# A Trace-Based Statistical Worst-Case Execution Time Analysis of Component-Based Real-Time Embedded Systems

Yue Lu[1], Thomas Nolte[1], Iain Bate[2], and Liliana Cucu-Grosjean[3]
[1]Mälardalen Real-Time Research Centre (MRTC), Västerås, Sweden
[2]Department of Computer Science, University of York, York, United Kingdom
[3]INRIA Nancy-Grand Est, Nancy, France
yue.lu@mdh.se

***Abstract*—This paper describes the tool support for a framework for performing statistical WCET analysis of real-time embedded systems by using bootstrapping sampling and Extreme Value Theory (EVT). To be specific, bootstrapping sampling is used to generate timing traces, which not only fulfill the requirements given by statistics and probability theory, but also are robust to use in the context of estimating the WCET of programs. Next, our proposed statistical inference uses EVT to analyze such timing traces, and computes a WCET estimate of the target program, pertaining to a given predictable probability. The evaluation results show that our proposed method could have the potential of being able to provide a tighter upper bound on the WCET estimate of the programs under analysis, when compared to the estimates given by the referenced WCET analysis methods.**

## I. Introduction

The design of a Real-Time System (RTS) revolves heavily on a model known as a task schedule, which allocates the CPU resource to executing tasks. Many different scheduling algorithms [1] are invented, all of which depend on some temporal properties pertaining to each task. One such property is the Worst-Case Execution Time (WCET), which is the longest time a piece of software will execute without any interrupts, on a particular hardware platform. A lot of research has been done in the field of WCET analysis, and a good overview can be found in [2]. Techniques to perform WCET analysis can broadly be categorized as *Static WCET Analysis* (SA), *Measurement-Based WCET Analysis* (MBA), *Hybrid Measurement-Based WCET Analysis* (HMBA) and *statistical WCET analysis* [3]. Because of high complexity, WCET analysis tools are often not able to find the exact WCET [4].

Both SA and HMBA have to get the access to the source code or the ability to dissemble the binary in order to construct the program model used in analysis. However, both of these assumptions are challenged when the source code is not available, and/or even disassembling of its object code is disallowed, along with the lines of protecting intellectual property. This is very likely to be a typical case about the construction of real-time embedded systems which are built up on a number of prefabricated pieces of software called *components*, which is a pervasive area of interest. Only the relocatable object code of a component is shipped to the client so that it can later be linked into overall application. Consequently, existing SA and HMBA techniques to WCET analysis are not ideal in above context, and the only alternative to obtain a WCET estimate is through MBA.

The problems of MBA lie: 1) MBA may not guarantee to find the actual WCET in the general case, and may consequently underestimate the WCET and, 2) To remedy the situation, a *safe margin* or an ad hoc *safety factor* is usually to be added to the WCET estimate given by MBA, which for instance is from engineering wisdom from previous projects. Nonetheless, there is no systematic way to determine the appropriate safety margin or predict how good the WCET estimate will be, and if the actual WCET is bounded. Moreover, when the High Water-Mark Time (HWMT) given by MBA equals the actual WCET in fact but safety margin is too excessive, there could also lead to be severe underutilization of system resources.

In this paper, we propose a statistical WCET analysis method based on measurements, which does not require the disassembly or source code. Typically, our method is comprised of a novel sampling mechanism (which tackles with some problems raised when statistics is used in WCET analysis) and a statistical inference about computation of a WCET estimate of the program under analysis, with a certain predictable probability. The evaluation results show that our proposed method could have the potential of providing a tighter upper bound on the WCET of the program under analysis, when compared to the referenced measurement-based WCET analysis methods.

## II. Problem Overview and Our Contributions

Throughout this work, we particularly study the problems about using statistics in WCET analysis proposed by Griffin in [5]. Such problems are critical for which we mention them next.

The first problem argued by Griffin is that the Gumbel distribution, along with any other continuous probability distribution, makes an assumption that all values are possible. However, the control flow graph of a program shows that a program cannot terminate at any point. Therefore, it is not realistic to use a *continuous random variable* i.e., the

Gumbel Max distribution in EVT, to model the execution time of the program.

Moreover, Griffin also introduced the second problem about using statistics in WCET analysis, i.e., the *i.i.d. assumption in EVT*. To be specific, EVT makes the assumption required by statistics and probability theory, i.e., the observations (or analysis samples) have to be *independent and identically distributed* (i.i.d.). Unfortunately, the execution times of programs usually are not independent or identically distributed, for instance, due to the dependencies between different states of the data structure in the program. So for some systems, such i.i.d. assumption is not a practical assumption.

