Report on the Second International Workshop on Teaching Performance Analysis of Computer Systems 2023

Vittoria de Nitto Personé
Tor Vergata University of Rome
denitto@ing.uniroma2.it

Y.C. Tay
National University of Singapore
dcstayyc@nus.edu.sg

1. INTRODUCTION

The First International Workshop on Teaching Performance Analysis of Computer Systems (TeaPACS1) in 2021 came to the conclusion among participants that there is a need to continue and regularly update the discussion on teaching strategies, changing the curriculum, reaching out to the systems community, facing the new generation and their new learning habits, etc.

TeaPACS1 was online because of COVID. There was no enthusiasm for a second workshop in 2022, since the pandemic had not abated then. For 2023, most conferences were held offline again, so TeaPACS2 joined the ACM SIGMETRICS Conference in colocating with the Federated Computing Research Conference (FCRC) in Orlando, Florida.

However, FCRC required a timeslot to be set aside for the plenary lecture, so we had to reduce the number of invited talks from 5 to 4. The plethora of concurrent activities, the long-distance travelling required and the need for visas also reduced attendance at TeaPACS2.

Even so, there was a lively and productive discussion at the Workshop, and this report hopes to capture that for those who missed the event.

2. TALKS

The first invited talk was given by Giuliano Casale (Imperial College London), who described how he had shifted the focus of his teaching from theory to hands-on exercises. Cloud computing has removed many classical aspects of capacity planning, but does not nullify the need for performance analysis. He gave examples of configuration optimization (hundreds of options), resource management (sizing, deployment), autoscaling (e.g. overheads and bottlenecks) and workflow scheduling (e.g. critical paths and preemption). He noted that cloud engineering is richer than classical settings, and engaging to students.

The next talk by Diwakar Krishnamurthy (University of Calgary) first describes the pushback from students when teaching a mandatory course on software performance. Part of their resistance comes from engineering accreditation requirements that limit the number of coding and data science courses offered to them, as well as students' general discomfort with mathematics. His strategies for overcoming this include case studies (e.g. web server crash), active learning (e.g. brainstorming realistic problems), prioritizing application over theory (e.g. case description), and linking lab exercises with lectures (e.g. a 13-week project on using an analytical model for sanity checks and what-if scenarios).

Mohammad Hajiesmaili (University of Massachusetts Amherst) gave the third talk, which revisits classical performance analysis of algorithms. Competitive analysis of online algorithms, for example, assumes the worst case, with no knowledge of the future. There is now a growing body of work on what performance improvements are possible when predictions are given by machine learning, say. There is then a tradeoff between being consistent (with the offline optimum) and being robust (despite bad predictions). Such Pareto optimality applies to performance analysis of computer systems too, where there is a tradeoff between efficiency and sustainability (e.g. carbon footprint).

Ziv Scully (Cornell University) proposed two theses: (Thesis1) performance modeling requires advanced mathematics and (Thesis2) we can teach that mathematics in an accessible way. He sees Thesis1 in the ubiquitous heavy tail, the complex structures of jobs (e.g. SQL stages), the metastable equilibria, and the scheduling practicalities. For Thesis2, he suggests simplifying the core foundations, emphasizing very flexible tools, and using rules that work most of the time.

As with TeaPACS1, the highlight of the Workshop was in the two discussions, where speakers and audience contributed equally. The following is a record of the discussions.

3. DISCUSSION1: WHAT TO TEACH

Vittoria de Nitto Personé (Tor Vergata University of Rome) started by asking how teachers could adapt to the expectations of a new generation of students, and whether to focus on what students want. Much of the discussion was then about the tension and tradeoff between students' expectations and teachers' goals.

Bruce Hoppe (OpenCilk) recalled his experience teaching Web Science, whereas what the students wanted was to build websites, and get academic credit for it.

Ziv noted that while students take away from some courses a baby artifact (e.g. web site), they do not get such an artifact with a course on queueing theory.

Giuliano agreed that teachers should select course content that has academic depth, but he is open to compromise, ready to sacrifice theory for relevance outside the ivory tower.

