Multicore OS benchmarks: we can do better

Ihor Kuz* Zachary Anderson Pravin Shinde† Timothy Roscoe
Systems Group, Department of Computer Science, ETH Zurich

Abstract

Current multicore OS benchmarks do not provide workloads that sufficiently reflect real-world use: they typically run a single application, whereas real workloads consist of multiple concurrent programs. In this paper we show that this lack of mixed workloads leads to benchmarks that do not fully exercise the OS and are therefore inadequate at predicting real-world behavior. This implies that effective multicore OS benchmarks must include mixed workloads, but the main design challenge is choosing an appropriate mix. We present a principled approach which treats benchmark design as an optimization problem. Our solution leads to a workload mix that uses as much of a system's resources as possible, while also selecting applications whose performance is most sensitive to the availability of those resources.

1 Introduction

We argue that benchmarks used in the Operating Systems literature for evaluating new designs and techniques are fundamentally unrealistic: they ignore the common case of running multiple applications (or subsystems) on the same machine. Bluntly, we are measuring the wrong thing. We show, using existing OS benchmarks running concurrently, how traditional benchmarks lead to unrealistic results, and propose composing benchmarks so as to obtain more useful information about how well an OS can multiplex the machine among competing programs.

The purpose of an OS is to allocate and share machine resources between applications in a controlled way. The mismatch between what an OS should do, and which properties we currently measure about it, becomes more serious in the case of multiple, parallel workloads on modern and future multicore processors, where the interaction between competing multithreaded workloads is poorly understood and hard to analyze. As research into OS designs suitable for multicore processors continues apace [2, 4, 10, 12], it is high time we fixed this problem.

Multicore workloads fall into three categories: *High-performance computing* (HPC) workloads are long-running applications that split work into parallel tasks executed across all the system's cores. *Server applications* are characterized by continuous execution of short, independent jobs in response to incoming requests. Scalability here is often a matter of executing many jobs concurrently. Finally, *dynamic* workloads, in desktop and other interactive systems, run a changing mix of applications concurrently. Currently we see interactive applications competing with background applications such as security scanners, indexers, and backup systems. In the future we can expect a much broader range of concurrent activity as so-called mining, recognition, and synthesis (RMS) applications become more prevalent [1].

Existing benchmarks focus on static HPC and server scenarios, using single applications, and neglect the mixed workloads typical of interactive systems. Hence, they fail to exercise or evaluate the performance isolation capabilities of a multicore OS, and are of limited use in validating novel techniques to improve performance and scalability of an OS in non-HPC or server scenarios.

We claim a good multicore OS benchmark suite must provide a mixed workload to be useful in analyzing how an OS performs what is, after all, its main purpose.

In the next section we review the benchmarking methodology used in recent OS research papers, and show that such benchmarks fail to capture important aspects of how an OS manages resources under a mixed workload. We then introduce a new approach for such benchmarks which provides detailed information about how the OS deals with a mixed workload, and describe some initial work on how to interpret the results.

*Also at NICTA and the University of New South Wales, Sydney. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

†Funded in part by a Microsoft Research PhD Fellowship.

OS	m + μ	S+H+D	Scale	Perf	Mixed
Tornado [8]	0 + 8	0 + 0 + 0	8	4	0
HeliOS [10]	2 + 4	1 + 0 + 1	0	6	1
Corey [4]	2 + 4	1 + 1 + 0	5	6	0
fos [12, 13]	1 + 6	1 + 0 + 0	0	7	0
Barrelfish [2]	3 + 3	1 + 2 + 0	3	6	0
Linux [5]	7 + 0	4 + 2 + 1	7	0	0

Table 1: Some recent multicore OS research benchmarks

2 The current way

Multicore OS benchmarks principally evaluate how application performance over the OS scales with the number of available cores. The goal is a workload that accurately reflects expected application behavior and exposes both the explicit and implicit effects of the OS.

We survey benchmarks used in recent publications on multicore OSes, and show how modifying one such benchmark to include mixed workloads exposes scalability issues missed by the original. We are certainly not the first to critique OS benchmarking [9,11], but our concerns here are orthogonal and focus on multicore issues.

2.1 Recent multicore OS benchmarking

OS research custom is to use real-world benchmarks to evaluate overall OS performance and micro-benchmarks to stress particular subsystems for further analysis.

