A Statistical Evaluation of the Impact of Parameter Selection on Storage System Benchmarks*

Nohhyun Park, Weijun Xiao†, Kyubaik Choi and David J. Lilja
Department of Electrical and Computer Engineering
University of Minnesota, Minneapolis, MN 55455

Abstract

The performance evaluation of storage systems is a difficult task due to lack of representative workloads. Storage benchmark tools test a single aspect such as random read performance at a time. While this approach does allow us to gain useful insights to the system behavior, current practices are largely inadequate due to large benchmark input parameter space. For example, a benchmark with 10 parameters require at least 1024 experiments if conducted exhaustively even if we only test extreme values of each parameter. Furthermore, there is no way to tell if the benchmark being used is enough to test all realistic scenarios. As a result, researchers and developers rely on multiple benchmarks with ad hoc input parameters.

We propose a method to quickly identify input parameters that have high effect on the performance metric of interest. We also show that using multiple benchmarks is unnecessary at times and a good benchmark can cover all operational space by providing a control over key parameters that affect the performance metric being measured.

1 introduction

Storage system performance is one of the most critical factors in meeting overall system performance expectations. The performance variances of typical storage systems today can vary by several orders of magnitude [1]. It is very easy to get misled by performance numbers without understanding the relationship between the storage workload and its performance variances.

Characterizing the workload for storage systems is typically much more complex than it is for processor. In a simple uniprocessor case, it is possible to calculate metrics such as instruction per cycle based on the various cache misses, mis-predictions and number of instructions. However, storage systems typically have a much more state-dependent response. For example, a cache miss does not always result in same penalty, and the service time for a given request depends not only on the current request but also on all requests in the queue as well as the layout of the data. Therefore, two very similar workloads may perform very differently since the small differences could force the state transition path into completely different directions. For example, current layout of the data could force a sequential write to become fragmented. This then cause the sequential read of the same data to be slow. This in turn has effect on any requests that are scheduled after it within the request buffer.

1.1 Current Approaches and Limitations

The best method for benchmarking a storage system would be to deploy the system under test in the real environment and look at the execution time of the processes that will be run. Obviously this approach is too costly to evaluate various feature and performance enhancements. Therefore, system architects and designers resort to either a set of traces collected from a real system or a set of synthetic benchmarks.

The problem with using a real trace is that there is no way to determine the coverage of the workload space. For example, a given Exchange server trace may perform very well on system A. However, it is difficult to determine if system A will perform as well for all Exchange servers. Furthermore, it is impossible even to provide any range of potential performances with a single trace. As mentioned above, a small change in the trace could result in a very different performance results.

Numerous synthetic benchmarks have been proposed over last few decades. The major benefit of these benchmarks is that they allow you to change the characteristics of the workload in a quantitative manner. However, current benchmarks either provide numerous parameters to set

*This work was supported in part by National Science Foundation grant no. CCF-0621462, the Center for Research in Intelligent Storage (CRIS), which is supported by National Science Foundation grant no. IIP-0934396 and member companies.

†It is also sponsored in part by NSF/CRA Computing Innovation Fellows Project under Grant CNS-0937060(subaward:CIF-187).

without providing any insights to the importance of each parameter [2–7] or they provide a representative set of parameters that claim to represent a wide range of applications. When there are numerous parameters to set, users typically test a few set of ad hoc settings based on their experience which does not always result in an accurate representation of real workloads.

1.2 Our Contributions

In this paper, we evaluate current storage benchmarking tools and analyze the effects of different parameter settings on various performance metrics. Based on the experiments, we quantify the effects of various parameters for two benchmarks. In addition, we also evaluate which parameters are needed to effectively cover the entire storage system performance space and we propose a new benchmark tool based on our evaluation.

1.3 Outline

The next section describes two techniques we use to measure the effect of different benchmark parameters on performance. Section 3 presents our methodology as well as the benchmarking tools we used and their use cases. The numerical results are discussed in Section 4. We conclude the paper in Section 5.

2 Background

Typical storage benchmarking tools have ten or more input parameters. If we let number of input parameters be n, to test all possible combinations would require $\Pi(L_i)$ experiments, where L_i is number of values ith input parameters can take. For a benchmark with 20 parameters each taking one of two extreme values, the number of experiments would be 1,048,576. If each experiment takes 10 minutes, it would take roughly 20 years to complete all the experiments. Clearly, this is a problem.

We present two statistical tools we deploy in order to minimize the number of experiments while providing quantifiable insights to how the input parameters affect the result of the experiment.