Diwakar pointed out that we should not be surprised by crippling crashes (Twitter 2013, Amazon 2018, etc.) when companies hire undergraduates with no training in performance analysis. He does not consider theory and practice as mutually exclusive — we can link academic exercises to real-world relevance. For example, the use of queueing theory can reduce data centre bloat, which the current generation of students can relate to.

Echoing Ziv's talk, Mohammad observed that there are concepts of interest to both theory and practice, rigorous foundational old-school tools that are relevant for the design of data centres, say. One difficulty lies in the diversity of students, where a course on networks may have students with little mathematics training (e.g. psychology majors).

Evgenia Smirni (College of William and Mary) observed that an old-style performance evaluation course that was an easy class when she first taught it years ago is now considered a "rigorous" course by students (even after cutting corners).

Mor Harchol-Balter (CMU) taught a course (with no mathematics) on system performance for engineers in the industry and found that they don't understand simple concepts — arrival rate vs inter-arrival time, throughput, response time, utilization, etc. (see her TeaPACS1 talk). She had students bring in their projects and found that it took considerable effort to just clarify the concept of "job size". The student for one project was surprised that an increase in cache hits can degrade performance, but that would be obvious if one were trained to think in terms of overheads and queueing.

In Diwakar's experience, even operational laws and MVA (mean value analysis) blew them away.

Vittoria asked if we still have to teach modeling, whether you need to update queueing analysis. Would it be enough for students to learn probability and statistics, and the use of tools?

Giuliano put himself in the place of the students, where they see a lot as data-driven. The industry is comfortable with machine learning (ML), so perhaps we should settle on their ability to pick ML models, rather than building models with assumptions that fail in real life.

Bruce noted that, in the case of OpenCilk, black box models would not work, as the interest lies in understanding how a number of cores and shared memory size, etc., affect performance.

Diwakar quoted the aphorism, "All models are wrong, but some are useful"; training in performance modeling and analysis helps one deduce from measurements whether a system is stressed, and whether a design can scale.

Ziv agreed that the principal value of a model lies not in analyzing a system exactly, but in clarifying what is going on, what the critical issues are. Crafting a model requires an unambiguous specification of the system, thus revealing what we know and what we don't.

Evgenia considers performance modeling as fundamental.

Vittoria then asked: how should a performance modeling and analysis course relate to currently popular courses?

Giuliano suggested spreading the material across probability and simulation courses; it would be more effective, as the students would immediately see its usefulness, and this might motivate those with weak mathematical training.

Bruce noted, in relation to OpenCilk, that some courses on parallel programming are evolving to include performance engineering.

Wrapping up the discussion, Vittoria noted that while we might try to cater to the students' interests, they don't know what they need to know. Ziv and Evgenia agreed: "They don't know what they don't know."

4. DISCUSSION2: HOW TO TEACH

A new attendee, Qian Xie (Cornell University), joined this discussion.

Vittoria asked: What are the best practices? What worked and what did not? What innovation can we bring to teaching performance analysis?

Ziv gave an example where GJ Sussman (MIT) taught classical mechanics with no theorems, just programming exercises and simulations, where students learned to think about systems.

Kristy Gardner (Amherst College) asked, "How do we introduce performance analysis to undergraduates?"

Giuliano suggested taking the technology of the day — say, MS Azure's serverless functions — to demonstrate queueing and caching, etc. Or take a topic from the syllabus and design a competition that taps into student creativity. (His students commented: "Modeling was hard, but blew our minds!")

Qian suggested presenting some real-world examples and letting the students work through optimizing policies. Bruce cautioned that they may confuse underlying theorems with handwaving, and Qian agreed that they may be unable to tell the difference.

Vittoria then described how, in her 90-hour course for graduate students, she requires them to propose modeling projects themselves. The students' proposals included transport, entertainment, healthcare, vaccination, first aid, criminal justice, as well as the more traditional cases of cloud and distributed systems. Vittoria requires them to demonstrate their modeling ability, collect real data, do statistical analysis, build a simulator and validate their models.

The pros of this approach: Students appreciate having self-proposed case studies and are more motivated and engaged. Moreover, this often allows them to unify concepts seen in different courses and understand the global meaning of the curriculum. The cons: weaker students find it difficult to identify a non-trivial case study; the teacher must acquire the relevant domain knowledge (from the students); each project is different, and this means more time for evaluation.

