How can we Teach Workload Modeling in CS Systems Classes?

Cristina L. Abad
Escuela Superior Politécnica del Litoral (ESPOL)
Guayaquil, Ecuador
cabadr@espol.edu.ec

ABSTRACT

In Computer Science curriculum guidelines, topics related to Performance Engineering have typically been listed as small, elective components, if at all. Even less has been said about how and when to teach workload modeling. In this paper, I discuss how Systems courses are a good place to include this topic, including suggestions on how to do so that are rooted in personal experience, existing literature and examples from course programs found online. The ideas presented in this paper were first presented as an invited talk at the TeaPACS 2024 workshop, and the paper was written post-workshop, so the feedback and discussions of the TeaPACS speakers and attendees have been incorporated into it.

1. INTRODUCTION

Workload modeling is crucial in how we design and evaluate systems, as incorrect assumptions about the workload can lead to sub-optimal designs and evaluations [15]. Understanding the characteristics of workloads allows for more accurate performance predictions, better resource allocation, and ultimately, more efficient and effective systems. Thus, teaching students to use representative workloads[1] is essential for them to develop reliable systems that perform well under actual operating conditions.

We must teach Computer Science (CS) students the significance of workloads in system design and evaluation; but this is not a trivial task. Students must learn which features of the workload to model and how to do so effectively. They need to understand whether to model features independently or in combination, and when to use full real workloads (traces) or synthetic ones. This requires a deep understanding of both the system being modeled and the nature of the workloads it will encounter. While many of these things can be learnt post-graduation, any on-the-job learning can be hindered if the student has no previous understanding of the impact of workloads on software, and of the complexities involved in workload characterization and modeling.

We argue that teaching about workloads is best done in Systems courses, where students already learn about system design and performance. Integrating workload modeling into these courses can provide students with practical, hands-on experience and a deeper understanding of how systems operate under different conditions.

Some questions that I address in this paper are: Do current CS curriculum guidelines reflect the importance of the topic of workloads? (Section 2); Where in the CS curriculum can we teach workload modeling? (Section 3); What are the benefits of teaching workload modeling in CS systems courses? (Section 4); How can we teach workload modeling in CS systems courses? (Section 5). For this analysis, I draw on my own experience having taught different systems classes at some point in my career (Computer Communications, Computer Networking, Operating Systems, Distributed Systems, Cloud Computing, Advanced Operating Systems, Big Data Architectures), and complement this with information from CS education papers published at past "International Workshop on Education and Practice of Performance Engineering (WEPPE)" and "Teaching Performance and Analysis of Computer Systems (TeaPACS)" workshops as well as some examples taken from the websites of CS courses in the world.

2. DO CS CURRICULUM GUIDELINES REFLECT THE IMPORTANCE OF THE TOPIC OF WORKLOADS?

The most important Computer Science curriculum guidelines are the ones periodically updated by the Joint Task Force on Computing Curricula of the ACM and IEEE Computer Society, with the Association for the Advancement of Artificial Intelligence (AAAI) joining in for the 2023 version. Their recently released CS2023 [13] mentions *workloads* in several knowledge units within the Architecture and Organization (AR) and Systems Fundamentals (SF) knowledge areas (area and subtopic indicated in parenthesis):

1. AI algorithms for *workload* analysis (AR-Heterogeneity: Heterogeneous Architectures).

2. Energy footprint of data centers at various *workloads* (AR-SEP: Sustainability Issues).

3. *Workloads* and representative benchmarks, and methods of collecting and analyzing performance figures of merit (SF-Evaluation: Performance Evaluation).

4. Understanding layered systems, *workloads*, and platforms, their implications for performance, and the challenges they represent for evaluation (SF-Evaluation: Performance Evaluation).

[1]In this paper, I use the term *workloads* to refer to the collective aspects of workload characterization, modeling, and synthetic generation.

