the average number of observations falling into them is 30. However, we do not let the number of bins fall below six. Since the Chi squared test is sensitive to bins with low numbers of observations, we collapse adjacent bins with less than five observations. This is done by scanning the blocks from the lowest to the highest combining blocks with less than five observations in it until there are five or more observations. Again, we always retain at least six bins.

For the example task T_1 at a block size of $b = 100$ there were 3007 block maximum values which resulted in an initial set of 100 bins for the Chi Squared test. Figure 5 shows the number of block maximum values that fell in each of the bins (bars) along with the expected values for each of these bars (the solid line). After collapsing small bins, we got a total of 62 bins and computed a Chi Square value of 236.0. Since this exceeds the required Chi-Squared value of 77.93 for $59 = 62 - 3$ degrees

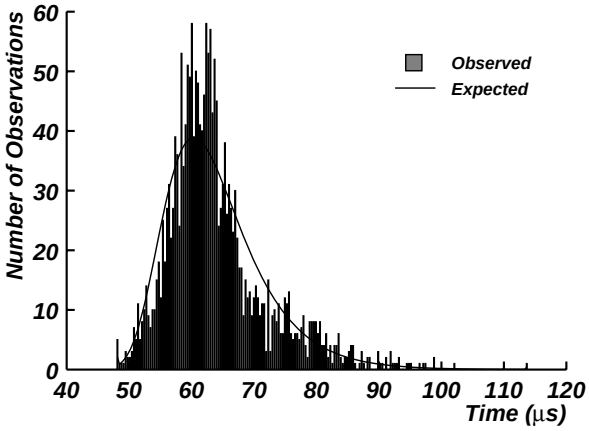

Figure 5. Fit of Task T_1 Block Maximums Against Gumbel for $b = 100$

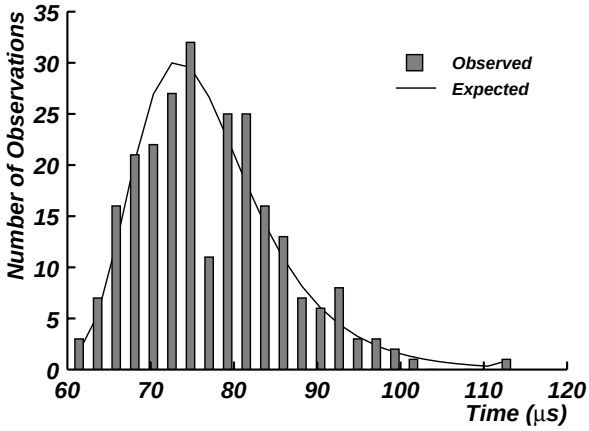

Figure 6. Fit of Task T_1 Block Maximums Against Gumbel for $b = 400$

of freedom at the $p = 0.05$ confidence value, we must reject the hypothesis that the observed data matches a Gumbel distribution.

Since the goodness-of-fit test failed, Step 6 calls for us to double the block size and try again. Doing this repeatedly, we find that at a block size of $b = 400$, we obtain a Chi Squared value that does not require us to reject a match between the estimated Gumbel parameters and the block maximum data. In this case we got Gumbel parameters of $\mu = 70.0$ and $\beta = 6.23$. Note that these values are not directly comparable with the $b = 100$ case since they are estimated from a different set of values.

In applying the goodness-of-fit test to the $b = 400$ case, we initially get 25 bins which collapsed into 19 bins after we combine the bins with fewer than five observations. This results in 16 degrees of freedom and thus a critical Chi squared value of 26.3 for a fit at the $p = 0.05$ confidence level. Figure 6 shows the fit between the observed bin counts (bars) and the expected bin counts (line) for this case.