Mohammad asked if the students were given guidelines for how to choose their projects. Y.C. Tay (National University of Singapore) also requires students to propose their modeling projects and, in his case, he encourages them to model the system that they, or their labmates, are working on (so they can add a modeling section to their paper/thesis, say).

Qian pointed out that undergrads may not know enough to make a good choice (Mohammad: it may be too easy or too difficult), so frequent feedback from the teacher would be necessary.

Giuliano sees value in a combination of simulation and analytical modeling. The cross-validation helps the students see the value and limitations of the models — it can be a revelation to them that equations can actually capture simulation measurements. Vittoria also combines analysis and simulation in parallel teaching. She shows the students how the simulation, in some cases, exactly realizes some analytical laws, e.g. Utilization Law.

Diwakar worries that the projects do not demonstrate what the instructor wants them to learn.

In summary, Qian sees this approach to projects as a tradeoff: If the teacher picks the project, the students may not care much, and just focus on checking off the boxes in the evaluation criteria; but they get to understand the principles and practice advanced mathematics. If students pick their projects, they are more engaged, possibly gaining deeper learning from seeing, understanding and doing.

The discussion ended with some time put aside for Mor to raise the issue of NSF support for courses on performance evaluation and analysis. She invited suggestions on how to obtain NSF funds for such grant proposals.

5. CONCLUSION

Somewhere in the discussions, there was a suggestion to create a repository for teaching materials. For a start, there is now a collection of TeaPACS1 and TeaPACS2 slides and reports at `https://teapacs.github.io/materials.html`. It is hoped that those who visit this site will benefit from the experiences described and the suggestions given by the invited speakers and participants.

Performance evaluation teaching in the age of cloud computing

Giuliano Casale
Department of Computing
Imperial College London
gcasale@ic.ac.uk

ABSTRACT

Cloud computing has been one of the most significant developments in computer science of the last two decades, fostering sharp changes in performance engineering practices across the computing industry and, at the same time, profoundly steering research trends in academia. A distinctive trait of this paradigm is that cloud engineers can programmatically control application performance, raising an expectation for computing graduates who find employment in software and system development to have basic performance engineering skills. This, in my view, calls for a broader and deeper education on software and system performance topics as part of the computing curriculum, while at the same time requiring a rethink of the syllabus of a classic performance evaluation module. This abstract presents my personal experience in doing so, including a discussion on the educational strengths and weaknesses of performance engineering emerging from cloud computing practice.

1. INTRODUCTION

If we wish to reflect on how we should teach performance-related topics in university modules, I believe a good place to start is to ask ourselves what professional profiles can leverage the methods and results developed in performance evaluation and engineering (PE), and whether the way the community teaches the discipline is appropriate in educating students for these roles. Indeed, university education ultimately has a mission to prepare students for their careers, therefore such a reflection is in my view needed to avoid designing modules based on personal preferences alone.

Several professions can surely make good use of the tools, insights, and fundamental laws developed over the years by the performance evaluation community. For example, network traffic engineers can benefit from a deep understanding of queueing theory and point processes. Similarly, data scientists are often confronted with understanding complex stochastic phenomena from data, a challenge that shares similarities with performance measurement and workload characterization. Simulation and modelling tools are widely used by professionals in enterprise management and planning (e.g., logistics). Consultants need to design appropriate workloads to find bottlenecks and determine the performance limits of systems built by their customers. Classic PE topics, such as stochastic modelling, performance measurement, and operational analysis, can therefore still undoubtedly play a role in shaping the minds and strengthening the preparation of our students for all these professions and tasks. Yet, in this paper, I take a different view on the suitability of classic PE teaching topics for educating students who wish to understand performance in the context of software systems. This is an important student group, since software engineering is one of the topics that regularly motivates students to choose computing degrees. In particular, I believe that academics who wish to tailor their courses to prepare students to deal with software performance engineering in the age of cloud computing should consider to:

1. reduce the prominence in PE modules of stochastic modelling topics, part of which may still be taught within other modules, such as probability and statistics courses;

2. refresh the PE module syllabus with novel topics, problems, and examples emerging from cloud computing, striking a balance with classic PE concepts and theory.

