Why You Should Care About Quantile Regression

Augusto Born de Oliveira
Sebastian Fischmeister
Dept. of Electrical and Computer Eng.
University of Waterloo
Waterloo, ON, Canada
{a3olivei,sfischme}@uwaterloo.ca

Amer Diwan
Google Inc.
Mountain View, CA, USA
diwan@google.com

Matthias Hauswirth
Faculty of Informatics
University of Lugano
Lugano, Switzerland
Matthias.Hauswirth@usi.ch

Peter F. Sweeney
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA
pfs@us.ibm.com

Abstract

Research has shown that correctly conducting and analysing computer performance experiments is difficult. This paper investigates what is necessary to conduct successful computer performance evaluation by attempting to repeat a prior experiment: the comparison between two Linux schedulers.

In our efforts, we found that exploring an experimental space through a series of incremental experiments can be inconclusive, and there may be no indication of how much experimentation will be enough. Analysis of variance (ANOVA), a traditional analysis method, is able to partly solve the problems with the previous approach, but we demonstrate that ANOVA can be insufficient for proper analysis due to the requirements it imposes on the data.

Finally, we demonstrate the successful application of quantile regression, a recent development in statistics, to computer performance experiments. Quantile regression can provide more insight into the experiment than ANOVA, with the additional benefit of being applicable to data from any distribution. This property makes it especially useful in our field, since non-normally distributed data is common in computer experiments.

Categories and Subject Descriptors D.4.8 [*OPERATING SYSTEMS*]: Performance

Keywords Scheduling, Latency, Performance Evaluation, ANOVA, Quantile Regression

1. Introduction

Both academia and industry use performance evaluation to identify bottlenecks and to evaluate innovations. Unfortunately, performance evaluation is difficult and error prone because it requires (1) profound knowledge of hidden factors that on the surface seem irrelevant to the experiment [11, 18], and (2) deep understanding of statistical methods used to analyze the data from the experiment [22]. This paper sheds light on both of these difficulties by attempting to reproduce a prior experiment: the comparison between two Linux schedulers.

The two Linux schedulers we considered are the current mainline scheduler, the Completely Fair Scheduler (CFS) [16], and the current contender, the BFS [14]. The CFS concentrates on maximizing throughput on a wide range of platforms, while the BFS aims to maximize interactivity by reducing latency. A current hot debate is which of the two schedulers is superior to the other and under which conditions. To help resolve the debate, the Linux community came up with experiments to measure scheduling latency and computing throughput to identify the best scheduler for interactive platforms.

Our results show that while we were able to repeat prior experiments for these schedulers, we were unable to reproduce the community's conclusions based on the results. By this we mean that when we attempted to generalize the prior experiment, we encountered a series of contradictions. For example, in one experiment, CFS had superior latency and throughput to BFS while a minor change suddenly reversed the decision for both. After conducting a series of experiments, we were not only unable to reach a decisive conclusion, but also unable to determine when enough experimentation had been done.

Trying to produce a reproducible experiment, we applied textbook experimental design and analysis techniques, namely factorial design and analysis of variance [17] (ANOVA). Through a factorial design, researchers can clearly define an experimental space, and through ANOVA they can draw conclusions that apply in that space. We found that while this was effective for some metrics, it was inapplicable for others because they failed to satisfy the necessary assumptions for applying the technique.

Next, we applied a new statistical technique to our experiment: quantile regression [12]. Quantile regression allows researchers to analyse data in greater detail than ANOVA, without requiring the data to satisfy specific statistical properties. While ANOVA focuses on the mean, quantile regressions allow the quantification of factor effects on any given quantile (e.g., the median, or the 99^{th} percentile) of the metric of interest. Our experiments with quantile regression led to correct and reproducible results.

This paper is organized as follows: Section 2 presents the scheduler comparison used as a base for the rest of the paper. Section

ASPLOS'13, March 16–20, 2013, Houston, Texas, USA.

3 describes the benchmark and the platforms used in experimentation. Section 4 demonstrates the pitfalls of unstructured experimentation. Section 5 presents our factorial design, and demonstrates both the strengths and weaknesses of ANOVA. Section 6 demonstrates the successful use of quantile regression. Section 7 describes a series of other pitfalls encountered in our experimentation. Section 8 discusses related work, and Section 9 concludes.

2. Background

In this paper, we compare two Linux schedulers — the Completely Fair Scheduler (CFS), and the BFS — to investigate the difficulty in measuring and analysing computer performance. The Linux scheduler is important, since it potentially affects the performance of platforms ranging from cellphones to large data centers. The CFS [16], Linux's official scheduler since 2007, aims to be a unified solution for all target platforms, scaling to a large processor count. The BFS [14] was created in 2009 to minimize scheduling latency, trying to yield better performance in interactive systems.

The release of the BFS led several developers on the Linux Kernel Mailing List to conduct independent experiments to measure the performance of both schedulers [6]. BFS proponents argue that its focus on lowering scheduling latency yields better interactivity and responsiveness on commodity hardware; a claim that experimenters could verify through scheduling latency benchmarks. Furthermore, CPU and mixed-load benchmarks can measure scheduler overhead, predicting applications' throughput under the BFS.

Initial attempts to verify the effectiveness of the BFS were inconclusive, and after some discussion, a kernel developer released the Latt [1] benchmark. Latt measures scheduling latency while generating CPU load; this ensures the scheduler treats the benchmark process as it would a typical interactive application. Section 3.1 describes Latt. Even with the acceptance of Latt by both sides of the discussion, all experimentation conducted was unstructured and unscientific; unsurprisingly, the kernel developers did not come to a consensus, and the BFS and the CFS are both under concurrent development. We use Latt for all experiments in this paper, since it adequately measures scheduling latency, and is already accepted and used by kernel developers, precluding the need for revalidating it. Using Latt is also important because of our goal of reproducing the unstructured experiments that the kernel developers conducted.

3. Experimental Setup

Through a series of experiments, we attempt to determine how the BFS and the CFS compare in terms of latency and throughput.

3.1 Latt

Latt is composed of a server thread and a set of one or more client threads. Figure 1 shows the control flow of Latt. At the start of execution, the server thread spawns the client threads, and opens a pipe (a stream for interprocess communication found in UNIX-like systems) to each of them. The client threads then block trying to read from the pipe. At random intervals, the server thread then writes a *wakeup request timestamp* to each of the pipes, freeing the client threads. As soon as the clients finish reading that timestamp, they collect a *woken up timestamp*. The difference between the woken up and the wakeup request timestamps is the time during which the client thread was ready to execute but did not receive the processor, i.e., the scheduling latency.

As soon as each client collects the woken up timestamp, they perform a compression task. This is to avoid scheduler heuristics that may prioritize client threads for consistently releasing the processor before using their full share of CPU time. Latt registers the time to complete these compression operations, and depending

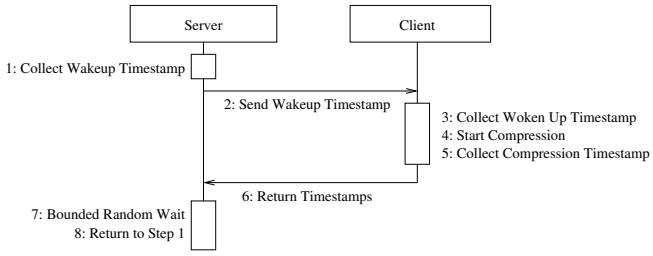


Figure 1. Control flow of the Latt benchmark.

on the load of the system, these measurements serve as a surrogate for throughput[1]. The user can configure the number of clients and the size of the data to compress. There must be at least one client (so that latency measurements are collected), but the compression size may be zero, meaning that the client threads perform no work, and Latt only collects latency information.