2.1 Plackett-Burman designs

The *Plackett-Burman design* (PB design) [8] minimizes the number of experiments required to estimate the effect of independent variables on the dependent variable. Basically, it estimates the effect of single input parameter on an output in such a way that variance of this effect due to other inputs is minimized. The effect is an approximation of the full factorial design but with linear complexity. Required assumption is that higher order interaction effects are negligible. PB design has been widely used to evaluate effect of benchmark input parameters on processor performance [9] and database query processing performance [10], for example.

Formally, the main effect m_1 of input variable x_1 on output y is defined to be

$$m_1 = \frac{[\sum f(x_1', x_2, ..., x_n) - \sum f(\overline{x_1}, x_2, ..., x_n)]}{2^n} \quad (1)$$

where x_1' represents some extreme value of x_1 and $\overline{x_1}$ represents the mean of x_1. f is an unknown function which maps all x_i to y. The summation is carried out for all possible combination of x_2 through x_n. Hence, the effect of x_i is simply maximum deviation of y given maximum deviation of x_i.

The goal of PB design is to estimate all m_is such that

$$S = \sum_j (\widehat{y_j} - M - a_{j1}m_1 - a_{j2}m_2 ... - a_{jn}m_n + \alpha)^2 \quad (2)$$

is minimized. $\widehat{y_j}$ is simply the sampled value of y at time j and M is mean value of all y_js. a_{ji} takes value ± 1 based on whether the x_i had positive or negative effect on $\widehat{y_j}$. α is the higher order effects which are ignored based on the observations made by Fisher [11]. Since there are n values to be estimated, the minimum number of experiments required to derive a unique set of m_i is $n + 1$. Plackett and Burman proved that S can indeed be minimized using $\lceil (n+1)/4 \rceil * 4$ experiments assuming $\alpha = 0$ [8].

2.2 Independent Component Analysis

While the PB method quantifies effect of each input parameters to output of interest, it does have one major drawback. Because the effect estimation assumes any higher order effects to be zero, the estimated effects of any main factor is partially aliased with any interactions not including that particular main factor.

In contrast, *Principal Component Analysis* (PCA) and *Independent Component Analysis* (ICA) estimate the effect of uncorrelated components of original factors [12]. These components may not reflect any intuitively meaningful parameter. However, they do provide a uncorrelated set of factors that can be derived from the original ones. The limitation of the PCA is that it requires the original factors to be normally distributed. ICA is a more generic approach in that such assumption is not required [13]. It has been observed that ICA performs better than PCA in capturing processor workload space [14].

In essence, ICA tries to identify statistically independent components from given factors.

Formally, ICA estimates a matrix $\mathbf{W}$ such that given $\mathbf{x} = (x_1, x_2, ..., x_n)$ with joint *probability density function* (pdf)

Table 1. Description of system under test.

CPU	2*2cores*2SMT Xeon 3GHz
Memory	10GB
OS	Lucid Server
Kernel	2.6.32
FileSystem	EXT4, noatime
Storage Conf.	RAID5
Controller	PERC3
Disks	SAS Cheetah 15K

$p_x(x_1, x_2, ..., x_n)$ such that $\mathbf{y} = \mathbf{W} \times \mathbf{x}$ has joint pdf $p_y(\mathbf{y}$ with following property.

$$E[y_1^{p_1}, y_2^{p_2}, ..., y_m^{p_m}] = E[y_1^{p_1}]E[y_2^{p_2}]...E[y_m^{p_m}] \quad (3)$$

for every integer value of p_i and $m \leq n$. Equation 3 is the definition of independence.

3 Methodology

We use PB design to determine a set of experiments required to evaluate the effect of two different benchmarking tools. ICA is used to evaluate the coverage of storage system operational space by each benchmark.

Many of the parameters provided by the benchmarks are dependent on each other. For example, one parameter might be percentage of seeks that occur while another parameter determines whether IO should be random or sequential. We did our best to separate out the dependent parameters into independent ones based on common sense. We believe a more mathematically provable approach is possible but it is beyond the scope of this paper.

We also classify the parameters as the following.

- System Parameter: Parameters that control the way the storage system behaves. (eg. use of filesystem buffer cache)
- Workload Parameter: Parameters that changes the characteristics of workload. (eg. access pattern)
- Benchmark Parameter: Settings that manages how the benchmark is ran and result is collected. (eg. Duration of run)

This classification allows easier management of benchmark runs. Technically, a benchmark should focus only on the workload and benchmark parameters. However, because many aspects of storage system is controlled through the same interface, it is convenient for microbenchmark designer to provide these capabilities as well. In this paper, we do not evaluate benchmark parameters. While they could have effect on the correctness of the benchmark result, they should have no effect on the performance if correctly measured. For example, as long as the duration of the run is sufficiently long, the average performance of the system should not change much for the same workload parameters.