3. feature in the PE module a “hands-on” component to foster a better understanding of real-world performance issues that arise in cloud systems and techniques available nowadays to monitor and dynamically address such problems while the system is in operation.

The above recommendations are subjective views that reflect my personal experience in teaching PE for the last 15 years. The paper intends to expose my opinions and suggestions in an informal way. I give in Section 3 an overview of the evolution of PE teaching within the Department of Computing at Imperial College London to illustrate an instance where the above changes were applied in practice and positively impacted student participation.

Concerning recommendation 1), stochastic modelling has often been justified in the context of PE with a need to predict system behavior under unseen, or uncertain, operational conditions. Problems such as capacity planning and design-time software performance analysis have been put forward to generations of students to motivate the need for PE. However, in my view, the whole idea of predicting the behavior of complex software and services in advance of their deployment is less viable and widespread today than it used to be at the time those earlier PE modules were designed. At the heart of the issue lay various technological developments, mostly related to cloud practices, which the paper discusses in Section 2.

Stemming from the above, several classic stochastic tools still taught in long-running PE modules, ranging from Markov chains to queueing theory and Petri nets, appear somewhat less relevant to me for professional practice in nowadays cloud and software engineering practice. Recommendation 2) suggests that other subjects may be used to refresh the PE module syllabus. As explained in Section 4, a number of performance-related problems and topics emerging from cloud computing may be used to replace, fully or partially, the stochastic analysis subjects of a traditional PE module. In many cases, such topics still offer space to integrate classic PE notions into the lectures, allowing the module to strike a balance between new and old PE content. Section 4 also elaborates on my teaching experience with these new topics and their perceived value and shortcomings from an educational standpoint.

Concerning recommendation 3, it is now routine for software and service engineers to control software and system performance programmatically, for example leveraging cloud APIs to scale the compute and network resources available to an application. Moreover, leveraging the diffusion of DevOps practices [1], changes to a cloud system are increasingly often performed after its deployment, shifting the emphasis of quality-of-service engineering from "correct design" to "problem fixing at runtime". This state of play raises, in my view, a need for computing graduates to receive, as part of their education, some basic training on how to address performance problems for a live cloud system while in operation. Configuring cloud auto-scaling may be an example of a basic skill required by many employers. This requires the ability in a student to reason on how the capacity allocated to a software system should be varied dynamically to ensure that the desired level of performance is maintained despite workload fluctuations. Auto-scaling systems vary among different cloud platforms but, at least in their basic implementations, they often require a limited theoretical preparation, making them well suited for self-contained lab exercises. Topics such as distributed tracing and benchmarking are also well-suited to increase student familiarity with these problems. In my view, lab components can therefore enrich PE teaching and help convince students to opt in for such modules through a better alignment with cloud engineering practice.

2. EDUCATING FOR THE PROFESSION

PE is a continuously evolving discipline, allowing us to look back at its trajectory over the years to spot how today's PE differs from the one for which classic PE modules were designed. As mentioned, since the majority of students seek university education to prepare for a career in companies, I believe it is important to look at the application of performance concepts in a practical setting to steer the educational offering proposed in academic PE modules.

One possible way to do so is to look at the notion of performance discussed in practitioner events. The focus of this section is therefore on observations on the evolution of the Computing Measurement Group (CMG) [12], a prominent community of performance practitioners. This community featured at the height of its popularity several international branches, which organized local events and brought together several hundreds of people worldwide with a keen interest in the topic of performance evaluation. The community also ran the popular CMG conference for many decades, which gathered contributions from industrial practitioners and scholars in the PE field. In a typical year, the CMG conference program would include presentations from experts and capacity management teams, for example operating in banks or in ICT consulting companies. Notably, despite its practical scope, the event included presentations from research scholars, demonstrating the presence of a healthy interaction between the research and the practitioner communities.

For a historical perspective on the notion of performance discussed at these events, the reader may look at the survey in [12], which focuses on themes touched upon in the 1970s and 1980s. Taking CMG 2003 as a more recent example, the conference mostly featured papers on mainstream technology (e.g., Windows, DBMS, z/OS mainframes, . . .), capacity management methodologies (planning, ITIL, sizing, measurement, QoS), and stochastic modeling (queueing, simulation, forecasting), demonstrating, at least in the last two topic areas, an overall good alignment with the themes of both PE research and education of those days.