Latt thus collects two metrics, scheduling latency (in microseconds) and client throughput (in bytes compressed per microsecond). The lower the latency measurement, the more responsive an interactive system will be. The higher the client throughput, the faster a user would expect a CPU-bound task to complete.

Measuring latency and client throughput in isolation is unrealistic; to solve that, Latt permits defining arbitrary background loads, by taking an arbitrary program as a parameter. Latt collects measurements for as long as that background load executes. For example, if `sleep 30` is passed as a parameter, Latt will collect latency and client throughput data for 30 seconds, resulting in multiple measurements per execution. The background load parameter guarantees that data collection happens only while the load executes (the loads used in our experiments are described in the next section).

By default, Latt outputs only aggregate statistics (such as the mean and standard deviation) of its latency and client throughput measurements. To enable in-depth data analysis, we modified Latt to output each individual latency and client throughput measurement. Our implementation of this feature allocates a large buffer in the beginning of execution and only writes the data to disk once the background load completes, thus minimizing interference on the metrics of interest. Furthermore, in addition to latency and client throughput data, we collected end-to-end execution time measurements for each execution, to analyze the effect of each scheduler on the background load execution time.

3.2 Background Load Configurations

We used three different load configurations: *idle*, *compile*, and *encode*. To differentiate the schedulers under a wide range of conditions, we chose loads that represented the spectrum from an idle system to one executing a multi-threaded, CPU-bound load. As described earlier, Latt receives each of these loads as a command line parameter.

The idle load blocks on a timer for a set amount of time, using no CPU time. This load serves to establish a baseline for the metrics on an idle system. The command passed to Latt for this load is `sleep 30`.

The compile load consists of compiling the *xz v5.0.3* [20] open-source compression utility using *GCC 4.6.3*. It is a single threaded load, consisting of a mix of CPU and I/O operations. The command passed to Latt is `./configure; make`.

The encode load is based on *x264 v0.120.x* [21], an open-source H.264 video encoder. It is a multi-threaded, CPU-bound load. The

[1] This approach is problematic, and Section 7 discusses its shortcomings.

command passed to Latt is `x264 -B 2000 soccer_4cif.y4m -threads 4`. The input file to the encoder is a freely-distributable test sequence [23].

3.3 Experimental Platforms

We used two different computers for the experiments. A Core 2 machine, consisting of a 2.4GHz Intel Core 2 Quad Q6600 with 3GB of RAM, and a P4 machine, consisting of a 2.0GHz Intel Pentium 4 with 512MB of RAM. The P4 machine was chosen to measure latency behavior on a slower processor, on which the scheduler choice may prove more significant. Both machines executed version 3.2.7 of the Linux kernel, with BFS version 416. The distribution used was Arch Linux, and only essential software executed during data collection (most notably, no graphical interface was used during benchmarks). Since these machines required remote access, their network connection was active throughout benchmarking. The relatively long running time of the benchmarks and the fact that the machine rejected connections except on the remote terminal port minimize any network interference on the metrics of interest.

4. Repeating Prior Experiments

Our first experiments started in an attempt to replicate experiments described in the Linux Kernel Mailing list. The goal of all experiments in this section is to determine how the schedulers compare in terms of latency and end-to-end performance. The initial intuition (according to the developers involved in the discussion) is that the BFS yields better latency, but the CFS yields better end-to-end times. We will now test this intuition with a sequence of experiments, using information gained along the way to determine the next step.

All results presented in this section were collected on the Core 2 machine. For each experiment, the machine boots into the kernel with the appropriate scheduler, executes the benchmark, and offloads the data to a network server. Figures 2(a) and 2(b) show box plots of the latency and end-to-end times under both schedulers for all experiments described in this section. The data set for each experiment consists of ten separate executions of Latt, each of which yields several latency samples and one end-to-end sample. The three horizontal lines in each box represent the upper quartile, the median, and the lower quartile. The lower and upper whiskers extend to the last value within 1.5 times the interquartile range of the lower and upper quartile, respectively. Dots represent outliers that are outside this range. The cross ($\times$) represents the mean value of each distribution. Note that the y-axis on Figure 2(a) is in a logarithmic scale.

4.1 Experiment 1: Idle

The first experiment was conducted on an idle system. Latt executed a single client thread, which performed no compression, and we used the idle load. The goal of this experiment was to ascertain if there was a significant performance difference between the schedulers in a system with no load. This experiment also serves to establish that the method used by Latt provides reliable data, that is, the experiment is repeatable.

The plot for Experiment 1 in Figure 2(a) confirms the intuition that the average latency under the BFS is lower than that of the CFS. Nevertheless, even the worst case under CFS is below 30μs (much lower than the human-perceptible 100ms [3]). The standard deviation under both schedulers is very low (under 3μs for both schedulers), which suggests Latt's method for measuring latency is robust, at least in an idle system.

This experiment confirms the expectations of the developer of the BFS, but the latency behavior measured with this unrealistic load may not be observed in real applications. Therefore, investigating how the schedulers behave under load is still necessary.

4.2 Experiment 2: Compile

The goal of this experiment is to evaluate the different schedulers under more realistic conditions than in Experiment 1. This experiment has all the same conditions of Experiment 1, except for the use of the compile load instead of the idle load.

The plot for Experiment 2 in Figure 2(a) shows that, under these conditions, the BFS has a higher mean latency and higher variance than the CFS. These results contradict the previous experiment *and* the initial claim. However, this could be a scheduling artifact: the lack of work in the Latt client threads may be causing the CFS to prioritize them over all others, since it queues threads by spent processor time. This, in turn, leads to artificially low latency measurements. Adding load to the Latt clients could confirm this hypothesis.

4.3 Experiment 3: Compile with Compression

The goal of this experiment is to check whether increasing load in the Latt client threads will affect latency measurements. To accomplish this, we changed two parameters from the last experiment: we increased the number of clients in Latt from 1 to 8, and each client compressed 64kb of data. In a real application, latency-sensitive threads also perform work, which is approximated by the compression functions in Latt.

The plot for Experiment 3 in Figure 2(a) shows that the increase in the number of client threads and the addition of compression to their work cycles causes a reversal in the results from the previous experiment, and now confirm the initial claim once again. In contrast to the previous experiment, now the BFS has better average latency and better worst-case latency than the CFS. The CFS' highest latency measurement approaches 60ms, which is significantly closer to the human perceptible threshold than the BFS' worst observed case. If latencies of 60ms are observed in these relatively powerful desktop systems, the perceptible threshold of 100ms will likely be surpassed on slower processors, such as those found on mobile systems.