Another difficulty was deciding what should be the two extreme values for each parameter. Again we resorted to common sense and experience to decide its range. We do not believe that there exist a systematic method to do this. It will continue to change as the systems and workloads evolve with time. We believe this to be more of a philosophical problem in nature and did our best so that the ranges accurately reflect corner cases of systems used today.

One thing to note is that PB design assumes that the input parameters have monotonic relationship with the output. When input parameters represent things like algorithm and API choices, the problem is simplified since you can always order them in such a way that the resulting performance values are monotonic. In this case, it is simply choosing two cases where the resulting performance is either maximized or minimized. The problem is slightly more complex for the quantifiable values. If the relationship between the parameter and the performance is not monotonic, we can choose to relax the constraint by bounding the parameter within the range where the relationship is monotonic. For example, if we know that the larger request sizes typically yield in better throughput within some range, we choose the maxima and minima values for the request size in that range. The effect estimation of the parameter will than become effect estimation of parameter within that range. As long as the range covers most of realistic usage of the parameter, the estimated effect is still useful. Another approach is to divide the parameter into multiple ranges where within each range the relationship is monotonic. While this approach can provide a more generic estimation over wider range of values, the number of experiments doubles with every additional range. This is because the you are essentially doubling the degree of freedom by introducing a mutually exclusive parameters.

Another challenge is that some of the parameters are obviously correlated. While the methodology described in this paper works as long as the high order effects are much smaller than that of the main effects, it is still worth while to transform the parameters into less correlated parameters. For example, file open flag used in FIO can be converted into two separate parameters, [sync, async] and [buffered, unbuffered].

Every experiment was conducted on freshly formatted partition and each of the experiments were repeated five times. Repetition was necessary to provide statistical significance of the effects estimated as well as the results. Table 1 outlines the system under test.

Table 2. PostMark input parameters and input levels assigned to each parameter.

Name	Level Low	Level High
min_file_size	512B	4KiB
max_file_size	4KiB	16MiB
init_file_count	1000	10000
transaction_count	10000	100000
read_size	512B	32KiB
write_size	512B	32KiB
file_system_buffer	false	true
read_append_ratio	1:9	9:1
create_delete_ratio	5:5	9:1
directory_count	1	1000

Table 3. FIO input parameters and input levels assigned to each parameter.

Name	Level Low	Level High
operation	read	write
access_pattern	sequential	random
files_used	1	100
min_file_size	512B	1MiB
max_file_size	1MiB	1GiB
min_block_size	512	4KiB
max_block_size	4KiB	64KiB
io_depth	1	100
overwrite	false	true
fsync	false	true
thinktime	0	1000
write_buffer_sync	false	true
file_service	roundrobin	random
thread_count	1	8
threads_similarity	false	true
posix_fadvise	false	true
async_io_engine	false	true
io_engine_queue	false	true
directio	false	true
buffer_alloc	malloc	mmap

3.1 Benchmark Tools

In this preliminary work, we evaluate two different benchmarks tools, PostMark [4] and FIO [2].

Typically, the evaluated systems are only tested on single configuration of a given benchmark. This is problematic since system A may perform better than system B for one configuration of the benchmark but not the others. It is critical to either define when the system A performs better or show that system A performs better B in wide range of workloads by testing with multiple configurations that provide wide coverage.

3.1.1 PostMark

PostMark was designed for filesystem benchmarking with specific interest to performance of handling small file accesses in Internet software [4]. It provides 10 different parameter settings shown in table 2 together with their level settings. Most of level decisions were straight forward except the *create_delete_ratio*. Initially, we set the low bound to be 1:9 where 10% of the operations are create and 90% deletion. However, we soon realized that the bound is too unrealistic. If the deletion of file happens more than the creation, the system will soon become empty of files. Therefore, we set the low bound to be 50% where the creation and deletion is equally like to occur.

Aside from these, it also provide two more benchmarking parameter for setting the random variable seed and controlling result format. The *transaction_count* only controls the duration of the run and not the arrival rate or the IO depth. Therefore, it is also a a benchmark parameter. The *file_system_buffer* parameter is a system parameter which is forces all writes to bypass the filesystem buffer.

PostMark lacks control over arrival pattern and does not support sequential access pattern. Since these two characteristics play a major role in how the storage system performs, we can guess that the coverage of PostMark will not be great. Also, Postmark does not perform any overwrites which suggest that Postmark is unsuitable for evaluating storage systems where overwrites are expensive such as SSDs.