Let us now move forward 20 years to the present day. The same organization now brands itself as a technology community, featuring a more industrial composition. Topic-wise, the conference program now focuses on more recent infrastructure technologies (cloud databases, cloud servers, containers, ...), observability (monitoring, distributed tracing, ...), and AI (automated system configuration, operationalization, ...). Interestingly, aside for monitoring and mainframes, few contributions overlap significantly with the topics of older conference editions. From an educational and research standpoint, it is striking in particular to see a near absence of tools from classic PE theory, such as Markov chains, queueing theory, but also a modest adoption of simulation-driven analysis. Clearly, a dramatic shift in PE practice has occurred over the last two decades. In my opinion, this suggests that PE educators should take notice of this change along the lines of recommendation 2 in the introduction.

The reasons for such a shift in PE practice are surely multiple. It may still be possible to conjecture on what some of these might be so as to inform how PE education should change as a result. To begin with, computer systems are generally much more complex than in the early days of PE, and basic analytical and simulation models tend to perform less accurately in complex systems than data-driven AI models. AI models also take a black-box approach to the problem that broadens their applicability, whereas stochastic networks typically require some knowledge of how the system works, and in this sense they are much more time-consuming to develop, besides requiring skilled professionals.

There may also be several other reasons why classic PE modeling tools are not as widespread as they used to be. For example, if cloud engineers can quickly and cheaply correct performance problems on-the-fly, allocating and removing capacity in a matter of minutes, and thus the cost of resource allocation mistakes is low and easily fixable, reactive methods in use in many cloud systems, albeit not perfect, are sufficient for many companies. Moreover, if monitoring is cheap, ubiquitous, and allows the observation at a high-frequency of live distributed systems, steady-state analysis via a model may be a less appealing way to characterize performance than directly doing performance troubleshooting on the running system. And even in situations where steady-state predictions can be helpful, with applications that are densely layered, built on top of a stack of platform services and software-defined infrastructure (e.g., IaaS, containers, JVMs, serverless, etc), such predictions may become brittle, as the models can only characterize a small slice of the overall system.

Summing up, the technological evolution driven by cloud computing has shifted the attention of PE practitioners in the software domain to observing systems in a live environment and to using AI models to reason on their performance. The cost of applying hand-crafted performance models remains high and companies face skill shortages of staff experts in building such models. As suggested in the recommendations in the introduction, this warrants in my view a major rethink of which PE topics should be taught in academia in the perspective of better preparing students in computing degrees towards careers in software and service engineering.

3. PERSONAL TEACHING EXPERIENCE

In many discussions I have had over the years with colleagues in the PE field, the subject of a perceived change of tastes of the students in recent years has come up many times. This circumstance is one that I have also observed myself in the PE courses taught within the Department of Computing at Imperial College London. The goal of this section is therefore to give an example of how my PE teaching has evolved in response to the challenges described in the previous sections.

In the Department of Computing at Imperial College London taught modules normally involve 28 hours of frontal teaching, including both lectures and tutorials. The latter may be either in-class tutorials or lab sessions. Classes in the last two years of the curriculum, where PE topics are typically taught, include a mix of both undergraduate and Master's students. A short module duration of just 28 hours, which maps to a mere 4 hours a week for 7 weeks, is not uncommon in the UK. It implies that module lecturers need to carefully select their syllabus, as it would be difficult for example to present the content of an entire book in such a short time span. Assessment is typically based on coursework and a written exam.

A *Performance Analysis* module was taught for decades in our department, which students could optionally select in the last year of their studies. The Performance Analysis module was focused on probabilistic modelling, with an emphasis on Markov processes, queueing theory, and numerical exercises. The course ran smoothly for many years and students were generally keen (if not fascinated) to explore the mathematical techniques available for computer system analysis, such as Markov processes, queueing systems, queueing networks, Petri nets, stochastic process algebras, and others. However, the recent generations of computing students showed a decreasing interest in stochastic theory, possibly as a result of the factors outlined in the previous sections and the concurrent rise of AI, which has shifted the interest of those keen on mathematical modelling onto subjects related to machine learning. A decision was taken to end the long-running *Performance Analysis* module in 2014, replacing it with a module with the title but, due to other staffing needs, with half credits (14 hours of frontal teaching) and an entirely new syllabus.