4.4. Step 7: Estimating WCET Value

The final step is to use the computed and verified Gumbel parameters and the exceedance probability p_e to estimate the WCET. This is done using the Gumbel percent-point function (Equation 1). This function returns the block maximum value for which there is a probability q that a measured block maximum will not exceed this value. We can compute q as:

$$q = (1 - p_e)^b \tag{3}$$

That is, q is the probability that all b samples in the block are below the WCET value. Combining Equations 3 and 1 gives us the equation:

$$\omega = \mu - \beta \log(-\log((1 - p_e)^b)) \tag{4}$$

for computing the WCET estimate ω in terms of the block size b, the exceedance probability p_e, and the Gumbel parameters μ and β. In the example for Task T_1, at a target exceedance probability of $p_e = 10^{-4}$ we get the WCET estimate:

$$\omega = 70 - 6.23 \log\left(-\log\left((1 - 10^{-4})^{400}\right)\right) = 90.05\mu\text{s}$$

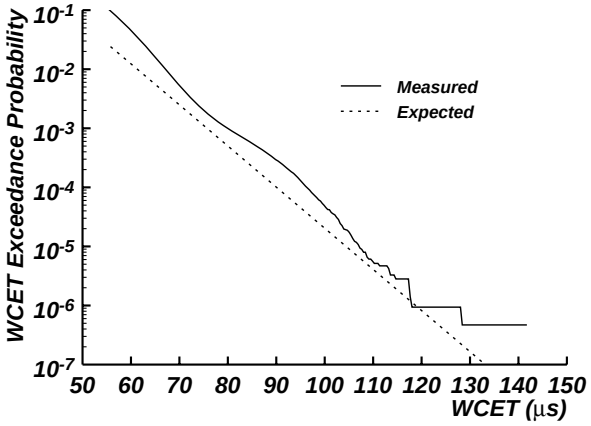

Figure 7. Predicted vs. Measured Exceedance Probability for Task T_1

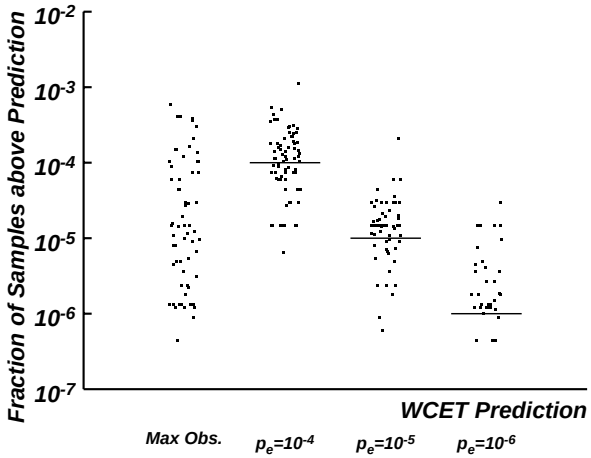

Figure 8. Distribution of Exceedance Probability for All Tasks

5. Validation

In applying our WCET estimation algorithm to the 15 minute estimation part of the 122 traces described in Section 2., 75 of the traces passed the Chi squared test and yielded a WCET estimate. These tasks will be the focus of our analysis of the quality of the WCET estimates. However, we point out that by increasing the amount of data used by the estimation algorithm from 15 minutes to 30 minutes, the number of traces for which we could make an estimate was increased to 95.

5.1. Single-Task Validation

A comparison between the predicted and the measured exceedance probability for Task T_1 example is shown in Figure 7. The predicted exceedance probability curve was generated from Equation 4 using the parameters estimated in the previous sections($\mu = 70.0$, $\beta = 6.23$, and $b = 400$). The measured curve was generated by sweeping through WCET values and measuring the number of samples that fall above each value. A total of about 2.2 million execution times where used to validate the WCET estimates for T_1.

We can see that the predicted exceedance probability matches well with the measured probability. In particular, the slopes (on a log scale) match extremely well. However, there is some divergence, particularly in the lower-right corner of the graph where the measured curve takes on a "stair step" appearance. This "stair step" effect is due to sample size limitations at the right-hand side of the graph. Even with 2.2 million samples, single values passing below the WCET estimate can cause large changes when measuring events near the 10^{-6} probability level.

We also examined the curves for the predicted versus observed exceedance probabilities for the other 74 tasks for which we generated a WCET estimate. We found results similar to those for task T_1. While a slight positive or negative bias between the observed and predicted exceedance probability curves was present in most traces, the slopes generally matched extremely well.

5.2. Task Set Validation

We compared the WCET estimates made using our WCET estimation algorithm with a "Maximum Observed" method in which the highest observed value in the estimation part of the trace was taken

as the WCET estimate.