Clearly, before any statistics is used in WCET analysis, the above two problems have to be tackled. Nonetheless, the analysis samples collected by using traditional sampling mechanism, i.e., end-to-end measurements, can never ever fulfill the i.i.d. assumption. A new way of collecting *qualified* analysis samples (or *individuals*) which could be used by statistics is thereby necessary. The contributions of this paper concern both aspects:

1) In Section III-A, we propose a novel sampling mechanism by employing the simple random sampling technique with bootstrapping to collect qualified and robust analysis samples (i.e., timing traces), which can be used by our proposed statistical inference.
2) We propose a statistical WCET analysis method namely *RapidET* which computes a WCET estimate of the program under analysis based around analyzing timing traces, by using EVT.

Our research problem can be defined as follows: We are given a program $p$ which is running on a specific hardware platform $S$. The goal of the problem is then to find an estimate, under a certain probability of being exceeded $P_e$. In addition, by given some hard statistical constraint (i.e., a lower value of $P_e$), such the estimate could have the potential of bounding the actual WCET[1] of the program $p$.

## III. A Statistical WCET Analysis RapidET

This section is split into two parts: Section III-A explains our proposed bootstrapping sampling, and Section III-B introduces our proposed statistical WCET analysis based around EVT.

### A. Bootstrapping for Collecting Timing Traces Taken from Programs

Our bootstrapping sampling proposed in this work consists of three steps, which are about the collection of *non-i.i.d. SRS samples*, *i.i.d. SRS samples* and *bootstrap samples* respectively. The three steps, namely:

1) **Construction of non-i.i.d. SRS samples:** First, when any type of statistical techniques is applied to analyze the observations (or samples)[2], there is a key issue of selecting such samples from the population of all individuals concerning the desired information, i.e., any bias on the sampling has to be avoided. In this work, we therefore employ the technique of Simple Random Sample (SRS) [6], which gives every possible sample of a given size the same chance to be chosen. Practically, the SRS can be done in terms of randomizing program inputs by using the *uniform distribution*. Due to the existence of dependencies between different states of the data structure in the program, an upcoming ET data may not be independent with the ET data previously measured at program executions, when the SRS technique is used. This violates i.i.d. assumption required by any statistical methods. We refer to the samples collected at this step as *non-i.i.d. program SRS ET samples*.
2) **Construction of i.i.d. SRS samples:** In order to tackle with the problem about non-i.i.d. program SRS ET samples, we propose to first execute the target program for $n$ times by using the SRS technique which results in $n$ *sub-timing traces*, and each of sub-timing traces contains $m$ non-i.i.d. program SRS ET samples. Next, per sub-timing trace, the highest value of $m$ ET data of the program measured, will be chosen as a sample to construct the new sampling distribution of the i.i.d. SRS ET data samples of the program. Since there are no dependencies between any maximum of the ET data of the program from two independent sub-timing traces, as a result, all the individuals in the new reconstructed sampling distribution are mutually independent. Hence, the underlying i.i.d. assumption is satisfied. We call such new constructed ET data sampling distribution of the program as the *i.i.d. program SRS ET samples*, hereafter.
3) **Construction of bootstrap samples:** With the intention to have more robust analysis samples, we will use *bootstrapping*, which draws randomly with replacement from $n$ i.i.d. program SRS ET samples for N times. Consequently, there are N *program bootstrap ET samples*. It is interesting to note that resampling with replacement means that after an observation from the i.i.d. program SRS ET samples is randomly drawn, it will be placed back before drawing the next observation [7].

The detailed implementation of SRS is the function *SRS* as shown in line 2 in Algorithm 1 (i.e., pseudo-code for our proposed statistical WCET analysis which is to be introduced in Section III-B), where the parameters $P$ and $m$ represent the program under analysis and the certain number of timing traces taken from the target program

[1]When the actual WCET is possible to deduce.

[2]From such observations, some interesting conclusions based on hypothesis tests can be drawn.


Figure 1. The workflow for bootstrapping sampling we developed in this work.

respectively. It is interesting to stress that the SRS technique also gives us the confidence that no matter how big the population is, the statistical inference based on the sampling distribution collected by using SRS can successfully estimate the parameters of the underlying population [6], such as the tail behavior of the underlying population, i.e., the WCET of the programs under analysis in our case. The implementation of bootstrapping sampling as a whole can be found in lines from 1 to 6, and $RR$ is the function *Resampling with Replacement*, where $X$ is $n$ i.i.d. program SRS ET samples, and $N$ is the number of program bootstrap ET samples. The workflow pertaining to bootstrapping sampling in this work is shown by Figure 1.