Postmark reports seven performance metrics, *transaction per second* (tps), create tps, read tps, append tps, delete tps, read throughput and write throughput.

3.1.2 Flexible IO Tester (FIO)

FIO was designed for benchmarking as well as stress/hardware verification [2]. It works directly on block devices as well as files. In this experiment, we tested

it on files for the sake of consistency with PostMark.

FIO has over 30 different input parameters. However, some of those parameters are completely dependent on other parameters and was discarded. Also, some of the parameters, when set, caused bus errors on the system we tested. As a result, we extracted 20 input parameters as shown in table 3. Every experiment was run for 10 minutes., which was sufficient to ensure that the system reached a steady state.

FIO can generate multiple independent streams of requests. In this study, we evaluate the effect of multiple streams as function of two parameters, *thread_count* and *threads_similarity*. The thread_count parameter is self-explanatory. When the thread_similarity is true, we generate either set of very similar streams with only difference being the random seed. When it is false, we generate threads that are at maximum distances from each other in the parameter space. This can be achieved by choosing different experiments within PB design.

FIO also allows you to control number of outstanding requests controlled by *io_depth*. The *posix_fadvise* option allows to you enable file advisory information to the operating system [15]. FIO provides 13 different types of IO engines. Some of them did not work on our system but we use 4 based on the two characteristics of the IO engines, synchronous/asynchronous and multiple/single buffer.

FIO provides a detailed distribution of performance measurements as well as aggregated numbers. In our study, we use 4 metrics, read throughput, write throughput, average read latency and average write latency. These metrics were chosen not only because they are important performance metrics but also so that the results can be compared against PostMark for out coverage analysis. For the experiments with more than 1 thread, the throughputs are simply added and the latency is averaged.

3.2 Experiment Flow

The experiment is designed to evaluate the effect of various parameters of different benchmarks and coverage of each benchmark. We first use PB design to evaluate the effect of each parameter with minimum number of experiments. Once the effect has been calculated, we perform ICA on the performance results from each benchmark to estimate equal number of independent components. These components are clustered to generate a view of coverage of each benchmark.

This lets us analyze what are the key parameters and whether both benchmarks are needed for a through evaluation of storage systems.

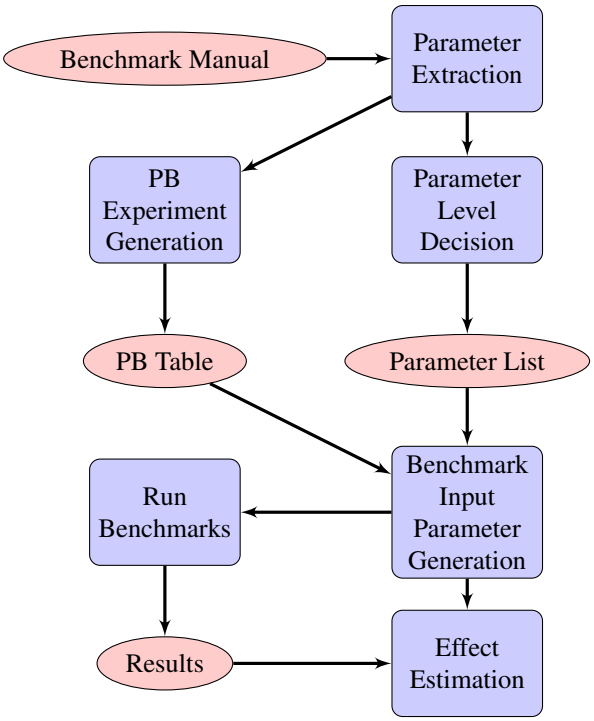

Figure 1. Experiment flow for estimating the parameter effects on performance

3.2.1 Evaluation of Benchmark Parameter Effect

First, we use PB method to determine the effect of each parameters on each benchmarks seperately. The experiment flow for effect estimation is shown in Figure1. *Parameter Extraction* phase tries to extract parameters from benchmark documentation in such a way that each parameter can be varied independently of others. This is not always easy as shown in the description of the benchmarks.

For every parameter extracted, a two extreme level values are designated as shown in table 2 and table 3. Also, once the number of parameter is known, the size of PB design is also known which is required to generate PB design table. Since the number of parameters for Postmark is 10 and FIO is 22, PB table of size 12 and 24 are used respectively. While any Hardamard matrix [16] can be used, we use the matrices shown in equation 4 for 11 factor experiment. PB table for 23 factors can be found in Plackett and Burman's original paper [8] and we omit it due to the space

constraint.

$$\mathbf{PB}(12) = \begin{bmatrix} 1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} \tag{4}$$

Each row of the PB table represents an experiment and columns represent parameters. -1 value represents a *low level* while 1 represents a *high level*. It should be noted that not every column can be assigned a parameter. You can leave any of the columns out for actual run of the experiments. However, it is important to ignore the effect of the unused column later.

