On the Execution Time of Programs in Stochastic Scheduling

Matheus Henrique Junqueira Saldanha[1], **Adriano Kamimura Suzuki**[1]

[1]Institute of Mathematics and Computer Sciences
University of São Paulo – São Carlos, SP – Brazil

mhjsaldanha@gmail.com, suzuki@icmc.usp.br

Abstract. *Scheduling appears frequently in distributed, cloud and high-performance computing, as well as in embedded systems. Here, the execution time of subtasks is the major factor influencing decision-making, and despite being random variables they are majorly treated in the literature as being deterministic. Our project intends to shed more light on the underlying distribution of execution times, attempting to verify: 1) if the usual assumption of normal distribution is reasonable; 2) if there exist more suitable distribution families; and 3) if anything can be inferred a priori by analyzing general aspects of the program. We have modeled the problem and experimentally assessed distributions, showing that they are often not normal. We suggest alternative distributions, which we released as R packages, and propose estimators that ease parameter inference. With this we hope to promote usage of stochastic scheduling by making it easier for users to define distributions when requested.*

1. Introduction

The execution time of programs is a key element in scheduling problems, which often appear in the context of cloud computing, distributed and parallel programming, as well as in operating systems (especially real-time systems) [Tanenbaum and Bos 2015; Bittencourt et al. 2012]. As will be discussed, despite being random variables, the randomness in execution times appears to be neglected in the literature as most studies consider only the average time to perform the scheduling [Bittencourt et al. 2012; Wilhelm et al. 2008]. Some works also make use of variance information [Xavier and Annadurai 2019], and the few works that consider an underlying probability distribution either assume it as being normal or uniform, or require the user to provide such distribution by themselves [Cai et al. 2017; Shestak et al. 2008], which is a hard task. On top of that, the complexity of computer architectures has been growing continuously, incorporating deeper pipelines, larger cache hierarchies, more involved speculation and prediction techniques, more levels of parallelism, and so forth [Hennessy and Patterson 2011]. Many of such techniques contribute to increase the unpredictability of computer programs' execution times [Wilhelm et al. 2008], which poses a problem to scheduling and making programs that can meet deadlines.

So far there has not been, to the best of our knowledge, a structured study on whether these distribution assumptions are reasonable, or if using only mean and variance incur loss of valuable information. This Scientific Initiation project aims mainly to fill this lack of knowledge on execution time distributions. We attempt primarily to verify:

1. if the usual assumption of normal distribution of the execution times is reasonable;

2. if there exist distribution families that can better model execution times, and what inference methods are most effective for them; and
3. if anything can be said a priori about the underlying distribution, by analyzing general aspects of the program such as its control flow graph or whether it uses mostly CPU, memory or disk.

By answering these questions, we mainly expect to contribute to the field of cloud computing by establishing realistic assumptions for the distribution of execution times. These could then be used by future studies for experimental purposes, instead of the usual choice of normal or uniform distribution, consequently allowing more efficient scheduling algorithms to be devised. We argue later, however, that our results may also be useful for high-performance computing and real-time embedded systems. Finally, we also hope to promote usage of probabilistic scheduling algorithms by providing guidelines for users to define distributions to subroutines when requested.

We have thus far defined a model for the random variable T representing execution times, and then performed experiments to collect samples of 1000 execution times of three small programs in various machines. To assess the observed distribution, many distribution families were fit to these experimental data by means of maximum likelihood estimation (MLE), the results of which are presented in Section 3. During the process six submodels were devised and (so far) four were tested to overcome difficulties with the the MLE process (see Section 4); most of these submodels are not problem-specific and might have relevance in many other problems of Statistics. Some distributions used were not readily available in the R language, so we implemented them and released as three official packages (see Section 5). The project was initially motivated by the student's (computer science undergraduate) desire to achieve a better understanding of Statistics, but we hope to demonstrate in this paper that our results reached practical scientific value. All steps of the project (conceptualization, literature review, execution of experiments, software implementation, formal analysis etc) were carried out by the student with weekly feedback and suggestions from the advisor.