This is a significant improvement from the CC2020 guidelines [9], which did not include the term *workload* in the report. This was a result of the adoption of a competency-based model instead of the body of knowledge approach of the CS2013 [11] and CS2023 versions. While focusing on competencies has several benefits, the CC2020 guidelines failed to include many important topics, as they were presumably implicit in achieving the recommended competencies. For example, for the case of workloads, one would expect them to be discussed when preparing students to achieve competencies like "[m]easure the performance of two application instances running on separate virtual machines at a local engineering company and determine the effect of performance isolation."

Another important curriculum effort is that of the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing (PDC), which has been meeting periodically since 2010. The current version of their guidelines, version 2.0-beta [10], mentions *workloads* in two advanced (not core) learning outcomes, both within the Performance metrics topic (within the memory and network bandwidth subtopics):

1. Be able to explain significance of memory bandwidth with respect to multi-core access, and different contending *workloads*, and challenge of measuring.

2. Know how network bandwidth is specified and explain the limitations of the metric for predicting performance, given different *workloads* that communicate with different contending patterns.

From this analysis, we can observe that existing curriculum guidelines for Computer Science do not mention *workload modeling* as a core topic, though the importance of understanding how *workloads* affect computer systems is included in some guidelines to a greater or lesser degree. In all cases, the topic is addressed in the Systems knowledge area (or the very-related architecture and networking areas).

3. WHERE IN THE CS CURRICULUM CAN WE TEACH WORKLOAD MODELING?

In theory, we could have **specialized standalone courses** dedicated to workload characterization and modeling. However, this approach is unlikely to be adopted, as such a course would be too specialized for most undergraduate and graduate programs.

The subject could also be taught as part of an **applied statistics or probability course**. This approach would be (in the opinion of this author) quite interesting, but unlikely to be adopted given that there are many other more traditional applications that can be considered in those courses.

When **software engineering courses** focus on testing, the issue of benchmarks and the workloads that come with them are usually discussed in class. Another natural fit for the topic of workloads are **computer simulation courses**. However, both software testing and computer simulation courses are electives and most students don't take them.

A class of (elective) courses that has gained some popularity with the advent of the DevOps philosophy are **monitoring courses**. In a sense, monitoring is the basis of workload characterization [4]. Furthermore, when monitoring, the system must be under an artificial workload (active monitoring) or real workload (passive monitoring), so such a class could discuss not only how to characterize a workload, but how to model it and synthetically generate it. As Calzarossa et al. [4] point out, these kinds of courses bridge the gap between theory and practice as monitoring makes it possible to see in action concepts taught in classes (e.g., Internet protocol stack, software systems).

When discussing when to teach workload modeling, two other possibilities deserve a slightly longer discussion: **performance courses** (Section 3.1), as these are the focus of the TeaPACS community, and **systems courses** (Sections 4 and 5), which is the approach argued for in this paper.

3.1 Workload modeling in performance courses

Performance courses can vary significantly, but we can broadly classify them into those focused on performance evaluation (and benchmarking) and those focused on performance modeling.

For courses focused on **performance evaluation (and benchmarking)**, workloads are essential and an important percentage of the course would likely focus on them. For example, Kounev presented a course on Systems Benchmarking in his WEPPE 2021 Keynote [12], and highlighted that benchmarks are characterized by their metrics, *workloads*, and measurement methodology.

A second example of this type of course is "Topics in Performance Evaluation" taught in the past by Prof. Dror Feitelson[2], which featured a course plan that concentrated heavily on workloads, with 50% of the course plan dedicated to this topic and subtopics (see Table 1), and covering workload characterization, modeling and synthetic generation techniques.

Similarly, the course "Performance Evaluation and QoS" taught in the past by Prof. Maria Carla Calzarossa[3] also concentrated heavily on workloads, including machine learning techniques that can be used for workload characterization (see Table 2).

Courses on **performance modeling (PM)**, on the other hand, can be taught without a thorough discussion on workload modeling techniques, and inclusion of such topics or not depends on the instructor's approach. Prof. Vittoria de Nitto Personè [7] analyzed 75 performance modeling courses and classified them into three groups, depending on their instructional focus: General performance modeling, performance modeling for communication systems, and performance modeling for software. Of these three groups, she found that only the first one is somewhat homogeneous in their content, usually focusing on probability, simulation and stochastic processes. Her findings on the topics commonly covered in these courses show that the General PM and PM for Software courses usually include workload characterization in their discussions, though the specifics across all subtopics vary significantly.