The number of experiment required in equivalent to the number of row of the matrix. Every experiment is repeated three times and mean is used to calculate the effect. Given n experiment result $\mathbf{y}$, the effect, e, of a parameter represented by column i of PB matrix is simply calculated by

$$e = \frac{\mathbf{PB}_i \cdot \mathbf{y}}{\sum_i (\mathbf{PB}_i \cdot \mathbf{y})}. \tag{5}$$

Here the denominator is a simple normalization factor to ensure that sum of all effects is 1.

3.2.2 Evaluations of Benchmark Coverage

We use R JADE package [17] to generate 2, 4, 8 independent components from the input parameters and the read and the write throughput measured from each benchmark. Since the read and the write throughputs are the only common metrics reported from the two benchmarks, only they can be directly compared.

Once components are estimated, we evaluate the Euclidean distances between all components which corresponds to a single performance metric are calculated. Based on the distance, the components are clustered. The number of clusters to be used were estimated using dendrograms [18].

We evaluate the best number of components for clean clustering and determine the coverage of each benchmark on two performance metrics.

4 Results

4.1 Effects

Figure2 shows effect of each parameter on each performance metrics reported by Postmark. The *maximum_file_size* and *buffering* are the clearly two most important factor in deciding the overall tps. The *buffering* parameter controls whether file system buffer should be bypassed using direct IO and the importance of the parameter is obvious. However, the effect of the *maximum_file_ size* is less obvious. Potentially, the high effect is due to the fact that larger file sizes allow a room for longer sequential accesses while the small file sizes force overall access pattern to behave in a more random fashion.

You can also see that different performance metrics are effected differently. The *create_delete_ratio* is clearly and obviously important for create tps and delete tps while for append tps, file sizes are more important. Another interesting to note is that while read throughput is affected the most by the *read_append_ratio*, the write throughput is much more impacted by the file size.

Table 4 shows list of parameters with the most effect on each performance metric such that sum of the effects exceed 50%of overall effects. It is shown that roughly 50% of the all performance variances can be captured using just 2-4 parameters. For example, three parameters, *maximum_file_size*, *number_of_initial_files* and *buffering*, contribute to 61% of overall tps variation observed. This equate to 4 to 16 experiments for exhaustive evaluation. This is a huge improvements over 2048 experiments required for original 12 parameters. Furthermore, it is clear that different input combinations must be tested for each performance metric which must be taken into consideration when benchmarking storage systems.

Figure3 shows the effect of each parameter on four performance metrics measured and table 5 shows list of parameters with the most effect on each performance metric such that sum of the effects exceed 50%of overall effects. It is interesting to note that the most important parameters for read and write latencies are the same. We can assume that read and write operations themselves do not affect the latency in our system. This can expected, since on HDD, read and write operations do not result in significantly different response time. Furthermore, we see that the *threads_same* parameter is important for all metrics. This indicates that the interference between the threads can seriously affect the performance. While the *thread_count* parameter is also important, the effect on the read throughput is shown to be negligible. Both throughputs are sensitive to the *access_pattern* which is expected but surprisingly the latency is not. Instead, they are much more sensitive to the *posix_fadvise* setting. This suggests that latency is not de-