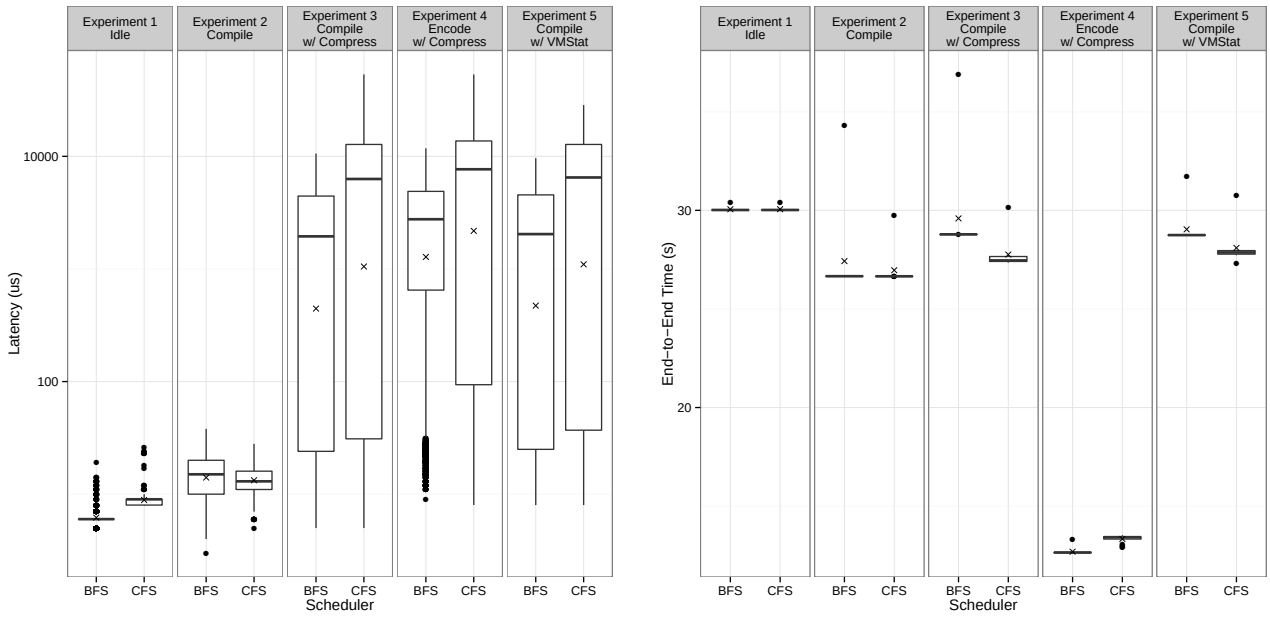
While the conditions of this experiment represent a more realistic environment than the previous experiments, the CPU may not have been fully utilized (due to I/O operations in the background load, and the fact that Latt contains sleep cycles). Therefore, the behavior of the two schedulers under full CPU load requires further investigation. Since interactive systems intermittently go through temporary periods of full utilization, making a conclusion at this point could be premature.

4.4 Experiment 4: Encode

The goal of this experiment is to compare the schedulers under full CPU load. To accomplish this, we modified the parameters of last experiment by using the encode background load instead of the compile load. Preliminary experiments showed this multi-threaded background load to fully utilize the CPU on the Core 2 machine.

Figure 2(b) shows end-to-end execution times for all experiments. The BFS has a lower execution time than the CFS for this experiment, contradicting yet again the initial claim and the previous two experiments. Figure 2(a) shows latency was also lower for the BFS than for the CFS.

With these additional results, there is still insufficient evidence to choose a scheduler based on either metric. To attempt to explain the reversal in end-to-end time performance, we decided to collect idleness data; if the BFS is utilizing the CPU at the same rate as the CFS, then overhead associated with scheduling may be causing the observed discrepancy in end-to-end time. Since the encode load



(a) Latency

(b) End-to-End Times

Figure 2. Results of the sequential experiments.

requires fewer scheduling decisions, this overhead would not be as pronounced.

4.5 Experiment 5: Compile with Idleness Data

The goal of this experiment is to revisit Experiment 3, with the addition of the idleness data collection. We added calls to the *VMStat* Linux utility to Latt, so that every time Latt collected a latency sample, it used VMStat to collect an idleness value. We instrumented Latt to buffer idleness data in memory, along with latency and client throughput data.

The last plot on Figure 2(a) shows that, while the overall conclusion is the same from Experiment 3 — the BFS has better average and worst-case latency — the worst-case latency for the CFS decreased by nearly 50%, from 53,370μs to 28,530μs. The latency behavior of the BFS, on the other hand, remained the same. This change in behavior was later revealed to be caused by a probe effect specific to the CFS, caused by a scheduling policy introduced in late 2010 (the `CONFIG_SCHED_AUTOGROUP` kernel option). Section 7 includes details on this effect.

4.6 Discussion

The unstructured experiments demonstrated that several factors affect both the latency and the end-to-end time behavior of both schedulers: the background load, the Latt client load, and the presence of VMStat calls in Latt. Unfortunately, they also led to a series of contradictions, and no clear conclusion as to what scheduler has a better latency or end-to-end performance. Worst of all, there is no indication of when these sequential experiments would converge to a conclusion.

There was, however, a significant difference between schedulers on all experiments, and the changes in experimental conditions affected both metrics on both schedulers. We used Kruskal-Wallis tests [17][2] to analyze the data in two ways: first, we verified that the latency and end-to-end time distributions differed between schedulers for all experiments (P ¡ 0.01), second, we verified that latency and end-to-end time distributions for each scheduler differed between experiments (P ¡ 0.01). Since the schedulers differ, there must be a correct scheduler choice under each operating condition.

An important question to answer is "when have you experimented enough to draw conclusions?" To answer this question, we realized we needed to more systematically explore the factors affecting our metrics. Using traditional design and analysis of experiments, we can delimit our experimental space and, if analysis is successful, draw conclusions about the performance of each scheduler.

5. Factorial Design

We now use established statistical methods to try to successfully compare the performance of the two schedulers. Factorial experiment design, coupled with ANOVA [9, 15, 17] allows isolating the effect of factors on response variables — latency, client throughput, and end-to-end times — measuring not only the magnitude of their individual effects, but also any interaction between factors. In this section we will demonstrate how ANOVA can be insightful when used correctly, but can also produce misleading results when used incorrectly.

Analysis of Variance (ANOVA)

A statistical analysis method that partitions the variability in a data set into its component parts.

Output: A model of the mean of the response variable as a linear function of factor levels.

Distribution Assumptions: Normally and independently distributed errors with constant variance.

[2] The Kruskal-Wallis test is a non-parametric test used to verify if two or more samples belong to the same distribution.

We used a two-level full factorial design [17] for our experiments. In this design, the experimenter varies the factors between two levels — low and high — and executes experiments containing all combinations of levels for all factors. This experiment design assumes that the factors are fixed, and allows quantifying not only the effect of factors in isolation but also their interactions. Table 1 shows the factors and levels explored in this design. All of the factors listed here measurably affected the latency in the experiments described in Section 4, and the machine factor was added to ensure results were not exclusive to the Core 2 machine used in the previous experiments. We fixed the Latt client number at eight and the compression size at 64kb, since those were the only conditions under which realistic latency measurements were collected. We omitted the sleep load for the same reason. Measuring latency in an idle system would lead to unrealistic results as demonstrated in Section 4.1.