2. Background and Related Work

Originally, **cloud computing** was the main target field to which this project would contribute; it is defined as the use of outsourced computational resources on-demand in a pay-per-use basis, and marks the latest evolution of IT infrastructure, which began with centralized mainframes. It has been made feasible thanks to advances in distributed computing, networking and techniques for virtualization [Vouk 2008; Tanenbaum and Bos 2015]. With the cloud, instead of buying powerful machines that might become idle most of the time, businesses can leverage the necessary computational power from the cloud solely for the time needed. It has also enabled wonderful things in the academia. For example, the problem of protein structure prediction (PSP) is very computationally intensive, but now online servers offered by universities and associated entities (e.g., [Drozdetskiy et al. 2015; Yang et al. 2015][1]) can be used by biology and pharmacy researchers around the world – even in countries where good computers are scarce – to get predictions regarding the proteins they are researching. It is not surprising that this market is growing, with a revenue of \$150 billion dollars in the first half of 2019 [Synergy 2019].

[1]See also their websites http://www.compbio.dundee.ac.uk/jpred/ and https://zhanglab.ccmb.med.umich.edu/I-TASSER/ (Access: Mar 14th 2020).

The PSP situation above also serves as an example of problems that arise in cloud computing. Assume multiple users request a prediction at the same time; the underlying machine has multiple resources (CPU, memory, disk etc), and performing a prediction uses all of them[2], but not all at the same time. Ideally, we would like multiple predictions to be run concurrently so as to maximize usage of all resources. **Scheduling** comes to enable this. Formally, the problem is characterized by multiple directed acyclic graphs (DAGs, called *workflows* in cloud computing) representing each program as a set of tasks $T_1, \ldots, T_n$ with edges representing data dependencies. The aim is to determine the optimal way to execute them in the available processors in order to finish all programs in the lowest possible time. This is known to be an NP-complete problem in many of its forms [Ullman 1975], so it is often formulated as an optimization problem solved with heuristic algorithms; usually, the tasks are seen as having an expected execution time [Panda and Jana 2015; Zuo et al. 2015] possibly with an expected variance [Xavier and Annadurai 2019], or seen as having a computational cost in FLOPS [Shishido et al. 2018] that will later be transformed into a deterministic expected time based on machine specifications.

The area of **stochastic scheduling** is responsible for using more statistical information to perform the scheduling, and there has been some work in such subject. Li and Antonio [1997] provide a general statistical framework for stochastic scheduling, and served as a basis for subsequent works; for experimental purposes, they assumed tasks have a normal distribution. Heuristic optimization methods whose objective function involve execution time probability distributions are presented in [Dogan and Ozguner 2004]. Shestak et al. [2008] proposes the use of confidence intervals, and Cai et al. [2017] presents a method for simulating stochastic scheduling algorithms; both of these studies require the user to provide the underlying probability distributions. A method for estimating an empirical probability density function for execution times is given in [Dong et al. 2010], which involves estimating histograms and, therefore, require a decent number of available samples. Finally, Zheng and Sakellariou [2013] use Monte Carlo simulation to overcome the mathematical difficulty of stochastic scheduling, but it also expects the underlying probability distributions to be given, and for experimentation purposes they assumed normal or uniform distributions.

Although the main focus was initially in scheduling for the cloud, it is worth commenting on two related problems that can benefit from the results presented here. One is the area of **real-time systems**, in which programs whose response time is critical are studied. In the specific case of *hard* real-time systems [Tanenbaum and Bos 2015] it is crucial that deadlines be met, and failing to do so may result in casualties or other catastrophes (e.g., embedded systems within modern cars). To ensure that deadlines are met, one has to determine the worst-case execution time (WCET) of the software in question; this has been widely studied [e.g., David and Puaut 2004; Braams 2016; Ferdinand and Heckmann 2004; Wilhelm et al. 2010; Li et al. 2007] and a survey on the topic is given by Wilhelm et al. [2008]. Despite the abundance of existing work, none of them attempt to reason about the general behavior of all programs (as this project is attempting to do), and most assume distributions as being given by the user or determined through experiments for a particular software. The second related area is that of **high performance comput-**

[2]One phase of structure prediction compares the given protein to a database of known protein folds, so many prediction methods indeed make intensive use of disk.

ing in general, not only because parallel programs can often be described as a DAG of tasks that should be scheduled, but also because this area involves comparing execution times between the proposed accelerated approach and existing approaches. If execution times are seen as random, these comparisons must be followed by a hypothesis test (as done in [Kadioglu et al. 2011; Jindal and Bedi 2017; Saldanha and Souza 2019]), and the underlying distribution defines the strength of the test; the usual t-test assumes normality of the mean, and our results indicate that such approximation is reasonable.