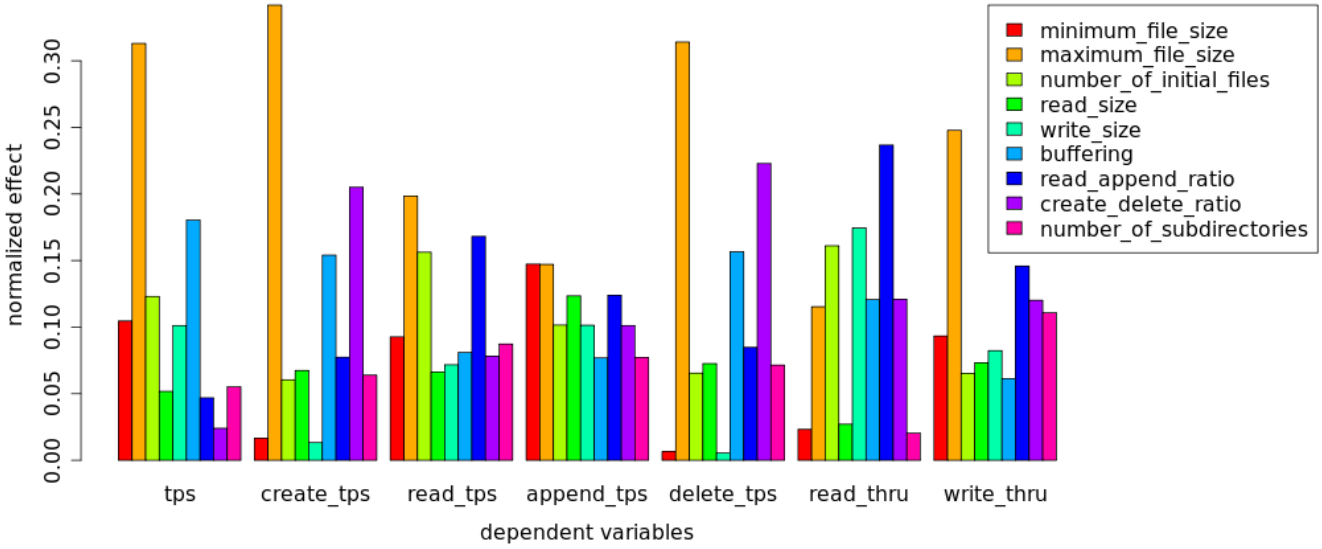

Figure 2. PostMark parameter effects.

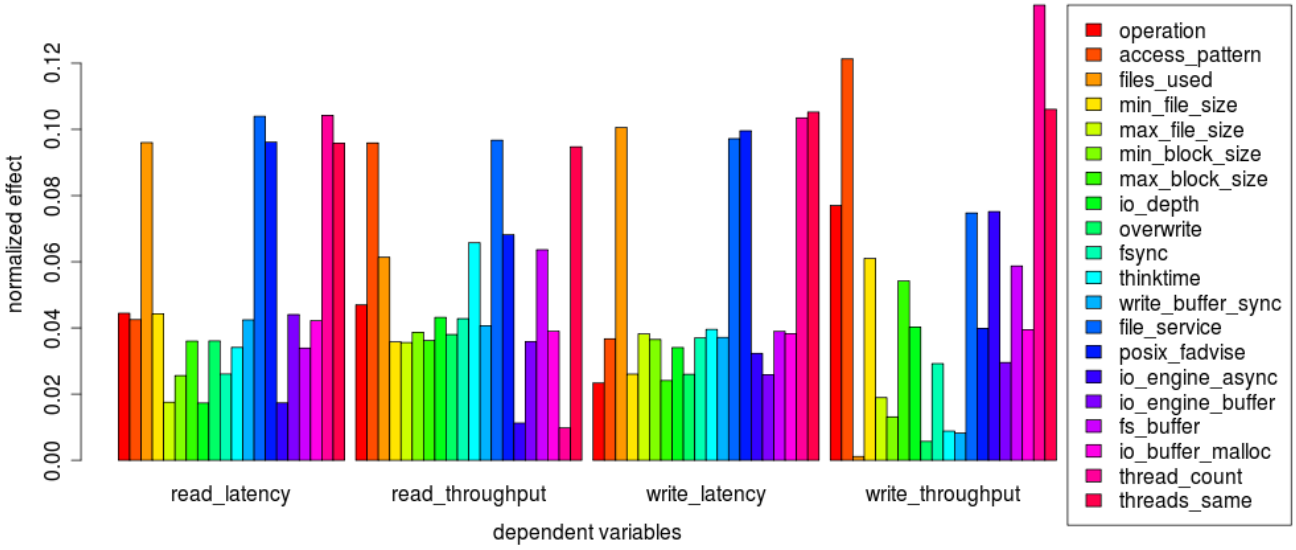

Figure 3. FIO parameter effects.

termined by the seek time but rather cache miss penalties.

4.1.1 Coverage of Benchmarks

The distance between the independent components as well as their clustering is shown in Figure4 and Figure5. In these graphs, each leaf node represents an independent component. Since the independent components have no physical meaning associated with them, we number them from 1 to 8 per benchmark. We only show the results of 8 components for the brevity. The closer the leaves are on the tree, more similar effect they have on the result. The *height* represents the difference between the subtrees. therefore, evenly spread out components of both benchmark on x-axis

Table 4. PostMark input parameters required to cover at least 50% of overall effect for each performance metrics reported.

Performance Metric	Parameters	Total Effect
tps	maximum_file_size number_of_initial_files filesystem_buffer	0.6164
create_tps	maximum_file_size create_delete_ratio	0.5469
read_tps	maximum_file_size number_of_initial_files read_append_ratio	0.5229
append_tps	minimum_file_size maximum_file_size read_size, read_append_ratio	0.5418
delete_tps	maximum_file_size create_delete_ratio	0.5369
read_thru	number_of_initial_files write_size read_append_ratio	0.5722
write_thru	maximum_file_size read_append_ratio create_delete_ratio	0.5137

Table 5. FIO input parameters required to cover at least 50% of overall effect of each performance metrics reported.