Factor	Low Lvl.	High Lvl.
Scheduler	BFS	CFS
Machine	Core 2	P4
VMStat	off	on
Load	compile	encode

Table 1. Factors and their levels.

Exp.	Scheduler	Machine	VMStat	Load
3	CFS	Core 2	off	compile
	BFS	Core 2	off	compile
4	CFS	Core 2	off	encode
	BFS	Core 2	off	encode
5	CFS	Core 2	on	compile
	BFS	Core 2	on	compile

Table 2. Factors and their levels for the sequential experiments.

The set of factors and levels specified above define our *experimental space*. For this full factorial design, experiments will consist of executing every combination of factor levels (a total of 16 combinations) five separate, non-consecutive times. This number of runs provided us with enough data for a robust analysis, since all effects were measured with a high significance level; if this were not the case, we would collect more data to improve the statistical power of the test (i.e., reduce the odds of a false negative). To ensure that the measurement error is distributed uniformly across all factor levels, we randomized the order of the execution of the trials.

Table 2 shows how the last three experiments from the previous section consist of a six trial subset of the full-factorial design. By simultaneously varying all factors in a series of automated experiments, the full-factorial design will allow us to reach general conclusions without having to sequentially explore the experimental space one factor at a time.

The analysis of the results consists of creating a linear model of the data, relating the factors to one response variable at a time using linear regression. Linear regression is a method to calculate the best fitting linear mathematical model to a data set, and assumes normally distributed, independent observations. The resulting model will be of the form

$$\begin{aligned} y \quad = \quad & \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \\ & \ldots + \beta_{12} x_1 x_2 + \ldots + \beta_{ik} x_i x_k + \\ & \ldots + \epsilon \end{aligned} \tag{1}$$

where y is the response variable, which is related to k factors, each with a level x_i and a coefficient β_i, where $i = 1, \ldots, k$. Terms of the form $\beta_{12} x_1 x_2$ allow the modeling of interactions between factors. Higher-level interactions (between three or more factors) can also be included in the model. The ϵ term is a random error component that incorporates other sources of variability (uncontrolled factors and measurement error). This model describes a hyperplane in the k-dimensional space of the k factors, and β_0 is the intercept for the plane, i.e., where the plane crosses the response variable dimension when all $x_i = 0$.

More intuitively, let x_1 represent the scheduler factor. We attribute indicator values to its levels: -1 for low (BFS) and +1 for high (CFS). The scheduler effect β_1 will then be *half* the expected change in the mean response variable when the scheduler factor is modified and all other factors are unchanged. In other words, the low indicator value $x_1 = -1$, will subtract the value of β_1 from the response variable, the high indicator value $x_1 = +1$, will add it; hence β_1 is half the total expected change in the response value when x_1 changes from -1 to $+1$. A positive β_i signifies positive feedback on the response variable with an increasing value of x_i; a negative β_i signifies a negative feedback under the same conditions.

There are a number of pre-requisites that the underlying distribution of the data must meet for the linear regression modeling to be successful, namely that (1) the error is normally distributed, (2) the error is independently distributed, i.e., one observation does not affect the probability of any other, and (3) the error has mean zero and constant variance. If a linear model adequately approximates the data set, ANOVA will allow us not only to quantify the impact of the factor on the response variable, but also to quantify unexplained variation. This is particularly useful in settings where hidden factors are commonplace, since a large aggregated effect by hidden factors may indicate the need for further experimentation. If these assumptions are broken, the true significance level (i.e., the probability of a false positive) of any statistical tests performed with the model (e.g., testing the statistical significance of a factor) will differ from what calculations report, generally with a loss of statistical power (i.e., the chance of a false negative increases).

Any factors that affect the response variables but were *not* controlled (i.e., do not appear in Table 1) will cause variability that is not attributable to any factor or factor interaction. In ANOVA, this variability will be indistinguishable from any other source of error (such as random measurement error). Experimenters should strive to minimize the amount of error, so that as much variability as possible is adequately explained. The smaller the error, the smaller the effects that will be detectable.

5.1 End-to-End Times

To demonstrate the use of linear regression and ANOVA, we apply them to the end-to-end time data from our experiment. We first verify that the assumptions of ANOVA are not violated, by checking whether the residuals — the distance, or error, by which each observation differs from what the model estimates it should be — are normal, and do not present a pattern or heterogeneous variance. Residual plots verify these properties, by relating the response variable values predicted by the model (plotted on the x-axis) with their distance from the actual, measured values that compose our data set (y-axis). In other words, for any combination of factor levels (values of the x_i variables) a value of y can be calculated (i.e., the model can predict a response value for a factor level combination). The residuals are simply the differences between these predicted values and the values that were experimentally observed for each combination of factor levels. Residual plots should show no discernible pattern when there is no systematic error (e.g., from faulty experimentation or analysis). Figure 3 shows that the residuals for this data have no discernible pattern or widely heterogeneous variances. A normal probability plot of the residuals (omitted for

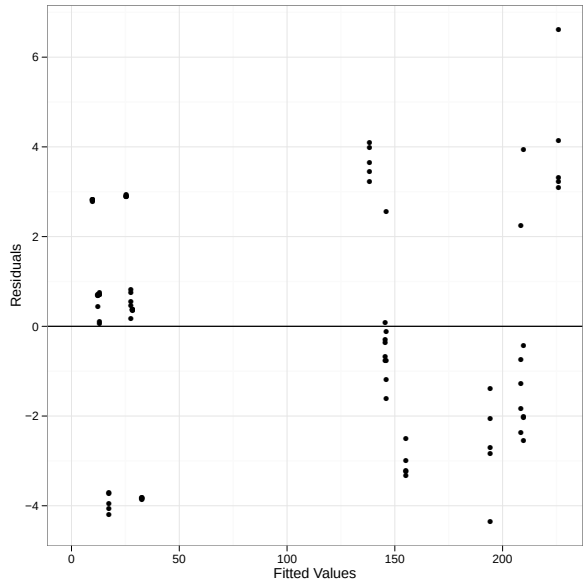


Figure 3. Residual plot for the linear model of end-to-end time response variable, showing no error patterns.

brevity) showed that the normality assumption held. Therefore, linear regression produced a model with a good fit, and we can be confident in the conclusions drawn from ANOVA. The more pronounced residual (with a value higher than 6) did not adversely affect the model[3].

Table 3 shows the ANOVA results for end-to-end times. The source column lists the source factor for the variability. The effect column lists the values of β_i calculated by linear regression, and the boundaries of their 95% confidence interval. The sum of squares column shows the total variability attributable to each factor (i.e., the higher the sum of squares attributed to each factor, the more variability was caused by that factor). The p-value column shows the statistical significance of each factor's effect. For example, a significance level of 0.01 means that a factor is significant (i.e., has a non-zero effect) at the 99% confidence level. The statistical significance of an effect does not indicate its magnitude, which is captured by the β_i model coefficients (c.f., the effects column).

