# L41: Lab 1 - I/O - Analysis Notes

Dr Robert N.M. Watson

Michaelmas Term 2015

This document provides notes on the data collection and analysis required for *Lab 1 - I/O*. It is not a sample solution and is hence not presented in the format of a lab report.

## Notes on statistics and graphs

To achieve full marks on this lab, reports will have been properly typeset in LaTeX (or formatted in Word, etc), illustrated using suitably presented graphs, and made reasonable use of basic statistical techniques such as means, medians, standard deviations, inter-quartile ranges, and simple statistical tests such as Student's *t*-Test. Graphs in these notes are reasonable examples of how the data might be presented – but not the only acceptable approach.

There is some debate in the systems research community regarding the use of means vs. medians in performance analysis. In most of these graphs, I present medians with errors bars based on the 25th and 75 quantiles (the inter-quartile range) rather than using means with error bars representing one standard deviation. This is because performance measurements of the sort we are using are unlikely to be normally distributed – at the very least, they will not be symmetric as latency tails are often quite long, meaning that standard-deviation-based error bars would be misleading. In addition, performance-measurement samples are almost always, themselves, means – as they are frequently rates of events or progress over time. This can further cause misleadingly narrow standard deviations as they will be with respect to means of means. One reason to use means, and make assumptions of a normal (or similar) distribution is to provide access to statistical tests and confidence intervals that can be built on those stronger assumptions – e.g., Student's *t*-Test.

For the purposes of this lab, either means with standard deviation or medians with quantiles will be accepted, and most published systems research papers will employ means, standard deviations, and means-based analysis techniques rather than medians and inter-quartile ranges.

## I/O buffer size

The lab assignment requests that you measure, report, and to some extent explain, the performance of the `read(2)` system call as the buffer size is varied. In this sample analysis, only `io-static` (the statically linked version of the benchmark) is used. Power-of-two buffer sizes up to 8 megabytes divide evenly into the 16 megabytes default file size for the benchmark. Eleven samples are run at each size, with the first thrown away, to facilitate analysis of the benchmark in its steady state. A simple shell script was used to save output of each run in a `.csv` file for analysis and presentation using Python's `pandas` and `matplotlib` libraries. The resulting graph is illustrated in Figure 1. A few things to observe:

- The X axis represents the *independent variable* – one controlled as an explicit parameter to the experiment using the `-b` block-size argument to the `read(2)` system call. The X axis is log scale in order to allow us to analyse both its behaviour with extremely small buffer sizes (e.g., 8 bytes) and also large buffer sizes (e.g., 16 megabytes). Because the benchmark requires that blocksize be evenly divisible into the total I/O size, we selected to run the benchmark for all power-of-two sizes between 8 bytes and 16 megabytes, causing points to be evenly distributed over the log scale. In some places in our analysis, more granular buffer sizes might allow for more certain or detailed interpretation of the data – e.g., the actual local maxima in the graph.

- The Y axis represents the *dependent variable* – the kilobytes/sec I/O rate accomplished via the `read(2)` system call – one we are measuring in our experiment, and for which we want to try to establish a relationship with the independent variable. Your graphs might instead show the inverse (i.e., the time to complete

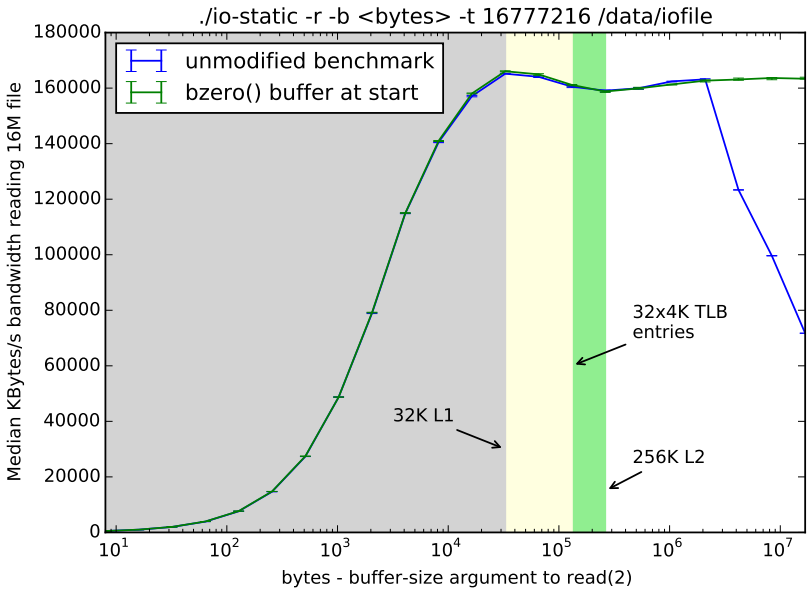


Figure 1: I/O benchmark performance reading a constant-sized file while varying the `read(2)` system-call `nbytes` argument (and hence buffer size).

runs of the benchmark), which would also be fine – as long as it is the time spent in the I/O loop (as specified in the assignment) rather than a whole-program measurement.