Performance Metric	Parameters	Total Effect
read latency	files_used, file_service thread_count, threads_same posix_fadvise	0.5406
read throughput	access_pattern threads_same write_buffer_sync	0.5462
write latency	files_used, file_service thread_count, threads_same posix_fadvise	0.5060
write throughput	thread_count threads_same access_pattern	0.5170

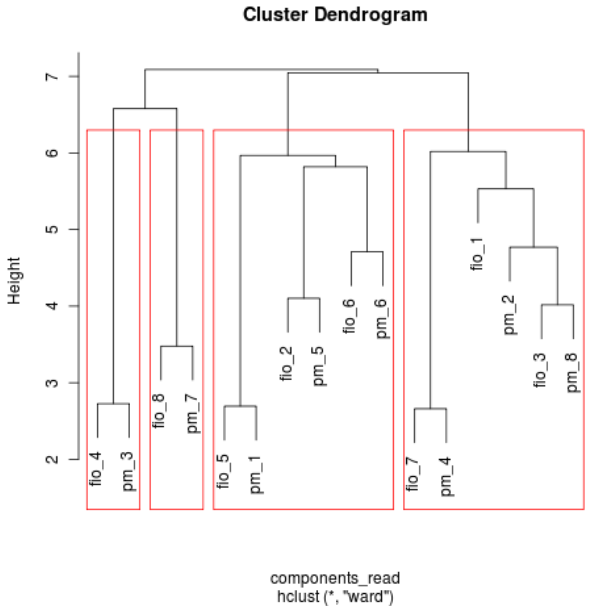

Figure 4. Read throughput performance coverage of PostMark and FIO

suggest similar coverage, while the height of the tree suggest the granularity of the coverage. For example, the FIO component 4 and PostMark component 3 have the similar effect on the read throughput based on the Figure4. At the same time, FIO component 8 and PostMark component 7 also have similar effect. However, there is a large gap between the two sets of components which are not covered by any of the components as suggested by the *height* of the edges connecting the two subtrees.

An interesting observation is that for read throughput, FIO and Postmark have roughly the same coverage. This is interesting since PostMark have far fewer parameters. It can be concluded if the metric of interest is read throughput, than it does not matter which benchmark you use as long as you can test the key parameters thoroughly.

For the write throughput, FIO does provide wider coverage to the left with component 3, while PostMark provide wider coverage on the right with component 3 and 7. We can safely conclude that each benchmark provide one of more input parameters that affect the write throughput that is not provided by the other benchmark.

The input parameters that affect the read and write

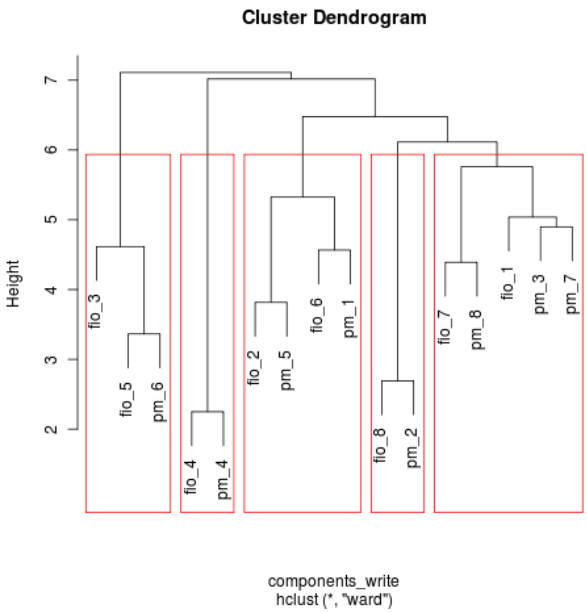

Figure 5. Write throughput performance coverage of PostMark and FIO

throughput performance shows some difference. They all depend on the size of the files, number of files and mix of operations. However, write throughput is also dependent on file creation and deletion which are not taken into account in FIO. Conversely, PostMark does not consider the number of threads which is another important factor provided by FIO.

We believe that a good benchmark should have all those components as inputs to provide a wider coverage.

5 Conclusion

Clearly, any storage performance evaluation should consider the effect of critical input parameters that has been described in this paper. We recommend that at least the critical parameters shown in table 4 and 5 should be tested thoroughly to gain an approximate sense of the system's performance variation.