Figure 8 shows a set of four point clouds showing the fraction of validation execution time samples (the last 110 minutes of each trace) that exceeded the WCET estimate for the Maximum Observed method and our WCET estimation method at three different p_e values. Each dot in each point cloud represents a single trace and its fraction of WCET exceedances. It can be seen that in the "Maximum Observed" case, the fraction of WCET exceedances is very unpredictable and has a high variance. In contrast, the variability for our EVT-based approach is significantly smaller and clustered around the predicted value indicated by the horizontal lines. Note that while the measured exceedance probabilities for the $p_e = 10^{-6}$ case appears to be biased toward exceedance probabilities higher than 10^{-6}, there are actually 42 tasks for which the measured exceedance probability was zero and thus can not be seen on the graph. This $p_e = 10^{-6}$ case is in fact near our limit to measure the WCET exceedances given the amount of validation data we have.

6. Conclusion

The most important aspect of a measurement-based WCET estimation method is the ability to predict the exceedance probability. By using a large collection of traces, we have shown that our EVT-based method can produce WCET estimates for which the exceedance probability is more predictable and controllable than using the maximum observed execution time.

References

[1] BEIRLANT, J., GOEGEBEUR, Y., SEGERS, J., and TEUGELS, J., "Statistics of Extremes: Theory and Applications," Wiley Press, 2004

[2] EDGAR, S., BURNS, A., "Statistical Analysis of WCET for Scheduling," in *Proceedings of the 22nd Real-Time Systems Symposium*, pg. 215-224, 2001

[3] HILLARY, N. and MADSEN, K., "You can't control what you Can't Measure, OR Why it's Close to Impossible to Guarantee Real-time Software Peformance on a CPU with on-chip cache," in *Proc. of the 2nd Int. Workshop on WCET Anal.*, June 2002

[4] HISSAM, S. A., MORENO, G. A., PLAKOSH, D., SAVO, I. and STELMARCZYK, M., "Predicting the Behavior of a Highly Configurable Component Based Real-Time System", in Proc. of 20th Euromicro Conf. on Real-Time Systems (ECRTS 08), 2008

[5] GUMBEL, E.J., "Statistics of Extremes", Columbia University Press, 1958

[6] LEAHY, K., "Efficient Estimation of Tighter Bounds for Worst Case Execution Time of Programs", Matser's Thesis, Dep. of Comp. Sci. and Eng., Washington Univ. in St. Louis, December 2004

[7] LIM, S.S., BEA, Y. H., JANG, G. T., RHEE, B.D., MIN, S.L., PARK, Y.C., SHIN, H. and KIM. C.S., "An accurate worst case timing analysis for RISC processors". In IEEE Real-Time Systems Symposium, pages 97–108, December 1994

[8] PERRONE, R., MACÊDO, R. and LIMA, G., "Estimation Execution Time Probability Distributions in Component-Based Real-Time Systems," in *Proceedings of the 10th Brazilian Workshop on Real-Time and Embedded Systems*, May 2008

[9] PETTERS, S., "How much Worst Case is Needed in WCET Estimation?", in *Proceedings of the 2nd International Workshop on Worst-Case Execution Time Analysis*, June 2002

[10] SCHAEFER, S., SCHOLZ, B., PETTERS, S. M. and HEISER, G., “Static Analysis Support for Measurement-based WCET Analysis”. In 12th IEEE Int. Conf. on Embed. and Real-Time Comp. Sys. and App., Work-in-Progress Session, August 2006

[11] WIND RIVER SYSTEMS, INC., “Tornado Getting Started Guide, 2.2, Windows Version”, Wind River Systems, Inc., Alameda, CA, 2002

[12] ZHANG, N., BURNS, A. and NICHOLSON, M., “Pipelined Processors and Worst Case Execution Times”, Real Time Systems, Vol. 5, pp. 319-343, 1993

[13] BERNAT, G., COLIN, A. and PETTERS, S., “pWCET: A Tool for Probabilistic Worst-Case Execution Time Analysis of Real-Time Systems”, in *Proc. of the ACM 2003 Conf. on Languages, Compilers, and Tools for Embedded Sys. (LCTES)*, 2003

[14] BERNAT, G., COLIN, A. and PETTERS, S., “WCET Analysis of Probabilistic Hard Real-Time Systems”, in *Proc. of the 22nd IEEE Real-Time Sys. Symp.*, 2002