For the new course, which started in 2015, I decided to soften the mathematical density of the module by switching to a mean-value analysis (MVA) based PE teaching, centering the presentation on materials covered in the classic PE book by Lazowska *et al.* [10], refreshed with real-world examples from cloud computing research and industry. A distinctive feature of Lazowska *et al.*'s book is the ability to present essential performance analysis theory without entering either into a probabilistic description of the system or formal proofs, but rather focusing only on operational analysis [3]. However, students kept demonstrating a limited appetite for queueing theory even if this was presented in a mathematically simpler form.

In the years that followed, I then looked to change my PE teaching more fundamentally, significantly reducing the focus on stochastic modelling, and seeking a stronger case for applicability and relevance to professional practice. This resulted in much-improved participation and an overall higher level of engagement from the students. The change consisted mainly of the following modifications. Firstly, a new *Performance Engineering* 28-hours module started in 2016 with the syllabus focused on the intersections between PE with cloud computing, but taking primarily a systems engineering and measurement view. The module wanted to retain some methodological elements of classic PE (e.g., Markov chains to describe user workload patterns). Moreover, this was the first course in our department to teach hands-on cloud computing basics to our students, including a lab-based coursework that leveraged an educational sponsorship from Microsoft Azure. In these lab sessions, the students could learn the basics of instantiating and sizing VMs and conducting benchmarking and workload characterization experiments, followed by coursework centered on measurement and autoscaling. This change introduced a practical side to PE teaching that students really enjoyed. In a handful of years, the module grew from 14 students in 2016 to 96 students in 2020. Throughout this period, a colleague joined me in teaching the module, introducing, among others, topics related to cache performance and monitoring hardware performance counters, further strengthening the exposure of the students to systems performance.

At the same time of the above developments, I decided to spread the more theoretical aspects of PE (e.g., Markov chains, scheduling, point processes) horizontally across other modules I taught. These included modules on probability and statistics (2nd year), scheduling (3rd year), and simulation (3rd year). My experience was that presenting topics such as stochastic processes and queueing theory within theoretically-focused modules that targeted a broader problem space then PE worked better in terms of the student reception. I also noticed that exposing some performance concepts only in a determining setting (e.g., deterministic scheduling) worked better for many students than looking at similar problems through a stochastic lens.

Summarizing, the recommendations provided in Section 1 may be seen as the key takeaways of the evolution of my teaching discussed in this section. Evolving the *Performance Engineering* module format with the inclusion of topics from cloud computing allowed me to address what seemed a progressive decline in student numbers and interest in PE. Several important topics that were traditionally taught in that module, such as Poisson processes and queueing theory, have then been spread horizontally across other modules in the degree, obtaining a warmer reception, possibly also due to a different composition of the student body.

4. TEACHING TOPICS

Leveraging the experience gained in teaching the modules described in the last section, this section presents some recommendations for cloud-related topics that may be included in a modern PE syllabus. My goal, in particular, is to review topics that integrate well with classic PE materials, allowing the lecturer to still teach several classic notions of performance evaluation but in a renewed context.

4.1 Configuration optimization

The topic of configuration optimization deals with elaborating strategies to automatically tune the configuration parameters of a software system so as to maximize some performance measures [7]. For example, in a Big data platform such as Apache Storm, hundreds of configuration parameters need to be assigned. Such parameters can profoundly influence the performance of cloud systems, which commonly rely on these open-source platforms.

As traditional queueing theory models focus on a narrow set of system parameters (e.g., number of service stations, server multiplicities, service rates, . . .), this topic allows the students to develop a broader view of factors that affect performance. The topic can also help them develop data-driven modelling skills by looking at system response surfaces. The topic is also useful to foster reasoning on how to find optimal solutions from such surfaces.