- Error bars in the Y axis show little variation, reflecting a carefully controlled benchmarking environment – e.g., no other significant activity on the system, and specific steps in the benchmark to allow the system to enter a steady state before beginning measurement. Micro-architectural thresholds only rarely align perfectly with actual inflection points in performance curves – often, effects will be at constant multiples or divisors of cache size. As caches are infrequently fully associative, aliasing affects may be visible as increasing variance as thresholds are approached – e.g., as the TLB limit is reached.

- Several micro-architectural thresholds are illustrated on the graph to place performance curves in context.

- During the first portion of the graph, roughly up to a measured buffer size of 32 kilobytes, performance increases smoothly as system-call costs are increasingly amortised by the buffer size. Amortised cost include the system call itself, looking up the file-descriptor object, setting up data-copy operations, etc. Performance rapidly exceeds the expected throughput of a Micro SD Card, which should lead us to conclude that the read workload is being serviced entirely from the buffer cache (in memory) rather than disk.

- As the blocksize approaches the closely scaled L1 cache, L2 cache, and TLB limits, growth flattens. The data-cache footprint of data copying will be, at minimum, twice the size of the data being copied, reflecting both the source (pages in the buffer cache) and the destination (pages in user memory). In practice, the footprint will be somewhat larger to account for user and kernel stack space required to process the call, kernel data-structure footprint, less than full associativity, and so on. In this system and benchmark configuration, peak throughput is achieved as amortisation of per system-call costs and full utilisation of the L1 or L2 caches is accomplished – roughly in the range between 32 and 128 kilobytes. Further measurements would allow better exploration of the precise shape of the curve in that optimal area.

- We have plotted two versions of the benchmark: an unmodified version of `io-static`, and a version version modified to explicitly zero the the buffer prior to entering the benchmarked I/O loop. As discussed in lecture, the *process model*, in which fresh pages of memory are pre-filled with zeroes, is far from free: pages will be zeroed on demand as they are first touched. With the unmodified benchmark, a sharp loss in performance is seen when the buffer size exceeds approximately 2 megabytes – explained by a lack of

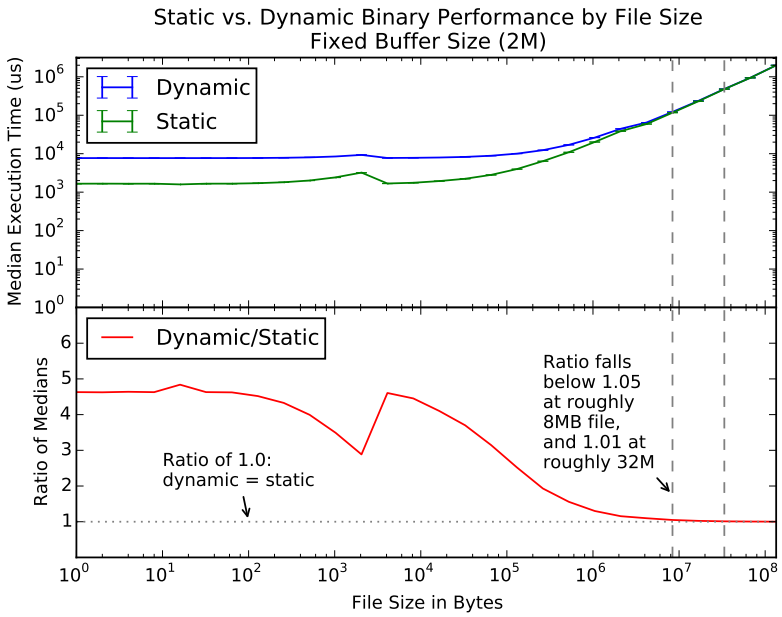


Figure 2: Statically vs. dynamically linked benchmark binaries measuring the time taken to read a file as the file size increases. The first graph shows the absolute times for the two cases; the second graph shows a ratio of their values to measure their relative difference. A difference in execution time below 5% and 1% are marked.

userspace initialisation of those pages, causing the kernel to perform additional work on its behalf. This work that grows (steeply) in proportion to blocksize, as larger and larger quantities of memory must be touched, dirtying more and more of the cache as there is less and less reuse of memory due to buffer size approaching total I/O size. This is particularly observable at a blocksize of 16 megabytes, in which case no amortisation of buffer preparation occurs. The modified version shifts all initialisation out of the benchmark loop – causing performance to remain consistent as increasing numbers of user pages are written to by the `read(2)` system call.

Further avenues of investigation might include a more detailed analysis of the performance hit around 1 megabytes – the details are non-obvious, and likely relate to a combination of the *superpage* size (1M), `jemalloc` *arena* size (8 megabytes), but also the kernel's behaviour during I/O – for example, whether it is able to promote to superpages during a `read(2)` system call.