Table 1 shows the number and types of benchmarks used in recent multicore OS publications. Column (m + μ) breaks down the benchmarks into macro- and micro-benchmarks. Macro-benchmarks are further classified as Server, HPC, or Desktop workload respectively (S + H + D). *Scale* shows how many benchmarks measured the system's scalability and *Perf* counts those comparing OS performance to existing OS (typically Linux); some benchmarks are counted in both classes. *Mixed* shows benchmarks run in combination with other applications.

Most papers use a few macro-benchmarks and verify their results with further micro-benchmarks, with a clear bias towards server workloads. Desktop and mixed workloads seem mostly ignored by the research community; the only mixed workload used is by HeliOS to measure isolation between two desktop applications.

Mixed workloads do appear in research that does not directly evaluate OS scalability. For example, in Frachtenberg and Etsion's study of OS mis-scheduling [7], mixed workloads are used as cases where existing schedulers perform poorly. This work uses synthetic load generators and does not aim at realistic workload mixing.

The few mixed-workload that are used are generally chosen based on intuition of what is expected to run on a real system, but these choices do not necessarily lead to mixes that best exercise the OS.

Overall, despite the fact that a multicore OS should provide isolation between running applications, most research does not use benchmarks which evaluate this.

workload 1	w1 cores	workload 2	w2 cores
psearchy	1,2,4,6,8	gmake	6
postgres	1,2,4,6,8	gmake	6
gmake	1,2,4,6,8	gmake	6

Table 2: Configurations of MOSBENCH workloads used

2.2 Case Study: MOSBENCH

To provide a concrete example of why a mixed workload is necessary, we modified the public version of the MOSBENCH [5] suite to run multiple instances at once, and compared the result of running a mixed workload with a single workload. MOSBENCH is a benchmark suite for multicore OSes that includes a wide variety of applications, but only runs one program at a time.

We modified the MOSBENCH harness to start and monitor two workloads at once. MOSBENCH divides a workload into a startup stage, a waiting and collecting stage, and a stopping stage. We ensured that both workloads would run through the stages in synchrony (i.e., both would execute the start stage in parallel, then the wait stage in parallel, and then the stop stage). Throughout the runs, we pinned the workloads to a disjoint set of cores, to reduce interference due to contention for cores. This is not strictly necessary (part of a multicore OS's job is to schedule applications on cores) but it simplifies interpreting the results. We also added a dummy workload that performs no work, to compare the results of a mixed workload to a single workload. We used a 16 core, 4 socket AMD Shanghai machine with 16GB RAM running Linux 2.6.32.

We present a principled approach to workload selection in the next section of this paper, but for this experiment we tried a number of arbitrary combinations of programs from the MOSBENCH suite, and we present a subset of the results in Table 2. In all experiments, workload 2 uses a fixed 6 cores, while we vary workload 1 from its minimum to maximum core count. For each such two-load configuration we also ran with workload 2 replaced by the dummy workload, providing both "mixed" and "non-mixed" results.

Figure 1 shows the slowdown of the mixed workload relative to the corresponding non-mixed run (calculated as $(nonmixed - mixed)/nonmixed$, where *nonmixed* and *mixed* denote jobs per second). For some workloads there is little or no slowdown, but for others resource contention significantly impacts performance. Note that

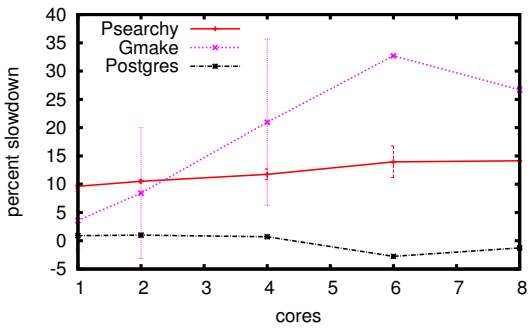

Figure 1: Slowdown for mixed MOSBENCH workloads

while the differences may be modest, they point to an isolation problem that none of the individual workloads uncovered, and one that would unlikely be uncovered by a different single-application load.