3. Problem Modelling and Analysis

We assume the scenario of a program running on a time-sharing operating system (OS). Therefore, the program P under study may be running concurrently with other programs and daemons, and it may be interrupted occasionally by the OS. On a cloud computing machine, it is safe to assume no daemons will be present. What remains is the effect of other programs: if another program P' is running on a different CPU than P, then they affect each other by competing for I/O resources such as disk; if P and P' are on different cores of a CPU, they compete for access to memory and higher-level caches; if they are on the same core (hyperthreading), they compete for low-level caches and pipeline slots [Hennessy and Patterson 2011]. In any of these cases, the existence of other processes may cause P to be preempted periodically, which also affects its execution time. On the basis of such considerations, we model the time T of a program as being $T = T_P + N(n)$, where T_P is the time of the program on an idle machine, and $N(n)$ is the noise caused by n other programs running concurrently.

A priori determination of the underlying distributions of T_P or N seems extremely hard at first glance. We can expect the distribution of T_P to depend a lot on the hardware and on whether it is CPU-, memory- or I/O-intensive, and N to depend on the number of concurrent processes and their types too. In order to assess the behavior of T_P, we implemented 3 programs, each exercising a particular portion of the hardware: 1) **mandelbrot set calculation**, which exercises the CPU; 2) **dijkstra's algorithm**, exercising memory; and 3) and a **database program** that creates a table and then performs a number of random insertions on it, ending with an all-rows retrieval. They were then executed with at least 3 different input sizes and in 3 different machines[3], yielding 1000 execution time samples for each pair program-input-machine (a total of 37 sets). The machines include one AMD and two Intel processors, different brands of memory chips, as well as two HD disks and two SSDs[4]. Finally, different probability distribution models were fit to each of these 1000 sample sets through maximum likelihood estimation (MLE); the models used were **Weibull** (2p, i.e., 2 parameters), **gamma** (2p), **generalized gamma** (3p), **exponentiated Weibull** (3p), **normal** (2p), **truncated normal** (2p), **lognormal** (2p), **Kumaraswamy complementary Weibull geometric** (Kw-CWG, 5p) [Afify et al. 2017] and the **odd log-logistic generalized gamma** (OLL-GG, 4p) [Prataviera et al. 2017]. Optimization was done using the L-BFGS method and a grid of initial parameter values (defined for each probability model). For illustration, Figure 1 shows an example of the shape of each model after they are fit to one of the sample sets.

[3]One machine had both an HD and an SSD, so the database program was run twice on it.

[4]Details are at mjsaldanha.com/articles/2-exec-time, where all histograms generated during the experiments are also presented. Experiment code is at github.com/matheushjs/ElfProbTET under codes-R/.