The residuals (shown in the last row of Table 3) allow us to quantify the effect of all uncontrolled factors that affected end-to-end times in this experiment. The residual sum of squares, 545, is an upper bound on the variability attributed to the sum of all hidden factors. This value is negligible in comparison to the machine factor (with a sum of squares of 493,851, three orders of magnitude greater). Therefore, for this experiment, in this design space, the effect of controlling for any further factors will be negligible and we can conclude that enough experimentation has been conducted. The residuals are also used in calculating the explained variability (or the R^2 statistic) of a model; in this case, the explained variability is of 99.8%. This means that, out of the total variability observed in the data set (total sum of squares) 0.2% is caused by error (residual sum of squares), and the remaining 99.8% is attributable to one of the factors or a combination of them.

To interpret the results of the ANOVA table, we focus on the effect estimate column. The effect estimate for the machine factor ($78.6s$) clearly shows that it is the defining factor for end-to-end times, with a main effect estimate far larger than that of any other main factor. The estimate of the interaction between the machine and load ($19.7s$) can be interpreted as follows: depending on the machine, the load factor affects end-to-end times at a different rate. In other words, changing the load type incurs a larger increase in end-to-end time on the slower machine than on the faster one. Several other factors and interactions are statistically significant at the 99% confidence level (i.e., have p-values lower than 0.01) but have comparatively negligible effects.

Most notably, these results show that the scheduler factor, despite being statistically significant, had a relatively negligible impact on the execution time of the benchmark. This means that (1) if end-to-end time is the user's only concern, then the scheduler choice will be inconsequential and (2) end-to-end time does not differentiate these two schedulers. Thus, end-to-end time is a poor metric to benchmark these two schedulers' performance, unlike it has been discussed in the Linux community.

Factor	Effect (s)			Sum Sq.	p-value
	Low 95%	Estimate	High 95%		
machine	77.9	78.6	79.2	493,851	< 0.01
load	11.4	12.0	12.6	11,497	< 0.01
sched	2.5	3.1	3.7	764	< 0.01
vmstat	2.4	3.0	3.6	725	< 0.01
machine:load	19.1	19.7	20.3	31,004	< 0.01
machine:sched	2.4	3.1	3.7	750	< 0.01
machine:vmstat	2.3	3.0	3.6	708	< 0.01
load:sched	1.4	2.0	2.7	333	< 0.01
load:vmstat	1.0	1.7	2.3	219	< 0.01
sched:vmstat	-0.1	0.5	1.1	19	> 0.05
Residuals				545	

Table 3. ANOVA results for end-to-end times.

5.2 Latency

Although ANOVA works well for end-to-end time measurements, our main interest is the worst-case of the latency response variable. We now apply ANOVA to latency, and again, must verify if the method applies. Figure 4 shows a latency histogram for the following combination of factor levels: CFS scheduler, the Core 2 machine, no VMStat, and the compile load. This distribution is highly non-normal, which, in turn, can cause residuals to also be non-normal; this is the first indication that linear regression is inapplicable.

Figure 5 shows the residuals plotted against the fitted values. In contrast to Figure 3, we see a clear pattern of increasing variability as the latency values themselves increase. While this problem can be mitigated by transforming the data into a form with constant variance and then applying ANOVA to the transformed data, these residuals are also not normally distributed. Figure 6, a normal probability plot of the residuals, demonstrates this; if the residuals were normal, they would lie alongside the solid, horizontal line — which indicates a perfectly normal distribution — following no discernible pattern. Non-normality of residuals causes errors in the significance of the estimated coefficients, and is difficult to resolve through data manipulation. The application of linear regression on the latency data resulted in a poorly fitting model, with an explained variability of 12.19%.

If we were to ignore these warning signs and attempt to choose a scheduler using this analysis, we would be basing our decision on a problematic model. The effect for the scheduler factor (i.e., the β_i value corresponding to the scheduler choice) given by this poorly fitting model is of $796\mu s$ (± 69 at a 95% confidence level). Consider, then, the plot for Experiments 4 in Figure 2(a), where the difference in mean latency between the CFS and the BFS is of

[3] We used Cook's distance, a metric designed to quantify the influence of outliers on a model, to verify this [17].

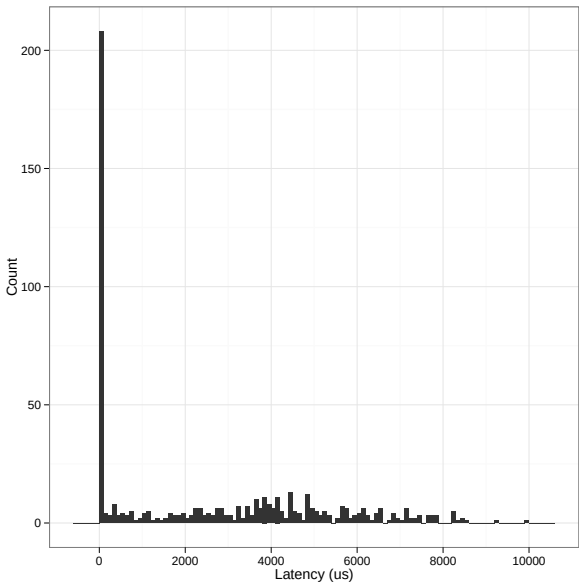


Figure 4. Example of non-normal latency behavior on the Core 2 machine, under the CFS, with no VMStat and the compile load.

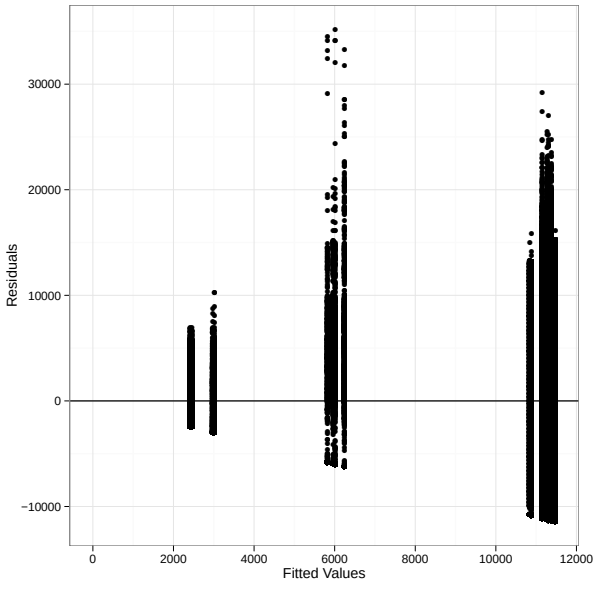


Figure 5. Residual plot for the linear model of the latency response variable, showing heterogeneous error variance.

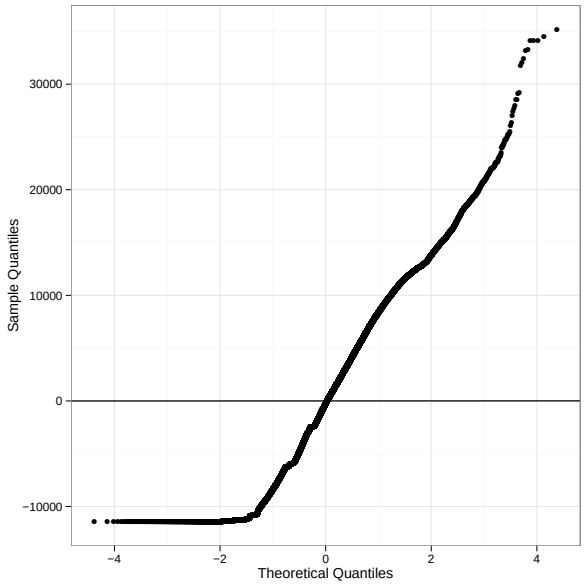


Figure 6. Normal residual plot for the linear model of the latency response variable, showing non-normal errors.