In summary, both performance isolation and scalability are affected by mixed workloads. This means that single-workload benchmarks alone are unlikely to provide sufficient insight into the working of the OS. Next, we determine what kinds of workload mixes yield the most information when used to evaluate a multicore OS.

3 A better way

We have argued and shown evidence that a good multicore OS benchmark should provide a mixed workload. The problem, however, is that it is not directly obvious what kind of mix should be used, since as we saw previously, not all workload combinations provide interesting results. The key questions that must be answered when choosing a workload mix include:

- Which applications to choose?
- Which application workloads and configurations to choose?
- Which combinations of applications to run?

Here we discuss an approach to answering these questions. Our work is inspired by work on the DaCapo benchmark suite [3], and the vector-based approach to benchmarking developed by Seltzer et al. [11]. Note that, while our approach may seem complex at first, much of it can be automated, greatly simplifying its application.

Our goal is to design mixed workloads that can reveal information about the scalability and performance isolation provided by an OS. Since such information is gained by pushing the OS to its limits, an effective mix should use as much of a system's resources as possible, and devote those resources to applications whose performance is sensitive to their allocation. In this way, any effect of the OS on those resources will be highlighted by the benchmark, making it easier to trace anomalous application performance back to the OS, or to interactions among OS subsystems.

Because there is no single metric of performance when multiple applications are run concurrently, evaluating the results of a mixed workload is also a problem. Therefore, we must incorporate into our approach application-specific measures of goodness, which we use both to evaluate benchmark results, and to guide the choice of a workload mix itself.

3.1 Optimal mix selection

In our approach, we solve an optimization problem where the constraints derive from the resources consumed by benchmark applications when run alone, along with the sensitivity of their performance to changes in resource availability. We explain how we derive the constraints, and how we use the solutions to compose mixed workloads. We also discuss how to evaluate scalability, performance isolation, and performance degradation in the face of resource overcommitment. Finally, we identify conditions under which the technique is valid.

To show why this is a plausible approach, consider a hypothetical mixed workload composed of typical desktop applications: a game, a web browser, and an anti-virus scanner – a common desktop scenario. Each application accepts many possible inputs, but for our approach we need only consider the set of inputs for which the proportion of system resources used varies as much as possible. We can also force the resources used by a benchmark to vary by placing external limits on an application. We assume that a suitable range of inputs and constraints is supplied by the benchmark designer.

Furthermore, we assume that the benchmark designer provides a way to score the results of a run according to some *goodness function*. For a game this might be a combination of graphics fidelity and frame rate. For a browser, it might be a function of the average page load latency, and for an anti-virus scanner, a function of the number of files scanned in some fixed time period.

With a variety of inputs and a function for scoring the results for each of the benchmarks, we derive the constraints for our optimization problem in two steps. First, we run the benchmark applications alone on all the provided inputs, measuring resource usage, and scoring the results with the goodness functions. Then, for each application we perform a sensitivity analysis to determine which resources were important for performance.

For example, suppose that our example benchmarks are provided with inputs that result in the resource consumption and performance as indicated in Table 3 (this is hypothetical data and not based on measured results).

In this table, the rows give the proportion of a resource

mix	CPU	cache	mem	disk	netwk	score
game1	0.25	0.25	0.25	0.1	0.1	0.25
...						
gameN	1.0	1.0	0.5	0.5	0.5	0.75
webb1	0.25	0.25	0.1	0.0	0.5	0.2
...						
webbN	1.0	1.0	0.75	0.0	0.75	0.6
antiv1	0.1	0.1	0.1	0.6	0.0	0.6
...						
antivN	0.1	0.1	0.1	0.8	0.0	0.8

Table 3: Hypothetical resource usage and performance for our benchmark applications.

bmark	CPU	cache	mem	disk	netwk
game	0.8	0.8	0.6	0.4	0.1
webb	0.8	0.7	0.5	0.1	0.5
antiv	0.2	0.5	0.4	0.8	0.0

Table 4: Example results of a sensitivity analysis.

used by a benchmark on one of N different inputs. For example, the *mem* entry for game1 is 0.25, indicating that the game uses a quarter of the system's memory with input 1. The *score* column of the table gives the application specific goodness score, which is calculated for each of the runs. When the game uses 0.25 of the CPU, 0.25 of the cache, 0.25 of main memory, 0.1 of the disk, and 0.1 of the network, it achieves an goodness score of 0.25.