### B. RapidET

Our proposed statistical WCET analysis method *RapidET* is based on EVT. Furthermore, RapidET is a recursive procedure which, as the first three arguments, takes $n$ and $m$, to generate $N$ bootstrap samples pertaining to programs' execution times. The algorithm next returns the WCET estimate of the program under analysis with a probability of being exceeded, e.g., $10^{-9}$, which is the fourth algorithm argument. For instance, Airbus [8] uses such the value $10^{-9}$ which is at the highest development assurance level in the safety-critical system domain. The last parameter $P_{GOF}$, i.e., the significance level of the hypothesis tests used in RapidET is chosen as 0.05, which is a typical value and based on preliminary assessments provides appropriate results [9].

RapidET consists of the following two steps: 1) construction of the bootstrap samples and, 2) derivation of a WCET estimate of bootstrap samples by using EVT that is given by the algorithm. Moreover, the outline of the algorithm is as follows:

1) Construct $N$ bootstrap samples for the WCET estimates by using our bootstrapping sampling described in Section III-A.
2) Calculate the WCET estimate of the bootstrap samples, i.e., $X^*$.
   a) Set the initial block size $b$ to 1.
   b) If the number of blocks $k = \left\lfloor \frac{X^*}{b} \right\rfloor$ is less than 30, the algorithm stops as there are not enough samples to generate an estimate [10].
   c) Segment $X^*$ execution times into blocks of size $b$, and for each of the $\left\lfloor \frac{X^*}{b} \right\rfloor$ blocks find the maximum values.
   d) Estimate the best-fit parameters of the Gumbel Max distribution, i.e., $\mu$ and $\beta$ to the block maximum values, by using a proposed exhaustive search algorithm together with Chi-square test. The pertaining implementation is shown in lines 7 to 21 in Algorithm 1.
   e) Calculate a WCET estimate based on Equation 1 which uses the best-fit Gumbel Max parameters estimated through Step $d$), i.e., $\mu$, $\beta$, and a target acceptance probability $P_e$.

$$est = \mu - \beta \times log(-log((1 - P_e)^b)) \quad (1)$$

**Algorithm 1** $RapidET(n, m, N, P_e, P_{GOF})$

```
1:  for all x_i such that 1 ≤ i ≤ n do
2:      et_i ← et_{i,1}, ..., et_{i,m} ← SRS(P, m)
3:      x_i ← MAX(et_i)
4:  end for
5:  X ← x_1, ..., x_n
6:  X* ← RR(X, N)
7:  b ← 1
8:  k ← ⌊X*/b⌋
9:  success ← false
10: while k ≥ 30 and success = false do
11:     S_i ← s_{i,1}, ..., s_{i,k} ← segment(N, b)
12:     Y_i ← y_{i,1}, ..., y_{i,k} ← maxima(S_i)
13:     if passChiSquareTest(Y_i, GumbelMax, P_GOF) > 0 then
14:         success ← true
15:         μ, β ← ChiSquareTest(Y_i, P_GOF)
16:         est_et ← evtgumbelmax(b, μ, β, P_e)
17:     else
18:         b ← b + 1
19:         k ← ⌊X*/b⌋
20:     end if
21: end while
22: return est_et
```

## IV. Evaluation

We present some preliminary evaluation results of the programs *bubblesort*, *quicksort* and *janne_complex* which are taken from the Mälardalen suite [11], by using our proposed method. The reasons for evaluating these programs are: 1) They are often used by many groups in WCET analysis to evaluate their tools and, 2) They are particularly appealing since the worst-case Test Vector (TV) is easy to deduce [12], which is used to find the exact WCET of the programs under analysis on our testbed. Due to space restrictions, we cannot give the detailed description of each program under investigation, and more details can be found in [11].

As shown in Column *Actual WCET* in Table I, the actual WCET of the programs under analysis[3] is expected to be in the proximity of 1 028 clock cycles (cc hereafter), 1 523 cc and 398 cc for bubblesort, quicksort and janne_complex respectively. Moreover, the results given by HWMT, HMBA and RapidET can be found in Columns *HWMT*, *HMBA* and *RapidET* separately in Table I.

[3]Such actual WCET of programs is obtained through the manual effort on deducing the worst-case TVs in [12].