## Static vs. dynamic linking

The lab assignment also asked you to measure and analyse the whole-program performance impact of static vs. dynamic linking – in particular, identifying a file-size threshold at which the difference falls below 5% and 1% thresholds. Figure 2 illustrates the results of benchmarking 10 runs each of both statically and dynamically compiled versions of the benchmark; buffer size is held constant at 2 megabytes (offering best performance per prior analysis), while total I/O size is varied between 1 byte and 128 megabytes. There are 21 runs of each configuration, with the first instance discarded. The following observations might be made about the graph:

- Due to the range of file sizes and I/O rates, the top half of the graph utilises a log-log graph. This allows the graph to capture scalability information at vastly different resolutions – 1-byte files and 128-megabyte ones. However, log-log graphs are often misleading in terms of representing asymptotic behaviour, and should be used only with caution and suitable analysis to mitigate potential confusion. The log-log graph appears to illustrate that the an initially substantial difference between the two cases is lost as file size increases.
- The second portion of the graph is a log-linear plot of the ratio of median execution times between static and dynamic cases – which may help alleviate visual misconceptions arising from a log-log graph. A ratio

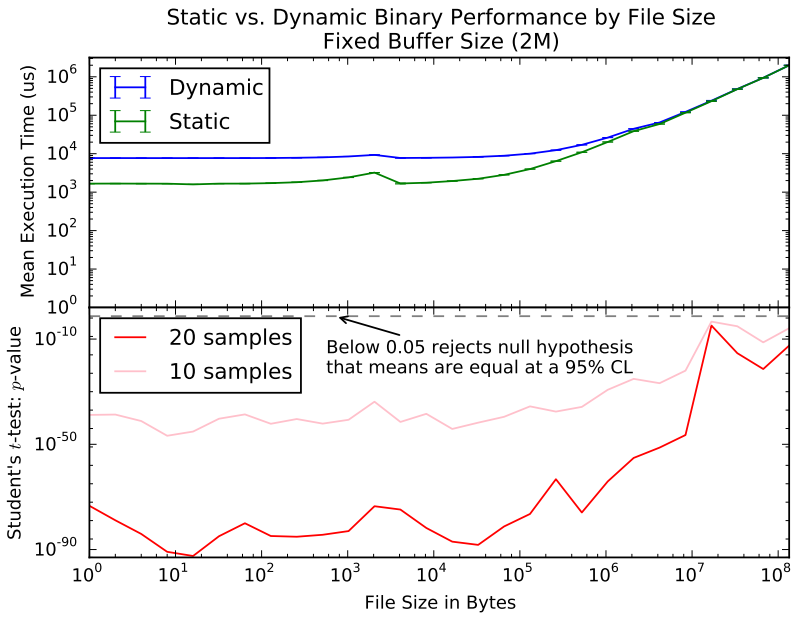


Figure 3: Statically vs. dynamically linked benchmark binaries measuring the time taken to read a file as the file size increases. The first graph shows the absolute times for each case; the second shows the $p$-value from applying Student's $t$-test to statically vs. dynamically linked samples. Two sample sizes (10 and 20) are used to illustrate the effect of sample size on certainty that a result is statistically valid. A $p$-value threshold of 0.05 is marked.

of 1:1 is labelled on the graph to illustrate whether convergence, represented as a percentage difference, is occuring between the two cases. Although the absolute difference is not captured, convergence is indeed observed, with a difference over 400% with a 1-byte file reducing to less than 5% by 8 megabytes, and 1% at 32 megabytes (both marked on the graph).

- Error bars on the graph show little variation – however, with a log Y axis and our earlier observation that an upward tail would be most significant, variance may easily be masked in this representation of the data.

To better understand the statistical significance of our measurements, Figure 3 illustrates a means-based analysis:

- As before, the top portion of the graph represents execution time on a log-log scale, this time using mean execution time and error bars based on standard deviation – a not particularly visibly different graph.
- The second portion of the graph illustrates the results of Student's $t$-test, a statistical test evaluating the probablity that an apparent difference in means for two samples of data reflect an underlying difference in the means in the distributions from which the samples have been taken. The *null hypothesis*, that the two means are drawn from the same distribution and any difference is a result of statistical error in sampling, is compared with the *alternative hypothesis* that a difference in means is actually present in the underlying data. As with standard-deviation-based error bars, Student's $t$-test makes a strong assumption regarding the underlying data sets conforming to a normal distribution – one widely accepted in current systems research practices, but of mixed validity given our understanding of the data.

  In a $t$-test, the experimenter selects a *confidence level* – i.e., an acceptable threshold at which statical error might lead to incorrect accepting of the alternative hypothesis. Current scientific practice frequently makes use of a 95% confidence level, presented by a $p$-value of 0.05. The second graph analyses, for our various samples, whether the $p$-value approaches a level that might allow the null hypothesis (that samples are drawn from the same distribution) to be accepted. To illustrate the effect of sample size on $t$-test behaviour, we have applied the test both to a full 20 samples taken in our measurements, and a truncated 10-entry sample. Note that, at no point, does the $p$-value exceed 0.05, meaning that there is a high probability, given normal assumptions, that there is a statistically significant measurement of difference between the performance of the benchmark when statically vs. dynamically linked, even with large file sizes.