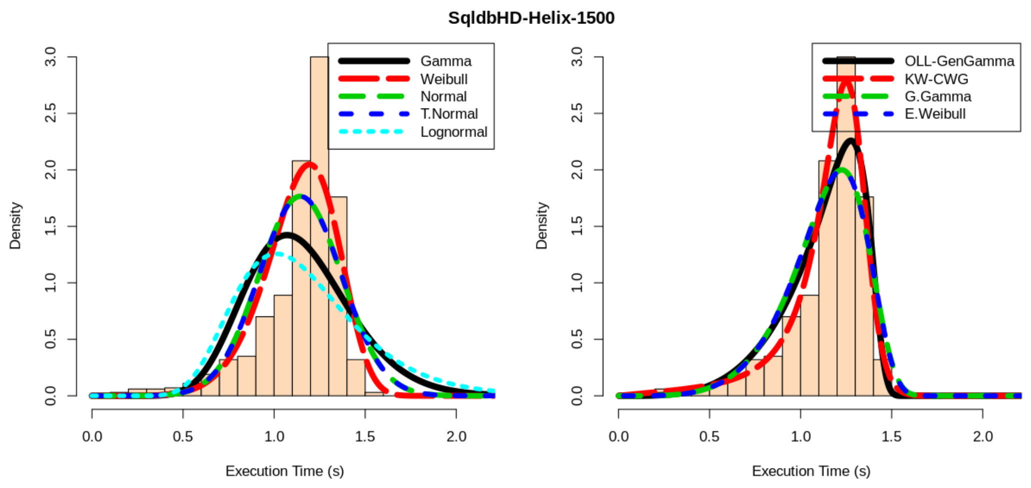
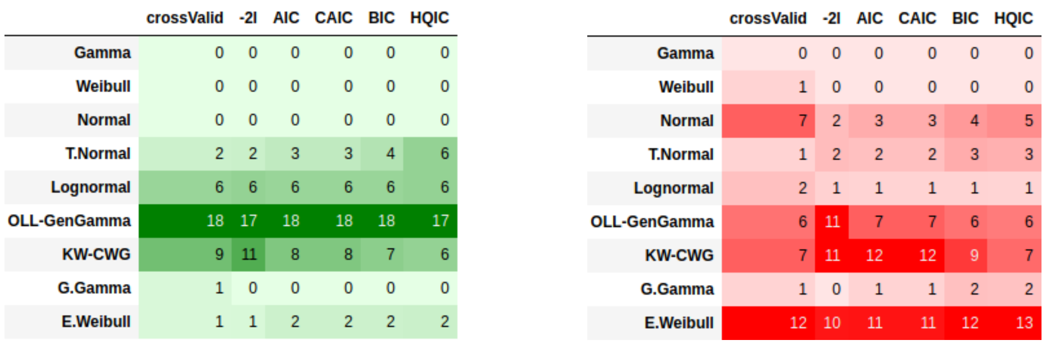


Figure 1. Example of MLE results. The histogram (equal on both sides) shows 1000 samples obtained by executing the database program (1500 insertions).

Initially, MLE performed poorly on the raw time samples, because their varying (often large) distance from zero would require very different initial values for the optimization to converge to a reasonable likelihood maximum. Our experience showed that modelling the time as $T_P = c + T'_P$, where c is the smallest time the machine might spend executing the program, made MLE much more effective. However, the population minimum c is not known, and must thus be estimated. This actually led to a whole new line of investigation, which we discuss in Section 4. In this section we present results relative to the most computationally cost-effective estimator we considered ($\hat{c} = [1-(CV/log_k n)]c_s$ as explained in Section 4). To assess goodness-of-fit, we used common metrics from the Statistics field: Akaike (AIC), consistent Akaike (CAIC), Hannan-Quinn (HQIC) and Bayesian (BIC) information criteria [Afify et al. 2017], each of which try to penalize models with too many parameters to prevent overfitting; the maximized likelihood $\hat{l}$ was also used in the usual form[5] $-2\hat{l}$. Finally, we also 5-fold cross validated the $-2\hat{l}$ metric, by fitting the models in subsamples of size 800 and then calculating the metric in the remaining 200. For all metrics, the lower its value the better the model fits the data.



	crossValid	-2l	AIC	CAIC	BIC	HQIC
Gamma	0	0	0	0	0	0
Weibull	0	0	0	0	0	0
Normal	0	0	0	0	0	0
T.Normal	2	2	3	3	4	6
Lognormal	6	6	6	6	6	6
OLL-GenGamma	18	17	18	18	18	17
KW-CWG	9	11	8	8	7	6
G.Gamma	1	0	0	0	0	0
E.Weibull	1	1	2	2	2	2

	crossValid	-2l	AIC	CAIC	BIC	HQIC
Gamma	0	0	0	0	0	0
Weibull	1	0	0	0	0	0
Normal	7	2	3	3	4	5
T.Normal	1	2	2	2	3	3
Lognormal	2	1	1	1	1	1
OLL-GenGamma	6	11	7	7	6	6
KW-CWG	7	11	12	12	9	7
G.Gamma	1	0	1	1	2	2
E.Weibull	12	10	11	11	12	13

Figure 2. Number of times each distribution model achieved the best fit (left) and second-best fit (right). Darker cells mean higher values.