An example of a PM course is "Analytical Performance Modeling" taught by Prof. Y.C. Tay, which uses systems research papers to illustrate usefulness of different analytical models [17], including in relation to workloads.

On the other hand, Dr. Giuliano Casale has a more hands-

[2]See: `https://www.cs.huji.ac.il/course/2013/perf/` & `https://www.cs.huji.ac.il/course/2013/perf/plan.html`

[3]See: `https://peg.unipv.it/ImpiantiLS/presentazione.html` & `https://peg.unipv.it/ImpiantiLS/programma.html`

Table 1: Course plan of course: Topics in Performance Evaluation (Prof. Dror Feitelson, The Hebrew University of Jerusalem, 2013).

Topic	Subtopic
Introduction (6 hours)	Motivation and problems
	The issues: techniques, metrics, and workloads
	Using measurements, simulations, and analysis
	Visual data representation
	Different types of graphs
	Avoiding misleading representation
	Introduction to event-driven simulation
Queueing analysis (8 hours)	Queues and queueing networks
	Response time, utilization, and system dynamics
	The M/M/1 queue
	Little's law
	Operational laws and bottleneck analysis
	Open vs. closed systems
	Case studies:
	Analysis of network router with bounded buffer
	Compare two slow processors to one fast one
	Queueing networks
	Mean value analysis
Workloads (14 hours)	Workload analysis and characterization
	Summary statistics such as mean and median
	Creating a variate from a distribution
	Useful distributions
	Parameter estimation techniques and goodness of fit
	Comparing distributions using quantile-quantile plots
	Heavy tails and long tails
	Power laws and the Pareto distribution
	Mass-count disparity and conditional expectation
	Popularity and the Zipf distribution
	Case study: load balancing
	Oblivious balancing
	Balancing based on workload characteristics
	Feedback in workloads
	The daily cycle of activity
	User-based workload modeling
	Self similarity
	The Hurst parameter
Simulation (12 hours)	Event-driven vs. time-driven simulation
	Simulating the system in its steady state
	Evaluating confidence intervals
	Termination conditions and simulation length
	Variance reduction
	Case study: networking evaluation
	The ns-2 simulator and its use
	The PlanetLab infrastructure
	Experimental design and analysis of variation
	Case study: parallel job scheduling
	Scheduling on parallel supercomputers
	Backfilling
	The effect of inaccurate runtime estimates
The End (2 hours)	Summary of exercises and simulations
	Complementary approaches: measurement and experimentation

Table 2: Course plan of course: Performance Evaluation and QoS (Prof. Maria Carla Calzarossa, University of Pavia, 2010); translated from Italian.

Topic list
Introduction Plants and services. Examples. Total Cost of Ownership. Quality of service: dependability, usability, performance. Availability, reliability, security. Service Level Agreement: characteristics and examples. Performance evaluation and capacity planning activities: motivations and phases. Critical issues: bottleneck. Load models and system models.
Monitoring measures Motivations and process. Why, what, where and how to measure. Active and passive measurement tools. Intrusiveness and overhead. Examples.
Workload Definitions. Types of load, levels of detail, parameters. Quantitative and qualitative parameters, measured and derived. Methodological approach. Exploratory analysis: basic statistics, frequency distributions, percentiles, scatter plots and correlations. Web server log analysis. Statistical and dynamic properties of the workload. Parameter scaling. Statistical techniques: clustering. Hierarchical agglomerative algorithms: dendrogram. Hierarchical algorithms: k-means. Principal component analysis. Correspondence analysis. Linear and nonlinear regression methods. Examples of studies that require detailed knowledge of the intensity of the workload.
Chapter 3 Lazowska et al. [14] Fundamental Metrics and Laws: Measured and Derived Quantities. Arrival Rate, Throughput, Utilization, Average Service Time. Utilization Law. Average Number of Requests in the System. Average Residence Time. Little's Law. User-System Interactions: Response Time Law. Visits and Total Service Time.
Chapter 4 Lazowska et al. [14] Service centers. Queuing networks. Input parameters and performance indices.
Chapter 6 Lazowska et al. [14] Solution of open single-class models. Stationarity condition. Performance indices for service center and system. Solution of closed models: Mean Value Analysis.
Simulation Simulation techniques: advantages and disadvantages compared to analytical techniques. Events and simulated time. Types of simulation: trace-driven, time-driven, event-driven. Time advancement mechanisms: simulator clock. Event scheduling mechanisms: event list. State variables. Counters and arrays. Exogenous variable: sampling of empirical and theoretical statistical distributions. Inverse transform method. Random number generators: reproducibility, independence, period. Endogenous variable: unconditional estimates. Confidence intervals. Variance estimation methods: batch means, repeated trials, regeneration.