5453μs; Experiments 3 and 5 show similar differences. The conditions of Experiments 3 through 5 are in this experimental space, so the results of those experiments should be approximated by this model. However, the model suggests that, on average, a change in mean latency of approximately 1600μs would be observed by a change of scheduler. That is a severe underestimation, and might lead to incorrect conclusions.

Most important of all, however, is the fact that the worst-case latency is of special importance for latency-critical applications. This means that using linear regression, a method to model means, would not allow the analysis of the effects on the right tail of the response variable distribution, providing instead an incomplete picture containing only the mean latency value. As we will show in Section 6, the scheduler effect discussed before will be even more conservative, if the worst-case is of any interest. Since the non-normal data seen here is common in computer experiments [4, 7, 19], a different analysis method is needed. We now apply quantile regression to analyse this same data set, determining each factors' effects on worst-case behavior, with the added bonus of not requiring normally distributed data.

6. Quantile Regression

Quantile regression [12], like linear regression, is a method to model the effect of factors on a response variable. The main difference between the two methods is that, while linear regression models effects on the mean of the response variable, quantile regression can model the effect of factors on *any given quantile*, such as the median or the 99th percentile. By performing multiple quantile regressions on different quantiles of a data set, a researcher can quantify the effect of a factor all along the response variable's probability distribution, and get a more detailed idea of how it is affected by the factors of interest.

Quantile Regression

An optimization-based statistical method to model the effect of factors on *arbitrary* quantiles.

Output: A model of any quantile of the response variable as a linear function of factor levels.

Distribution Assumptions: None.

Although the probability distribution of a performance metric will depend on many factors (such as processor speed), the effect of factors on this distribution can be more complex than a simple change in its mean. Indeed, factors may affect only *part* of the overall distribution. Consider a hypothetical CPU-bound benchmark. The best case performance of this benchmark occurs when it is entirely in cache at the beginning of measurements; in this case, memory latency is irrelevant to performance. In other executions, it may be partially cached or entirely out of the cache; in these scenarios, memory latency will affect performance to different extents. Applying linear regression in this scenario (investigating memory latency as a factor) would lead to a model of the mean behavior, overlooking the fact that memory latency affects *different quantiles of the response variable at different rates*. Quantile regression, on the other hand, is able to capture these details.

Linear quantile regression models are very similar to the linear models described in Equation 1. They take the form

$$\begin{aligned} Q_y(\tau|X) &= \beta_0(\tau) + \beta_1(\tau)x_1 + \beta_2(\tau)x_2 + \ldots + \beta_k(\tau)x_k + \\ &\quad \ldots + \beta_{12}(\tau)x_1x_2 + \ldots + \beta_{ik}(\tau)x_ix_k \end{aligned} \quad (2)$$

where $\tau \in [0, 1]$ is the quantile of interest[4] of the response variable y, X is the combination of factor levels, and $Q_y(\tau|X)$ is the τ^{th} conditional quantile of the response variable y (i.e., the τ^{th} quantile of y given factor levels X). As with the linear model, the β_i coefficients represent the expected change in the conditional quantile as the corresponding factor levels change.

Quantile regression also differs from linear regression in that it is non-parametric, that is, it does not assume anything about the distribution of the error component. This is because parameter estimates are only affected by the local behavior of the response variable near each quantile of interest. While the application of linear regression on our non-normal latency data yielded an incorrect model because the data failed to meet requirements, we now demonstrate that quantile regression can successfully analyse the same data set.

To quantify the effect of each factor at different points on the latency probability distribution, we apply ten quantile regressions to the dataset: nine from the 10^{th} percentile through the 90^{th} percentile at equal intervals, then one at the 99^{th} percentile. Coefficients were estimated for all factors and interactions up to the four-factor interaction. Figure 7 shows all main factors, the scheduler by machine interaction, and the intercept. The intercept, or the $\beta_0(\tau)$ coefficient, does not depend on factors to affect the response variable; it can be thought of as the baseline latency at each of those quantiles before factor effects are applied. The x-axis shows the quantiles of the latency probability distribution, and the y-axis shows the magnitude of effects at each of those quantiles. A black line is plotted through the estimated effects, and the grey band denotes the 95% confidence interval for the effects. For example, consider the machine factor subplot in the figure. This factor has no effect at the 10^{th} percentile (0.1 on the x-axis). Towards the right of the plot, the effect of this factor starts to grow, ultimately reaching 5,594.5μs at the 99^{th} percentile. This means that, all else being equal, a change from the Core 2 machine to the P4 will increase the value of the 99^{th} percentile of the response variable by approximately 11,189μs (as with ANOVA in Section 5, the net increase is doubled due to the coding used for factor levels). Similarly, a negative effect like the scheduler by machine interaction will cause a quantile to decrease in value.

The first observation of note is the significant effect of the scheduler on latency. The scheduler subplot in Figure 7 shows that this factor has no effect below the 40^{th} percentile, but then its effect gradually increases with each percentile, eventually reaching 2,244.1μs at the 99^{th} percentile. The difference between the BFS and the CFS near the worst-case latency will therefore be of approximately 4,488.2μs. While this latency may seem to be low, it accounts for *more than 10% of the single worst-case latency observed in all our factorial design trials*, which was of 41,164μs. A 10% improvement in worst-case latency would make a significant interactivity impact in a mobile system that operates near the 100ms threshold.

Table 4 provides a list of all effects on the 99^{th} percentile, the highest percentile shown in Figure 7. Similarly to Table 3, which presented the ANOVA results for the end-to-end time, the effects column lists the expected change in the 99^{th} conditional percentile when each factor level changes and all else remains equal, accompanied by the high and low boundaries of their 95% confidence interval. The statistical significance of each effect is shown in the p-value column. At this extreme percentile, the machine has the largest effect, closely followed by the scheduler by machine interaction, and then the scheduler main effect. We can conclude that, if the user is concerned about the worst-case latency, then the BFS will provide better performance.

Figure 7 and Table 4 permit further conclusions. Latency, for most quantiles, is independent of the load factor; Figure 2(a) also shows this, where the difference between the latency measured in Experiment 3 (compile load) and 4 (encode load) was minimal. The difference between the two experiments is concentrated in the lower quantiles, where most of the load factor's effect is located. We can also observe that the VMStat factor, while largely indistinguishable from zero, becomes more pronounced after the 90^{th} percentile. Table 4 shows that interactions with the VMStat factor are also significant at that quantile. Figure 2(a) for Experiment 5 already indicated this, where adding VMStat significantly decreased the worst-case latency of the CFS.