For each of the 37 sets of 1000 execution time samples, we counted how many times each distribution model achieved the best and second-best fits under each metric, which is presented in Figure 2. The figure shows that in general the OLL-GG model

[5]This form is merely so that it is directly comparable to the information criteria values.

performs very well, followed by Kw-CWG and lognormal. The exponentiated Weibull rises a lot when considering the second-best scenario, making it a possible good candidate too. Let us visualize this another way: for each of the 37 sample sets, consider how far from OLL-GG each model's cross validated metric was; that is, take $M_{\text{model}} - M_{\text{OLLGG}}$ where M_{model} is the metric of the model in question[6]. For each model, this will generate 37 paired differences shown in Figure 3, with which we can compare the models among themselves, and allows to answer one of the initial research questions: **is it reasonable to assume execution times as being normal?** It does not seem so, as it had the worst median and high variability. Even though it has one extra parameter, the Kw-CWG model performed no better than OLL-GG; the extra parameter makes MLE more difficult and lengthy, so we believe Kw-CWG should not be considered for execution times. Although the exponentiated Weibull performed worse, it displayed small variability and not many outliers; since it has one less parameter than OLL-GG, we believe it should be considered for modelling execution times.

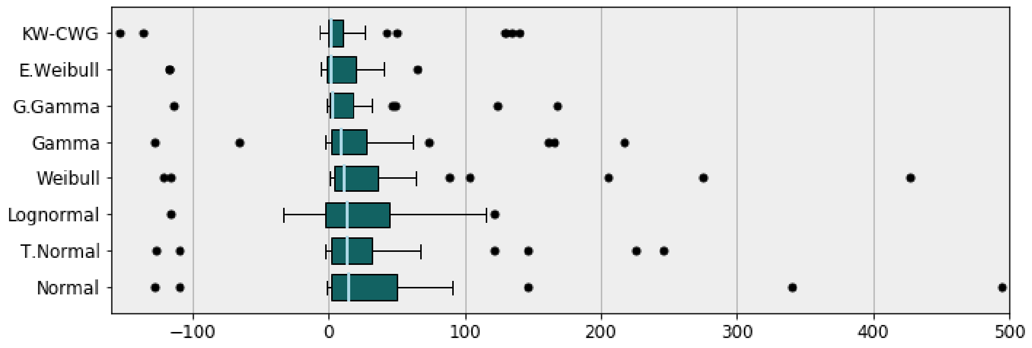


Figure 3. Paired differences of each model's cross validated $-2\hat{l}$ with the value obtained by OLL-GG. Boxes are vertically sorted by their medians.

A few notes are pertinent here. First, the conclusions above do not change a lot when considering any of the other collected metrics (e.g., AIC), nor when considering only samples of one specific machine or one specific algorithm. Second, when considering one of the information criteria, the very negative outliers of Figure 3 do not occur; this likely means that in two cases the OLL-GG model overfitted the data during the cross validation, so it achieved a very poor performance on the unseen fold. Finally, gamma was the best performing 2-parameter distribution family.

4. Devising Estimators for the Population Minimum

The biggest problem with modelling execution times as $T_P = c + T'_P$ is deciding the value of c, which ideally would be the population minimum. Since the population is not known, what value should be used as c? Our first attempt was to take the estimator $\hat{c} = \min\{x_1, \ldots, x_n\}$, where x_i form the sample. However, after subtracting $\hat{c}$ one of the samples becomes 0, which is a problem because some models (such as gamma and Weibull) assign $P(X = 0) = 0$ to that single sample for a large range of their parameter spaces. This makes the likelihood become constant at 0, thus making optimization impossible. Arbitrarily taking $r \cdot c_s$ with $r < 1$ failed to be useful simultaneously to all sets of execution times, so we had to devise better estimators. The nearest field to this problem is **quantile estimation** [Valk and Cai 2018; Takeuchi et al. 2006], but these estimators

[6]Taking the difference of log-likelihoods finds theoretical ground on Wilks' theorem.

have limited usage for increasing likelihood values and minimizing computational cost; the estimators presented below are intended to pursue these objectives.