Clearly, as shown in Table II, under the assumption that the actual WCET of the programs under analysis is accurate given by the manual deduction, RapidET gives the tightest upper bound on WCET estimates among all the measurement-based WCET analysis methods used in our evaluation, in terms of 1.36%, 2.27% and 6.82% more pessimistic than the exact WCET respectively. It is also interesting to stress that the WCET estimate given by HWMT which is the state-of-the-art measurement-based WCET analysis used in industry, cannot bound the actual WCET of any of the three programs under investigation. In addition, more programs in the Mälardalen suite will be evaluated as part of our future work, since our evaluation results show that the WCET estimate given by our proposed statistical WCET analysis method may have the potential of being a more accurate WCET upper bound of the program under analysis, when compared to the existing measurement-based WCET analysis.

Table I
ANALYSIS RESULTS GIVEN BY HWMT, HMBA WITH BASIC BLOCK AND RAPIDET FOR THE PROGRAMS UNDER ANALYSIS. THE UNIT IS ONE CC.

| Program | Actual WCET | HWMT | HMBA | RapidET |
|---|---|---|---|---|
| bubblesort | 1 028 | 1 008 | 1 818 | **1 042** |
| quicksort | 1 523 | 1 456 | 3 915 | **1 557.62** |
| janne_complex | 398 | 367 | 743 | **425.16** |

Table II
THE PESSIMISM OF THE RESULTS GIVEN BY HWMT, HMBA WITH BASIC BLOCK AND RAPIDET WHEN COMPARED TO THE ACTUAL WCET OF THE PROGRAMS UNDER ANALYSIS.

| Pessimism | HWMT | Actual WCET | HMBA | RapidET |
|---|---|---|---|---|
| bubblesort | -1.95% | - | 76.85% | **1.36%** |
| quicksort | -4.40% | - | 157.06% | **2.27%** |
| janne_complex | -7.79% | - | 86.68% | **6.82%** |

## V. CONCLUSIONS AND FUTURE WORK

In this paper we have presented the work towards using our proposed statistical Worst-Case Execution Time (WCET) Analysis method *RapidET*, to compute a WCET estimate of programs running on a single processor. To be specific, RapidET consists of a novel bootstrapping sampling mechanism and a statistical inference based around Extreme Value Theory [13]. Moreover, our evaluation results show that the obtained WCET estimates given by RapidET can actually bound the exact value of the WCET of programs in a less pessimistic way, when compared to the referenced measurement-based WCET analysis methods. To our best knowledge, this is the first work of WCET analysis for programs running on a single processor, which involves theoretically correct sampling mechanism and valid statistical techniques for the characterization and computation of execution time bounds on programs.

Future work will focus on a more thorough method evaluation and the selection of good value of algorithm parameters (from the perspective of accuracy of analysis results), as well as how to use the obtained WCET estimates and pertaining statistical constraints (i.e., certain predictable probabilities) in response time analysis and schedulability test which consider a system of tasks.

## REFERENCES


[1] *Handbook of Real-Time and Embedded Systems*. Chapman and Hall/CRC (July 23, 2007), 2007.

[2] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—overview of methods and survey of tools," *Trans. on Embedded Computing Sys.*, vol. 7, no. 3, pp. 1–53, 2008.

[3] Y. Lu, "Approximation Techniques for Timing Analysis of Complex Real-Time Embedded Systems," Lic. dissertation, School of Innovation, Design and Engineering, October 2010.

[4] J. Kraft, "Enabling timing analysis of complex embedded software systems," Ph.D. dissertation, Mälardalen University Press, August 2010.

[5] D. Griffin and A. Burns, "Realism in Statistical Analysis of Worst Case Execution Times," in *Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis (WCET' 10)*, 2010.

[6] D. S. Moore, G. P. Mccabe, and B. A. Craig, *Introduction to the practice of statistics*, 6th ed. New York, NY 10010: W. H. Freeman and Company, 2009.

[7] J. L. Simon, *Resampling: The New Statistics*, 2nd ed. ACM Press, 1997.

[8] "Airbus, www.airbus.com/en/, 2009."

[9] S. Stigler, "Fisher and the 5% Level," *Journal of CHANCE*, vol. 21, no. 4, p. 12, 2008.

[10] J. Hansen, S. Hissam, and G. Moreno, "Statistical-Based WCET Estimation and Validation," in *Proceedings of the 9th International Workshop on Worst-Case Execution Time Analysis (WCET' 09)*, 2009, pp. 123–133.

[11] "Website of the Mälardalen WCET benchmark," www.mrtc.mdh.se/projects/wcet/benchmarks.html.

[12] A. Betts and A. Marref, "WCET Analysis of Component-Based Systems using Timing Traces," in *In Proceedings of the 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS' 11)*, 2011.

[13] E. Gumbel, *Statistics of Extremes*. Columbia University Press, 1958.