Teaching benefits. In my experience, the configuration optimization topic is generally interesting to students, as it is easily understandable and a useful skill to gain due to its generality. A benefit of its teaching is that it has a tight link to benchmarking, which can be used to gather the data to fit the response surface. This then naturally leads to cover in the module associate topics such as design of experiments, the structure of a modern benchmark (such as the SPEC benchmarks[1]), surrogate modelling, and regression model fitting. Books such as [9, 11, 8] offer excellent source materials to develop lectures on the topic.

Another benefit of the topic is that it is fairly easy to set up coursework or lab-based exercises. For example, students may be asked to optimize a set of on/off options for a system, e.g., enabling and disabling hyper-threading on a CPU while observing the resulting performance changes of a service, or changing the flags passed to a compiler, followed by performance profiling of the compiled executable.

Teaching challenges. On the downside, this topic is rapidly evolving in the state of the art, and thus may require frequent updates to the slides. The analysis of surrogate response surfaces to determine optimal system configuration may also be best conducted if the students have some machine learning background (e.g., Gaussian processes) and a basic understanding of related optimization methods (e.g., Bayesian optimization). The topic may also be somewhat difficult to test in an exam setting, where limited data and mathematical calculations can be expected.

4.2 Cloud deployment

Cloud computing systems are increasingly deployed by means of declarative models of infrastructure resources, as in AWS CloudFormation[2] or OASIS TOSCA models[3]. An orchestrator then takes in input such specifications and deploys the application on the requested resources, instantiating them on-the-fly on the cloud. Yet, the amount of resources requested for an application is a controllable parameter that the application owner needs to decide. This opens interesting questions on how the performance engineer should decide which and how many resources to allocate to the application. This mindset brings into play questions similar to capacity planning, but from a different angle.

Teaching benefits. Analytical models may be justified with this problem as allowing to model performance within a computational optimization program used to reach a resource allocation decision. In its simplest form, to avoid the complications of stochastic models, this may be just a linear program that depends on the utilization levels of the resources, since operational analysis requires for these only simple laws [3]. Where desired, methods to select specific resource types (queue service rates), to respect service-level agreements (e.g., response time distributions), or to account for bare metal contention (e.g., multi-tenancy/multi-class) may also be studied, if the chosen modelling formalism is sufficiently expressive to address these problems.

Teaching challenges. Similarly to the configuration optimization topic, also this topic has the drawback of assuming some familiarity in the students with computational optimization methods, such as mixed-integer linear programming. Describing multiclass systems may require additional time and place additional complexities in explaining how to parameterize the relevant models (e.g., regression-based estimation of service demands). Also in this case, setting up exam questions that deal with computational optimization formulations presents challenges, as the student cannot derive by hand the solution at the exam. Assessment may be therefore lab-based or limited to writing the optimization program formulations without an explicit solution.

4.3 Autoscaling

As mentioned earlier, autoscaling is an essential topic for cloud engineering [13, 6]. The topic can be integrated within a PE module at varying degrees of sophistication (e.g., reactive vs. proactive autoscaling). The topic also requires the lecturer to introduce concepts of system transient and steady-state, since the time for the system to settle after a scaling decision is a parameter that affects the configuration of some autoscaling rules. Autoscaling is also easy to couple with load balancing topics, allowing again to integrate well with materials from classic PE theory.

Teaching benefits. Several students displayed significant excitement for the autoscaling topic in my modules and generated a follow-up demand for thesis supervision. The topic is easily linked with mathematically-rich topics such as forecasting, scheduling, and control theory. Methods from forecasting such as autoregressive and moving-average processes are appropriate for computing students as they require just basic elements of conditional expectations and Gaussian distributions, and the fitting of small models (e.g., AR(1)) requires simple algebraic formulas. Exposure to forecasting is also helpful to students who seek to build a career in software services for the financial industry. Such students are common across university degrees in cities like London.

[1]https://www.spec.org/benchmarks.html

[2]https://aws.amazon.com/cloudformation/

[3]https://www.oasis-open.org/committees/tosca/

Teaching challenges. A first issue with this topic is that if the content is presented using examples or lab exercises based on specific cloud providers, then slides may need yearly updates as autoscaling technologies and their interfaces evolve rapidly. A second issue is that specific elements of the theory are possibly too simple for a third- or fourth-year student (e.g. rule-based autoscaling), also presenting a low level of challenge in assessment. Lastly, a problem is that if the lecturer wishes to integrate elements from control theory upon teaching the topic, for example as in [5], these require theoretical baselines that may not be available to all computing students (e.g., Laplace and Z-transforms).