It seems feasible to consider: 1) the populational minimum c is lower than the sample's c_s; 2) we can expect the difference $c_s - c$ to be higher as the sample variance increases; and 3) we can expect it to be lower as the sample size n increases. This led to the estimator $\hat{c} = [1 - (CV/\log_k n)]c_s$ where CV is the coefficient of variation, a number commonly in $[0, 1]$ that measures variability in the samples, and k is an arbitrary basis for the logarithm ($k = 10$ served well for our purposes). So we are multiplying the sample minimum by a factor that will move it towards 0. The term $(1 - CV)$ goes to 0 as the variability increases; for example, if the variability is 78% our estimator would be $0.22c_s$ (ignoring the $\log$ for now). The logarithm serves to take the sample size into consideration. Consider $k = 10$ and $n = 20$, then the estimator in the same example would be $0.40c_s$, and if $n = 200$ it would be $0.66c_s$. Although this estimator is consistent (tends to the populational minimum as $n \to \infty$), we could not find a statistically-motivated way to choose k. An alternative to this estimator is to add to each distribution an extra parameter p representing the minimum, by defining $g(x|\theta, p) = f(x - p|\theta)$, for $x > c$ and 0 otherwise[7]. We effectively end up with a novel distribution with an extra parameter that can be estimated by maximum likelihood estimation (MLE). In 21 out of the 37 sample sets, this solution led to better values for the likelihood, but made the estimation process more difficult and lengthy due to the addition of one extra parameter. To illustrate, MLE for the OLL-GG model takes an average of 43.5 seconds, and 33.0 without the extra parameter.

So far we have only performed experiments with the methods above; the estimators described in the following will still be experimented with. The next estimator leverages what is known about the empirical cumulative distribution function (cdf) $F_n(x)$. The Glivenko-Cantelli theorem proves its convergence to the real cdf $F(x)$, and its rate of convergence is given by (see [Massart 1990] for a proof):

$$P(\sup_x |F_n(x) - F(x)| > \epsilon) \leq 2\exp(-2n\epsilon^2), \tag{1}$$

which gives the probability of error on approximating $F(x)$ by $F_n(x)$. Suppose we tolerate ν chance of error, then we have:

$$2\exp(-2n\epsilon^2) = \nu \implies \epsilon = \sqrt{\frac{-\ln(\nu)}{2n}},$$

and thus the following holds with probability of at least $1 - \nu$:

$$\sup_x |F_n(x) - F(x)| \leq \sqrt{\frac{-\ln(\nu/2)}{2n}} = \beta(\nu, n),$$

and in particular, the probability of sampling an execution time lower than the sample minimum c_s is $P(X < c_s) = F(c_s - \epsilon) \leq \beta(\nu, n)$ since $F_n(c_s - \epsilon) = 0$ (with any $\epsilon > 0$). $\beta(\nu, n)$ therefore measures (it is an upper bound) how likely it is to sample a new execution time that is lower than all the previous ones; if it is a low value, we expect c_s to be very near c. Consequently, we can substitute the previous arbitrary logarithm and use the estimator $\hat{c} = (1 - \beta(\nu, n))(1 - CV)c_s$ or just $\hat{c} = (1 - \beta(\nu, n))c_s$.

[7] g can be proved to be a density function by integrating it over the real line, but we omit the proof here.

The last alternative comes from order statistics. If $X_1, \ldots, X_n$ is an iid sample from a known cdf $F(x)$, then the minimum follows [DeGroot and Schervish 2012]:

$$P(\min\{X_1, \ldots, X_n\} < x) = F_m(x) = 1 - [1 - F(x)]^n \tag{2}$$

where the main problem is that the real $F(x)$ underlying execution times is unknown. However, we can use Eq. (2) conditioned on the underlying distribution having a given form, which is precisely what we do when using maximum likelihood estimation. If we do this, $F(x|\rho)$ is known given parameters ρ, and consequently so is $F_m(x|\rho)$. Inverting $F_m(x|\rho)$ in Eq. (2) yields the quantile function $F_m^{-1}(q|\rho) = F^{-1}(1 - (1-q)^{1/n}|\rho)$, and $F_m^{-1}(0.5|\rho)$ gives the median of the sample minimum, which is a safe assumption for the actual minimum obtained through sampling; following this reasoning, the sample should be subtracted so that the obtained sample minimum c_s coincides with $c + F_m^{-1}(0.5|\rho)$. We thus have our estimator:

$$\hat{c} = c_s - F_m^{-1}(0.5|\rho),$$

which depends on the parameters ρ, so this should be incorporated in the MLE optimization process itself; recall that our model is $T = c + T'$, so at every iteration the new $\hat{c}$ must be calculated and subtracted from the samples (so what remains is T') and then the likelihood is calculated again. The estimator for the first iteration can be chosen with any of the other estimators presented.

5. Conclusions and Future Work

So far in this project, we have investigated the suitability of probability models for the execution time of programs. The problem was analyzed, and then influencing factors and possible problems were identified, and finally a model $T = T_P + N(n)$ was proposed (see Section 3). Thus far only the T_P component has been studied: experiments with sample programs were performed in multiple machines, and the resulting execution times were extensively studied. Some of our initial research questions have been answered to some extent: 1) according to the experiments, assuming execution times as being normal does not seem reasonable; and 2) experiments indicate that the OLL-GG and the exponentiated Weibull (and possibly gamma) distributions work well for modelling execution times, and we are currently studying how to best cope with the problem of estimating the populational minimum (for the general problem, not just to our case), with the estimators described in Section 4. We recognize that the answer to the first question is limited in that we do not yet answer "how much" loss is incurred by assuming a normal distribution instead of one of the proposed ones; we hope to soon simulate scheduling algorithms using each of these distributions, and assess the practical differences in performance. During the project we also published three packages in the official R public repository (CRAN): ggamma, ollggamma and elfDistr, each of which contains density, cumulative density and quantile functions, as well as a random number generator for distributions we needed but were not available or were not efficiently implemented.

The third research question remains to be answered. We noticed that the distribution of execution times might be tightly related to its underlying control flow graph (CFG) as long as the nodes in the graph are sufficiently coarse so that their execution times are independent from each other. Our hypothesis is that if the time variance of each node is limited, and if the nodes are executed a fair number of times (e.g., > 30), then the overall time can be modelled as a normal distribution. This is because the loop would represent a long sum of random variables, which by the Central Limit Theorem would converge to a normal. Note that the programs we implemented were small (not enough CFG nodes were executed), so their distributions were not always normal. We intend to investigate how we can use these ideas to draw a priori conclusions about whether a program will result in normal execution times or not.

This project has shown itself to be an outstanding experience for the student, who learned more about Statistics and how it is applied in practice. Despite initially having this purpose, we hope to have convinced that scientifically significant results were achieved, about which we plan to write an article and have it peer-reviewed in the near future.

Acknowledgement. We thank prof. Ricardo Marcacini for valuable suggestions, as well as the LaSDPC and BioCom laboratories for the computational and other resources.

References

Afify, A. Z., Cordeiro, G. M., Butt, N. S., et al. (2017). A new lifetime model with variable shapes for the hazard rate. *Braz J Probab Stat*, 31(3):516–541.

Bittencourt, L. F., Madeira, E. R., and Da Fonseca, N. L. (2012). Scheduling in hybrid clouds. *IEEE Communications Magazine*, 50(9):42–47.

Braams, B. (2016). Deriving an execution time distribution by exhaustive evaluation. *Bachelor's thesis, University of Amsterdam.*

Cai, Z., Li, Q., and Li, X. (2017). Elasticsim: A toolkit for simulating workflows with cloud resource runtime auto-scaling. *J Grid Comput*, 15(2):257–272.

David, L. and Puaut, I. (2004). Static determination of probabilistic execution times. In *16th ECRTS*, pages 223–230. IEEE.

DeGroot, M. H. and Schervish, M. J. (2012). *Probability and statistics.* Pearson.

Dogan, A. and Ozguner, F. (2004). Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous systems. *Cluster Computing*, 7(2):177–190.

Dong, F., Luo, J., Song, A., and Jin, J. (2010). Resource load based stochastic dags scheduling mechanism for grid environment. In *12th HPCC*, pages 197–204. IEEE.