on approach in his Performance Engineering course [5], which is at the intersection between performance engineering and systems engineering, with a focus on cloud computing systems and measurement. In this course, some workload modeling techniques are introduced (Markov chains) to describe user workload patterns, and in a hands-on exercise, students are asked to instantiate and size VMs and conduct benchmarking and workload characterization experiments.

The approaches discussed in this section (e.g. teaching workload modeling within a performance modeling course) have the disadvantage that those courses are usually electives taken by few students. Particularly, the TeaPACS community has repeatedly discussed the problem that performance modeling courses are decreasing in popularity (e.g., as observed by Dr. Casale [5]), so hoping that students will learn about workload modeling in these unpopular courses is not ideal. For this reason, in the next two sections I discuss how these topics can be weaved into the content of systems courses, which are typically required in a CS program.

4. SHOULD WE TEACH WORKLOAD MODELING IN CS SYSTEMS COURSES?

Systems Fundamentals (SF) is one of the knowledge areas in the CS2023 [13] curriculum guidelines, and is defined as those concepts at the core of the computer systems courses like operating systems, parallel and distributed systems, communications networks, computer architecture and organization and software engineering[4]. In the CS2023, the SF and Architecture areas are the only place where workloads are mentioned. Furthermore, the SF area has two *core* knowledge units related to performance: System Performance and Performance Evaluation. In a way, the guidelines are already pointing to where the topic of workloads is best discussed: Systems courses.

Systems courses are required in most CS programs, so making sure we provide discussions, labs and exercises about workloads (characterization and modeling) in these classes

[4]The author notes that the SF area did not include software engineering in prior ACM/IEEE guidelines.

may be the only way to make sure students are exposed to these concepts. In addition, doing so enables departments to teach performance topics even if they have stopped teaching Performance Modeling courses. The approach also fits well with how performance practitioners (as a community) have evolved in their approach to performance of computer systems [5]. Finally, adding small modules on workloads in the context of a more practical course can help motivate students to take courses focused on PM in the future.

To some extent, this recommendation is a specific case of what Serazzi already pointed out at TeaPACS 2021 [16]: "To facilitate the dissemination of performance evaluation concepts thus increasing the number of students interested in this discipline, another action can be taken at the organizational level. It consists of the integration of performance evaluation concepts, with simple examples, in some popular computer engineering courses. Very few lessons are needed and an application-oriented approach should be adopted."

5. HOW CAN WE TEACH WORKLOAD MODELING IN CS SYSTEMS COURSES?

In this section, we discuss several ways (in **bold**) in which workloads can be weaved into different systems courses.

In classical operating systems classes, algorithms used by the OS like eviction policies and scheduling algorithms are presented with **pen-and-paper exercises** (e.g., classical caching eviction problems) in which toy workloads are used to evaluate these algorithms. However, this treatment of workloads is too superficial and does not help students understand workload models (other than very simple ones).