To summarize, quantile regression (1) allowed us to analyze a data set that linear regression failed to model correctly, and (2) provided a higher level of detail than linear regression ever could. The first point is important because non-normality is common in computer science experimental data; ANOVA yielded a poor model because of the non-normality and different variances in the latency data, while quantile regression provided insight into the data despite these properties. The second point is important because when researchers have more than a model of mean response (or indeed, simply a test of whether samples originate from the same distribution), they can gain more insight from their experiments. For example, the fact that the scheduler has an increasingly larger effect as latency values increase — making it very important for worst-case latency analysis — would be impossible to detect through ANOVA or any form of analysis that focuses on the response mean.

[4] The τ^{th} quantile of a distribution y is the smallest observed value in y such that the probability of obtaining smaller values of y is τ.

Factor	Effect (μs) Low 95%	Estimate	High 95%	p-value
machine	5,199.7	5,594.5	5,989.3	0.000
sched	1,849.3	2,244.1	2,639.0	0.000
vmstat	-947.9	-553.0	-158.2	0.021
load	-342.0	52.9	447.7	0.825
sched:machine	-5,087.5	-4,692.6	-4,297.8	0.000
sched:vmstat	-940.7	-545.9	-151.0	0.022
machine:vmstat	68.7	463.5	858.3	0.053
load:vmstat	-101.0	293.9	688.7	0.220
machine:load	-114.0	280.9	675.7	0.241
sched:load	-668.6	-273.8	121.1	0.254
sched:machine:vmstat	215.5	610.4	1,005.2	0.011
sched:machine:load	82.7	477.5	872.3	0.046
machine:load:vmstat	-636.5	-241.6	153.2	0.314
sched:load:vmstat	-232.6	162.3	557.1	0.499
sched:machine:load:vmstat	-602.827	-208.000	186.828	0.386

Table 4. Quantile regression coefficients for the 99^{th} percentile.

7. Further Pitfalls

While we conducted the experiments for this paper, several experimentation pitfalls became obvious. This section discusses those pitfalls and their proposed solutions.

Autogroup as a Hidden Factor The experiment described in Section 4.5 demonstrated how a small, seemingly innocuous change of experimental conditions can significantly affect results. In that case, the addition of VMStat calls to Latt caused a significant change in latency behavior. This was caused by the autogroup option in the Linux kernel (named `CONFIG_SCHED_AUTOGROUP` in the configuration options), which causes the CFS to schedule processes in the same TTY as a group.

We confirmed this hidden factor with Apache's *ab* benchmark. The *ab* benchmark is an automated HTTP client that measures how many requests per second Apache can serve. To demonstrate this hidden effect, we executed Apache and *ab* in two configurations: (1) in the same TTY, and (2) in different TTYs. Figure 8 shows that, while the mean execution time remained unaffected (verified with a t-test, $P > 0.05$), executing *ab* in the same TTY as Apache caused the best observed execution time under the CFS to decrease by 1.37s, and the worse case to increase by 0.33s, that is, the spread of measured values seemed to increase. This was verified through an F-test, which showed that the ratio between the *variances* of the two CFS distributions was approximately 0.2. In other words, autogroup made the execution time variance measurably larger for the CFS. On the other hand, this does not affect the BFS ($P > 0.05$ for both t and F-tests), as it ignores the autogroup kernel option. It is reasonable to assume that some server administrators are unknowingly measuring unrealistic performance because of this hidden scheduling artifact.

Incorrect Surrogate Metrics During our experiments, Latt's client throughput metric proved to be a poor surrogate for application throughput. We found that an execution of Latt could have a lower mean client throughput, but a higher end-to-end time than another execution, which can be easily interpreted as a fault in Latt. The surprising result is due to the fact that Latt does not tally time spent in the server thread, and therefore, the client throughput metric only captured a part of the total execution time of the benchmark. After discovering this, we started using the end-to-end time of a useful computation (compilation or video encoding) to provide a reliable metric of throughput.

Choice of Background Load "Clean room" experimentation may lead to incorrect or misleading results, as shown by the latency measurements for Experiments 1 and 2 in Section 4. By keeping the processor largely idle, the idle load yielded latency data that did not distinguish the two schedulers. Similarly, having a single Latt client perform no work failed to yield realistic latency data. Because of this, we encourage the exploration of the experimental space with a focus on determining not only what factors affect the response variable, but also how the response variable behaves at the different levels of those factors.

8. Related Work

Several researchers have investigated the difficulty of correctly evaluating computer performance. Mytkowicz et al. [18] demonstrate that seemingly innocuous experimental setup details, such as the UNIX environment size or the benchmark link order, can have a significant impact on performance. Harji et al. [8] show that the Linux kernel has had a series of performance affecting issues, and that papers that present data measured on Linux could contain incorrect results. Kalibera et al. [11] show that random symbol names generated by a compiler lead to different memory layouts at run time, and, consequently, random variations in performance. In this paper, we describe encounters with several hidden factors, and demonstrate the pitfalls of conducting sequential, undesigned experiments in their presence. We demonstrate that, for a fixed experimental space, ANOVA can be used to quantify the total effect of hidden factors and assist in determining when they have been sufficiently controlled.

Other previous work has argued that more statistical rigor is needed in computer science. Vitek and Kalibera [22] report that in PLDI'11, a selective conference where experimental results are commonly published, 39 of the 42 papers that published experimental results did not report a measure of uncertainty in their data, making the need for more rigorous statistical analysis in computer science clear. Kalibera and Jones [10] raise a similar point, presenting a random effects model tailored to computer experiments, while also noting that current textbook approaches may be insufficient in the field. Georges et al. [5] argue for the use of statistically rigorous analysis methods, however, they only go so far as ANOVA, which we show here to be insufficient in some cases.

Due to its relatively recent development, quantile regression is also gaining acceptance in other fields. Cade and Noon [2] present the method to ecologists, and Koenker and Hallock [13] demon-

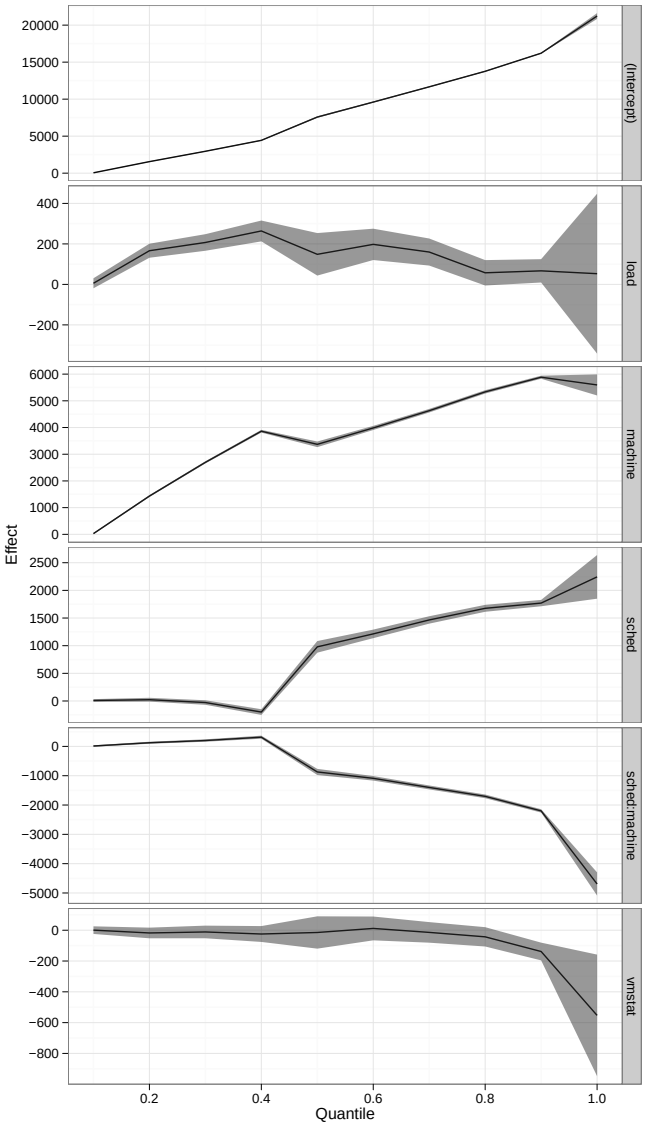


Figure 7. Quantile regression effects for the most relevant factors for the latency response variable.