Drozdetskiy, A., Cole, C., Procter, J., and Barton, G. J. (2015). Jpred4: a protein secondary structure prediction server. *Nucleic acids research*, 43(W1):W389–W394.

Ferdinand, C. and Heckmann, R. (2004). ait: Worst-case execution time prediction by static program analysis. In *Building the Information Society*, pages 377–383. Springer.

Hennessy, J. L. and Patterson, D. A. (2011). *Computer architecture: a quantitative approach.* Elsevier.

Jindal, V. and Bedi, P. (2017). Reducing waiting time with parallel preemptive algorithm in vanets. *Vehicular Communications*, 7:58–65.

Kadioglu, S., Malitsky, Y., Sabharwal, A., et al. (2011). Algorithm selection and scheduling. In *17th CP*, pages 454–469. Springer.

Li, X., Liang, Y., Mitra, T., and Roychoudhury, A. (2007). Chronos: A timing analyzer for embedded software. *Science of Computer Programming*, 69(1-3):56–67.

Li, Y. A. and Antonio, J. K. (1997). Estimating the execution time distribution for a task graph in a heterogeneous computing system. In *HCW'97*, pages 172–184. IEEE.

Massart, P. (1990). The tight constant in the dvoretzky-kiefer-wolfowitz inequality. *The annals of Probability*, pages 1269–1283.

Panda, S. K. and Jana, P. K. (2015). Efficient task scheduling algorithms for heterogeneous multi-cloud environment. *J supercomput*, 71(4):1505–1533.

Prataviera, F., Cordeiro, G., Suzuki, A., et al. (2017). The odd log-logistic generalized gamma model. *Biom Biostat Int J*, 6(4):174.

Saldanha, M. and Souza, P. (2019). High performance algorithms for counting collisions and pairwise interactions. In *19th ICCS*, pages 182–196. Springer.

Shestak, V., Smith, J., Maciejewski, A. A., and Siegel, H. J. (2008). Stochastic robustness metric and its use for static resource allocations. *J Parallel Dist Com*, 68(8):1157–1173.

Shishido, H., Estrella, J., Toledo, C., et al. (2018). Genetic algorithms applied to workflow scheduling with security and deadline constraints in clouds. *Comput Electr Eng*, 69.

Synergy, R. G. (2019). Half-yearly review shows $150 billion spent on cloud services and infrastructure. https://www.srgresearch.com/articles/half-yearly-review-shows-150-billion-spent-cloud-services-and-infrastructure. Last access: 31-Mar-2020.

Takeuchi, I., Le, Q. V., Sears, T. D., and Smola, A. J. (2006). Nonparametric quantile estimation. *Journal of machine learning research*, 7:1231–1264.

Tanenbaum, A. S. and Bos, H. (2015). *Modern operating systems*. Pearson.

Ullman, J. (1975). Np-complete scheduling problems. *J Comput Syst Sci*, 10(3).

Valk, C. and Cai, J. (2018). A high quantile estimator based on the log-generalized weibull tail limit. *Econometrics and statistics*, 6:107–128.

Vouk, M. A. (2008). Cloud computing: issues, research and implementations. *J comput inf tech*, 16(4):235–246.

Wilhelm, R., Altmeyer, S., Burguière, C., et al. (2010). Static timing analysis for hard real-time systems. In *Proceedings of VMCAI Workshops*, pages 3–22. Springer.

Wilhelm, R., Engblom, J., Ermedahl, A., et al. (2008). The worst-case execution-time problem – overview of methods and survey of tools. *ACM TECS*, 7(3).

Xavier, V. A. and Annadurai, S. (2019). Chaotic social spider algorithm for load balance aware task scheduling in cloud computing. *Cluster Computing*, 22(1):287–297.

Yang, J., Yan, R., Roy, A., Xu, D., Poisson, J., and Zhang, Y. (2015). The i-tasser suite: protein structure and function prediction. *Nature methods*, 12(1):7.

Zheng, W. and Sakellariou, R. (2013). Stochastic dag scheduling using a monte carlo approach. *J parallel distr com*, 73(12):1673–1689.

Zuo, L., Shu, L., Dong, S., et al. (2015). A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing. *Ieee Access*, 3.