A much better approach would be to **make workloads a "first-class citizen"** of the course (where appropriate); for example, by: (1) adding a unit or subsection on workload modeling, (2) discussing thoroughly when applicable, and (3) including code to test systems using different workloads. For an example, consider the book *Operating Systems: Three Easy Pieces* by Remzi and Andrea Arpaci-Dusseau [2]. In this book, workload assumptions are presented in the first subsection of some chapters; non-toy workloads are discussed in examples; questions about how workloads affect results are included for students to think about; additional code is provided so that students can run simulations with different workloads; and, more advanced workload modeling is left to suggested readings.

When courses include a laboratory component, we can include **experimentation with diverse workloads in one or more guided labs**; these labs can include workload characterization or modeling components. Courses where this approach fit well are Distributed Systems, Cloud Computing, and Networking. For example, Casale [5] presented a lab on cloud auto-scaling at TeaPACS 2023 in which workloads with different intensity are used to test the responsiveness of a cloud auto-scaler.

If the class does not include laboratory components, similar exercises can be assigned to students as **homework**. For example, a homework where students simulate a networked system, where students use specific workload models to simulate system load. This approach is frequently adopted in networking and distributed systems classes; adding external readings to the exercise can help provide more information about the workload model being used and why, or—preferably—a detailed section of the workload models and how to tweak them (if appropriate) can be included in the exercise description[5].

Similar to the homework approach (and perhaps better suited for in-depth coverage of workloads) is making the use of different workload models in the **final project** of a systems class. For an example, I describe an assignment I used in the past when teaching an undergraduate Distributed Systems class, though the same assignment could work for an Operating Systems class (concurrency topic, caching topic). The students were asked to implement a distributed, in-memory key-value store and a component of their evaluation was the performance of their implementation. To evaluate the performance, we used both YCSB [6] and KV-replay, a YCSB fork that we implemented to allow for trace replays [3]. YCSB comes with several predefined workloads, including and independent reference model implementation with Zipfian popularity for the requested objects. KV-replay was used to replay real traces from YouTube (for details on this workload, see our original publication [3]). We gave extra credit to the (correct) implementation with the top performance across several workloads. Anecdotally, we found this approach got many students really interested in performance, with one even opting to pursue it professionally.

In **research paper-based graduate systems courses** where one or two papers are discussed in each session, the instructor can lead the discussion towards the workloads used in the evaluation. This is the approach I use in the Advanced Operating Systems class at ESPOL, where we hold weekly paper discussions. Most sessions include explicit discussions about the workloads used in the assigned reading, with me posing questions like: Where the workloads used adequate and thorough? Was any interesting workload characterization presented? Did the authors use any workload models? What are the limitations of the workload models that were used? The course also has a final research project, and students are expected to use adequate workloads (models or traces) in their evaluations. In addition, I hold a similarly structured weekly non-credit seminar for undergraduate students interested in systems research, which is usually well attended and liked.

For CS programs that include co-ops or similar industry programs, an alternative is to have students **evaluate real systems for clients**. For example, the CC2020 [9] guidelines used a competency-based approach and included the following competency for CS students in the Systems Fundamentals area: "Measure the performance of two application instances running on separate virtual machines at a local engineering company and determine the effect of performance isolation." However, a risk with this approach is that the company could have uninteresting workloads.

In which systems classes should we include discussions about workloads? Ideally, in all (or most) of them. Performance issues and the related trade-offs are at the core of many systems classes; for example, as highlighted in our analysis of performance topics covered in Distributed Systems syllabi [1]. Whenever performance is discussed in systems classes, a discussion on how this performance is affected by different workloads and how this has important implica-

[5]For an example of such an approach in a Distributed Systems class, we refer the reader to a former exercise from CMU's DS class: `https://www.andrew.cmu.edu/course/15-446/applications/labs/proj2/proj2.pdf`

tions in system evaluations is relevant. I quote Serazzi [16] again, as he articulated this quite well at TeaPACS 2021: "To facilitate the dissemination of performance evaluation concepts thus increasing the number of students interested in this discipline, another action can be taken at the organizational level. It consists of the integration of performance evaluation concepts, with simple examples, in some popular computer engineering courses. Very few lessons are needed and an application-oriented approach should be adopted."