strate the method to econometricians. We hope that this paper will serve a similar purpose in experimental computer science.

9. Conclusion

Computer performance evaluation is essential to industry and academia alike. However, past research demonstrates that both collecting and analysing performance data is not a trivial process. In this paper, we investigate what is necessary to conduct a successful performance experiment.

First, we attempted to replicate an undesigned experiment, and failed to reach satisfactory conclusions. Instead, a series of contradictions left us with no indication of when we would have conducted enough experiments. Then, we applied textbook design and analysis of experiments with mixed results; while ANOVA worked well for one metric, it failed to provide insight on another.

Finally, we successfully applied quantile regression, a recent development in statistics, to the non-normal data set, and gained insights that ANOVA (or indeed any statistical tool that focuses solely on the mean value) would have failed to provide. We believe that quantile regression is a useful tool for computer scientists, given the abundance of data to which methods that assume normality do not apply.

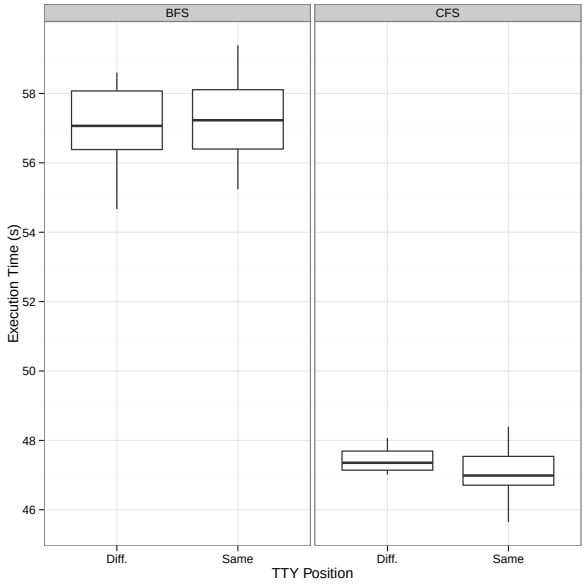


Figure 8. Effect of autogroup on the *ab* benchmark.

Acknowledgments

This research was supported in part by NSERC DG 357121-2008, ORF-RE03-045, CFI 20314, CMC, and the industrial partners associated with these projects.

References

[1] J. Axboe. Latt benchmark. http://git.kernel.dk/?p=latt.git;a=summary.

[2] B. S. Cade and B. R. Noon. A gentle introduction to quantile regression for ecologists. *Frontiers in Ecology and the Environment*, 1(8):412–420, 2003.

[3] S. K. Card, A. Newell, and T. P. Moran. *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1983. ISBN 0898592437.

[4] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Trans. Netw.*, 5(6): 835–846, Dec. 1997. ISSN 1063-6692. doi: 10.1109/90.650143. URL `http://dx.doi.org/10.1109/90.650143`.

[5] A. Georges, D. Buytaert, and L. Eeckhout. Statistically rigorous Java performance evaluation. In *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications*, OOPSLA '07, pages 57–76, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-786-5. doi: http://doi.acm.org/10.1145/1297027.1297033. URL `http://doi.acm.org/10.1145/1297027.1297033`.

[6] Gmane. Linux Kernel Mailing List - BFS vs. mainline scheduler benchmarks and measurements. http://thread.gmane.org/gmane.linux.kernel/886319.

[7] M. Harchol-balter. The effect of heavy-tailed job size distributions on computer system design. In *Proc. of ASA-IMS Conf. on Applications of Heavy Tailed Distributions in Economics*, 1999.

[8] A. S. Harji, P. A. Buhr, and T. Brecht. Our troubles with linux and why you should care. In *Proceedings of the Second Asia-Pacific Workshop on Systems*, APSys '11, pages 2:1–2:5, New York, NY, USA, 2011.

ACM. ISBN 978-1-4503-1179-3. doi: 10.1145/2103799.2103802. URL http://doi.acm.org/10.1145/2103799.2103802.

[9] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley-Interscience, New York, 1991.

[10] T. Kalibera and R. Jones. Quantifying performance changes with effect size confidence intervals. Technical Report 4–12, University of Kent, June 2012. URL http://www.cs.kent.ac.uk/pubs/2012/3233.

[11] T. Kalibera, L. Bulej, and P. Tuma. Benchmark precision and random initial state. In *Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunications Systems (SPECTS)*, pages 853–862. SCS, 2005.

[12] R. Koenker and J. Bassett, Gilbert. Regression quantiles. *Econometrica*, 46(1):pp. 33–50, 1978. ISSN 00129682. URL http://www.jstor.org/stable/1913643.

[13] R. Koenker and K. F. Hallock. Quantile regression. *Journal of Economic Perspectives*, 15(4):143–156, Fall 2001.

[14] C. Kolivas. FAQS about BFS. v0.330. http://ck.kolivas.org/patches/bfs/bfs-faq.txt.

[15] D. J. Lilja. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, 2000.

[16] I. Molnar. Design of the CFS scheduler. http://people.redhat.com/mingo/cfs-scheduler/sched-design-CFS.txt.

[17] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 2006. ISBN 0470088109.

[18] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney. Producing wrong data without doing anything obviously wrong! In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '09, pages 265–276, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-406-5. doi: http://doi.acm.org/10.1145/1508244.1508275. URL http://doi.acm.org/10.1145/1508244.1508275.

[19] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Trans. Netw.*, 3(3):226–244, June 1995. ISSN 1063-6692. doi: 10.1109/90.392383. URL http://dx.doi.org/10.1109/90.392383.

[20] Tukaani Project. The .xz file format. http://tukaani.org/xz/format.html.

[21] VideoLAN. x264. http://www.videolan.org/developers/x264.html.

[22] J. Vitek and T. Kalibera. Repeatability, Reproducibility, and Rigor in Systems Research. In *Proceedings of the Ninth ACM International Conference on Embedded Software*, EMSOFT '11, pages 33–38, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0714-7. doi: 10.1145/2038642.2038650. URL http://doi.acm.org/10.1145/2038642.2038650.

[23] Xiph.Org Foundation. Xiph.org Video Test Media. http://media.xiph.org/video/derf/.