Using this data we can now perform a sensitivity analysis for each of the benchmark applications. The results of the analysis are a sensitivity score for each resource that show, on a scale of 0 to 1, how sensitive an application's performance is to changes in each resource. Example results of a sensitivity analysis are given in Table 4. This table shows hypothetical sensitivities to resource allocations of our example applications. For example, the CPU (at 0.8) is more important for the game's performance than the network (at 0.1).

We now have all the data necessary to compose the optimization constraints. We phrase the optimization problem as an integer linear program. The solution to the optimization problem tells us which benchmark applications running on which inputs should compose the mixed workload. Let x_i be the integer variable for the i'th benchmark/input pair. The solution to the optimization problem will be an assignment of the x_i's indicating how many of each benchmark/input pair should be run as part of the mixed workload.

For each pair, we know the resource usage. Let r_{ij} be the proportion of the j'th resource used by benchmark/input pair i. We also know the sensitivity of each benchmark to changes in resources. Let σ_{ij} be the sensitivity of the benchmark in benchmark/resource pair i to changes in resource j. The problem is as follows:

$$\text{maximize} \sum_j \sum_i x_i r_{ij} \sigma_{ij} \quad (1)$$

$$\text{subject to } \forall\ j\ . \sum_i x_i r_{ij} \leq 1 \quad (2)$$

Intuitively, what this means is that, without overcommitting the system, solutions will devote as many system resources as possible to benchmark applications that are sensitive to their allocation. In (1) $r_{ij}\sigma_{ij}$ is a heuristic that can be thought of as the sensitivity of a benchmark to a resource, written in terms of the amount of a resource that the benchmark productively uses. It is large if a benchmark is sensitive to and uses a lot of a resource, moderate if it is sensitive to the small amount it uses or is not sensitive to the large amount it uses, and small if it is neither sensitive to nor needs very much of a resource.

We sum over all of the resources in the maximization condition. If the potential constituent benchmarks are sensitive to each of the system resources, then this maximization condition will result in solutions that use every resource as much as possible. Inspecting the resulting solution will indicate whether or not the set of constituent benchmarks is complete enough.

Finally, using this optimization problem we create a mixed workload that uses only the 6 runs listed explicitly in Table 3. Using this sensitivity data to generate the optimization problem yields the following mixed workload results: {game1, game1, webb1, antivN}. This uses 85% of CPU, 85% of cache, 70% of memory, 100% of disk, and 70% of the network, and includes benchmarks that together are sensitive to all of the resources.

3.2 Interpreting Results

Once we have composed a good mixed workload, we can compare the results of individual benchmarks in the non-mixed and mixed settings. Also, we can examine aggregate results in order to expose OS performance issues.

Identifying Bottlenecks: Ideally, a mixed workload should consume the same resources as the sum of those consumed by each constituent benchmark running alone. Deviations from this ideal may indicate subsystems, or interactions among subsystems, for which the OS is having trouble allocating resources when under load.

Performance isolation: Since we know how well each of these benchmarks performed when running alone on the system, we can compare against the performance when they are run all together. In particular, we can calculate the percent difference between the sum of performance scores of the benchmarks run alone, and run as part of the mix. If the percent difference is smaller, then the OS provides better performance isolation.

Scalability: It is also useful to see how an individual benchmark application scales up when others are running at the same time. To accomplish this, we can use the same optimization problem, with the additional constraint that one of the applications chosen must be the one we care about. If we choose inputs that show scalability when the benchmark is run alone, the same inputs should also scale when run as part of a mixed workload.

Resource overcommitment: We can also construct a sequence of mixed workloads in which system resources become increasingly overcommitted. In particular in (2) above, we can replace the requirement that the sum of resources used by all the benchmarks is less than one, with a more general constraint. That is, instead of using 1 as the upper bound of resource usage, we can use other values, even different values for different resources.

Validity of this approach: We also propose a test for determining whether or not this technique will yield consistent, meaningful results. Given a sufficiently large set of benchmark/input pairs, the optimization problems we described above will have several solutions with similar, near-optimal values of the objective function. If our approach is valid, then these solutions will give similar results. In particular, we can perform the performance isolation test for each mix, and obtain a set of percent differences in performance scores. If the variance of this set is small, then we can have confidence in our approach.