The methodology presented in this paper can be applied to any benchmarking tool with set of input parameters that can be adjusted. The key challenge is deciding the range of each parameter. Once this is done, the entire process can be automated.

Even if the system is designed for a specific workload, it is still beneficial to follow the methodology described in this paper. Since the characteristics of the actual workload is known, a tighter level bounds can be assigned to each parameter. The resulting effect of the parameters can describe how the system reacts to different parameters and can provide valuable insights to workload-system interactions.

We also show that different benchmarks can still cover roughly the same performance space if the input parameters are chosen appropriately. We suggest that performing a set of through experiments on a single benchmark provides more accurate description of system performance than running a single experiment on the multiple benchmarks.

However, it is clear that there are specific performance space that can only be explored with a specific benchmark. Since it is difficult to determine what benchmarks cover how much of the performance space, we propose a new benchmark with following properties.

- IO benchmark tool should provide maximum coverage of storage system's operational space with minimum parameter settings.
- The parameters of benchmarking tools should be as independent as possible.
- The parameters should be exclusively defined as either the system parameter, the workload parameter or the benchmark parameter.
- The target interface which the benchmark is testing needs to be clearly defined.
- A minimum validation of results should be handled. (eg. Results that suggest under-utilization of target system such that the performance result does reflect its capability.)

With this kind of benchmarking tool, most of the systems can be evaluated with a single benchmark with multiple parameter settings. This tool would allow different researcher and developers to report their performance in a more coherent manner which in turn makes performance comparison and reproducing the result easier.

As a future work, we plan to conduct a comprehensive analysis of more benchmarks and design a new benchmark that provides a wide coverage with few independent parameters.

References

[1] P. Chen and D. Patterson, "Storage performance-metrics and benchmarks," *Proceedings of the IEEE*, vol. 81, no. 8, pp. 1151–1165, 2002.

[2] J. Axboe. fio - flexible io tester. [Online]. Available: http://freshmeat.net/projects/fio/

[3] H. Vandenbergh, *Vdbench Users Guide*, 5th ed., Sun Microsystems, 2008.

[4] J. Katcher, "Postmark: A new file system benchmark," Network Appliance, Inc, Tech. Rep., 1997.

[5] B. Wolman and T. M. Olson, "Iobench: a system independent io benchmark," *SIGARCH Comput. Archit. News*, vol. 17, pp. 55–70, September 1989.

[6] W. D. Norcott and D. Capps. Iozone filesystem benchmark. [Online]. Available: http://www.iozone.org/

[7] R. Coker. Bonnie++. [Online]. Available: http://www.coker.com.au/bonnie++/

[8] R. Plackett and J. Burman, "The design of optimum multifactorial experiments," *Biometrika*, vol. 33, no. 4, pp. 305–325, 1946.

[9] J. Yi, D. Lilja, and D. Hawkins, "A statistically rigorous approach for improving simulation methodology," in *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on*. IEEE, 2003, pp. 281–291.

[10] B. Debnath, D. Lilja, and M. Mokbel, "Sard: A statistical approach for ranking database tuning parameters," in *Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on*. IEEE, 2008, pp. 11–18.

[11] R. Fisher, "Design of experiments," *British Medical Journal*, vol. 1, no. 3923, p. 554, 1936.

[12] J. Stone, "Independent component analysis: an introduction," *Trends in cognitive sciences*, vol. 6, no. 2, pp. 59–64, 2002.

[13] V. Christopoulos, D. Lilja, P. Schrater, and A. Georgopoulos, "Independent component analysis and evolutionary algorithms for building representative benchmark subsets," in *Performance Analysis of Systems and software, 2008. ISPASS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 169–178.

[14] L. Eeckhout, R. Sundareswara, J. Yi, D. Lilja, and P. Schrater, "Accurate statistical approaches for generating representative workload compositions," in *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*. IEEE, 2005, pp. 56–66.

[15] D. Plonka, A. Gupta, and D. Carder, "Application buffer-cache management for performance: running the world's largest mrtg," in *Proceedings of the 21st conference on Large Installation System Administration Conference*. USENIX Association, 2007, pp. 1–16.

[16] J. Hadamard, *An essay on the psychology of invention in the mathematical field*. Dover Pubns, 1954.

[17] P. Ilmonen, K. Nordhausen, H. Oja, and E. Ollila, "A new performance index for ica: Properties, computation and asymptotic analysis," *Latent Variable Analysis and Signal Separation*, pp. 229–236, 2010.

[18] G. Szekely and M. Rizzo, "Hierarchical clustering via joint between-within distances: extending ward's minimum variance method," *Journal of classification*, vol. 22, no. 2, pp. 151–183, 2005.