6. CONCLUSIONS

While the importance of workloads (characterization, generation, modeling) is obvious to Performance educators, our discussion at TeaPACS highlighted that other CS educators may not be as aware of how crucial it is to teach these topics to CS students. At a minimum, we want students to understand how the workloads used in system evaluations affect the results (garbage-in, garbage-out [8]), and that understanding expected workloads is key for system design and evaluation. One way to ensure these messages get to them, is to include *workloads* in our systems classes. This paper presented some examples of how to do this that we hope other CS educators will find useful. Workloads are typically students' last concern in their projects, so giving them explicit credit for using proper workloads (models or traces) in their evaluations is a way to ensure that they don't fail to do so; extra-credit for the best performant system has worked for me in my classes as competition can spark their interest. Finally—and tying to the TeaPACS discussion on data-driven approaches—workload modeling is inherently data-driven, and as such, the inclusion of this topic can help students get motivated in Performance Engineering, as many of them are already working on acquiring data science skills that they can naturally use for workload modeling.

7. ACKNOWLEDGMENTS

I thank Vittoria De Nitto Personé and Y.C. Tay, organizers of the TeaPACS workshop series, for their commitment in seeking to improve the state of performance education. I also thank all other TeaPACS presenters and attendees, for their insightful discussions that helped shape this paper.

8. REFERENCES

[1] C. Abad, A. Iosup, E. Boza, and E. Ortiz Holguin. An analysis of distributed systems syllabi with a focus on performance-related topics. In *Companion of the ACM/SPEC International Conference on Performance Engineering (ICPE)*, 2021.

[2] R. Arpaci-Dusseau and A. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 1.10 edition, November 2023.

[3] E. Boza, C. San-Lucas, C. Abad, and J. Viteri. Benchmarking key-value stores via trace replay. In *IEEE International Conference on Cloud Engineering (IC2E)*, 2017.

[4] M. C. Calzarossa, L. Massari, and D. Tessera. Performance monitoring guidelines. In *Companion of the ACM/SPEC International Conference on Performance Engineering (ICPE)*, 2021.

[5] G. Casale. Performance evaluation teaching in the age of cloud computing. *SIGMETRICS Performormance Evaluation Review*, 51(2), oct 2023.

[6] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *1st ACM symposium on Cloud Computing (SoCC)*, 2010.

[7] V. de Nitto Personè. Teaching performance modeling in the era of millennials, 2020. arXiv:2001.08949v1 [cs.CY].

[8] D. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, 2015.

[9] C. T. Force. *Computing Curricula 2020: Paradigms for Global Computing Education*. ACM, 2020.

[10] N.-T. C. W. Group. NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing : Core topics for undergraduates. Technical report, Center for Parallel and Distributed Computing Curriculum Development and Educational Resources (CDER), 2020. Version 2.0-beta.

[11] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, 2013.

[12] S. Kounev. A new course on systems benchmarking - for scientists and engineers. In *Companion of the ACM/SPEC International Conference on Performance Engineering (ICPE)*, 2021.

[13] A. Kumar, R. Raj, S. Aly, M. Anderson, B. Becker, R. Blumenthal, E. Eaton, S. Epstein, M. Goldweber, P. Jalote, D. Lea, M. Oudshoorn, M. Pias, S. Reiser, C. Servin, R. Simha, T. Winters, and Q. Xiang. *Computer Science Curricula 2023*. ACM, 2024.

[14] E. Lazowska, J. Zahorjan, S. Graham, and K. Sevcik. *Quantitative system performance: Computer system analysis using queueing network models*. Prentice-Hall, Inc., 1984.

[15] S. Mühlbauer, F. Sattler, C. Kaltenecker, J. Dorn, S. Apel, and N. Siegmund. Analyzing the impact of workloads on modeling the performance of configurable software systems. In *International Conference on Software Engineering (ICSE)*, 2023.

[16] G. Serazzi. Updating the content of performance analysis textbooks. *SIGMETRICS Performormance Evaluation Review*, 49(2), jun 2021.

[17] Y. Tay. Lessons from teaching analytical performance modeling. In *Companion of the ACM/SPEC International Conference on Performance Engineering (ICPE)*, 2019.