Discussion: If the variance in performance differences across mixes is small, then we will have also shown that, so long as a mix is near-optimal, its precise composition is not important: our approach has the potential to obviate the need for "standard" workload mixes, which may be biased toward particular architectures or systems. In the future we wish to show that this technique can tailor mixed workloads for particular systems in such a way that we can both obtain useful diagnostic results for a single system while comparing results across systems.

We can mitigate the complexity of this approach using a tool we are presently building that automates the entire process. Additionally, we can rely on previous work in IO benchmarking, e.g. the work on *self-scaling* workloads [6], to guide our interpretation of the results of sampling a large parameter space.

4 Conclusion

We claim that current benchmarks for multicore OSes do not reflect a realistic workload. In particular, they neglect mixed workloads consisting of several applications running concurrently. However, the difficulty with designing mixed workload benchmarks is in choosing an appropriate mix. We propose a principled approach to designing good mixes based on treating it as an optimization problem. The key advantages are that we can target specific resources of interest and gain a better understanding of how the mix is expected to behave.

In the future, we intend to further develop and evaluate our approach. Choosing a good mix is, however, only part of the problem, and we will address other problems, such as portability of benchmarks, burstiness, and dynamic workloads, as well. It is our intention to work together with others from the OS community to further develop this work, in particular to develop a framework for producing multicore OS benchmark suites, and to produce a standard suite that can be used for further OS research.

Acknowledgments

We thank Jan Rellermeyer, Tim Harris, and Simon Peter for their contributions.

References

[1] Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., Patterson, D. A., Plishker, W. L., Shalf, J., Williams, S. W., and Yelick, K. A. The landscape of parallel computing research: A view from Berkeley. Tech. Rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[2] Baumann, A., Barham, P., Dagand, P., and T. The Multikernel: A new OS architecture for scalable multicore systems. In *SOSP'09*.

[3] Blackburn, S. M., Garner, R., Hoffman, C., Khan, A. M., McKinley, K. S., Bentzur, R., Diwan, A., Feinberg, D., et al. The DaCapo benchmarks: Java benchmarking development and analysis. In *OOPSLA '06*.

[4] Boyd-Wickizer, S., Chen, H., Chen, R., Mao, Y., Kaashoek, M. F., Morris, R., Pesterev, A., Stein, L., Wu, M., hua Dai, Y., Zhang, Y., and Zhang, Z. Corey: An operating system for many cores. In *OSDI'08*.

[5] Boyd-wickizer, S., Clements, A. T., Mao, Y., Pesterev, A., Kaashoek, M. F., Morris, R., and Zeldovich, N. An analysis of Linux scalability to many cores. In *OSDI'10*.

[6] Chen, P. M., and Patterson, D. A. A new approach to I/O performance evaluation–self-scaling I/O benchmarks, predicted I/O performance. In *SIGMETRICS'93*.

[7] Frachtenberg, E., and Etsion, Y. Hardware parallelism: Are operating systems ready?(case studies in mis-scheduling). In *Workshop on the Interaction between Operating System and Computer Architecture* (June 2006).

[8] Gamsa, B., Krieger, O., Appavoo, J., and Stumm, M. Tornado: Maximizing locality and concurrency in a shared memory multiprocessor operating system. In *OSDI'99*.

[9] Mogul, J. Brittle metrics in operating systems research. In *HotOS-VII* (Mar. 1999).

[10] Nightingale, E. B., Hodson, O., McIlroy, R., Hawblitzel, C., and Hunt, G. Helios: Heterogeneous multiprocessing with satellite kernels. In *SOSP '09*.

[11] Seltzer, M., Krinsky, D., and Smith, K. The case for application-specific benchmarking. In *HotOS-VII* (Mar. 1999).

[12] Wentzlaff, D., and Agarwal, A. Factored operating systems (fos): The case for a scalable operating system for multicores. *ACM SIGOPS Operating Systems Review 43*, 2 (Apr. 2009).

[13] Wentzlaff, D., III, C. G., and Beckmann, N. An operating system for multicore and clouds: Mechanisms and implementation. In *ACM Symposium on Cloud Computing* (June 2010).