4.4 Serverless workflows

Serverless computing has been a prominent cloud computing trend for a number of years. Within it, Function-as-a-Service (FaaS) allows users to execute functions remotely in the cloud [4]. By enabling fine-grained autoscaling, FaaS offers a systematic advantage over traditional web services for enabling a scalable execution of scientific and business workflows in the cloud.

Serverless workflows may be used to teach various PE topics. Taking the user perspective, the module can introduce scheduling in the presence of workflows, resorting for example to the large body of literature available on deterministic scheduling [2], possibly in a lighter (albeit less rigorous) form than in algorithm theory courses devoted to the subject.

From the FaaS platform perspective, the PE module may touch upon heuristics, such as bin packing, that the platform may use to consolidate different serverless functions across server machines. The effects of memory constraints on performance are also an interesting topic aligned to serverless. Workflows are also useful to introduce the notion of a deadline, since they are often used to support critical end-of-month financial calculations in many businesses.
Teaching benefits. The topic has excellent appeal to students, who see its importance in the real world and the timeliness with respect to ongoing cloud computing trends. When coupled with scheduling theory concepts, the topic allows the lecturer to develop in the class a basic understanding of important general concepts such as NP-hardness, from which it is easy to pair the module with exercises involving metaheuristics (e.g., stochastic annealing, local search), which in my teaching experience generate a positive response in a computing class. It is also simple to define serverless scheduling problems in the coursework. My taught modules feature a workflow scheduling system for Azure Function that is supplied to the students. The exercises often involve studying some completion time minimization problems by developing a workflow schedule that is executed through this system. This generally has a good reception in the class as it combines both technological and theoretical elements.
Teaching challenges. One limitation of the serverless workflow topic is the limited number of optimal methods for general workflow scheduling [2]. Albeit several heuristics exist, this limits the appeal of the theory that can be developed in class. Moreover, real-world workflows can feature complex synchronization and be trigger-based and data-driven, which may introduce excessive complexity for modelling purposes. Lastly, workflow scheduling problems that are not solvable by brute force require tens of function calls. This may significantly extend the time required to solve coursework exercises by experimental means.

5. CONCLUSION

In conclusion, the advent of cloud computing offers an opportunity to refresh several contents of performance evaluation modules. In the early years of the performance community, there was general agreement on the importance of having a unified discipline centered around topic such as queueing theory. However, many more tools are available nowadays to study the performance of complex computing systems and performance evaluation teaching should evolve as a result. A review discussion has been presented on topics and techniques emerging from cloud engineering practice that may be added to the syllabus of a performance evaluation module, together with reflections on their educational merits and shortcomings.

6. REFERENCES

[1] A. Alnafessah, A. U. Gias, R. Wang, L. Zhu, G. Casale, and A. Filieri. Quality-aware devops research: Where do we stand? *IEEE access*, 9:44476–44489, 2021.

[2] K. R. Baker and D. Trietsch. *Principles of sequencing and scheduling*. John Wiley & Sons, 2013.

[3] P. J. Denning and J. P. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys (CSUR)*, 10(3):225–261, 1978.

[4] S. Eismann, J. Scheuner, E. Van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup. A review of serverless use cases and their characteristics. *arXiv preprint arXiv:2008.11110*, 2020.

[5] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback control of computing systems*. John Wiley & Sons, 2004.

[6] N. R. Herbst, S. Kounev, and R. Reussner. Elasticity in cloud computing: What it is, and what it is not. In *10th international conference on autonomic computing (ICAC 13)*, pages 23–27, 2013.

[7] P. Jamshidi and G. Casale. An uncertainty-aware approach to optimal configuration of stream processing systems. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 39–48. IEEE, 2016.

[8] A. I. Khuri and J. A. Cornell. *Response surfaces: designs and analyses*. Routledge, 2018.

[9] S. Kounev, K.-D. Lange, and J. Von Kistowski. *Systems Benchmarking*. Springer, 2020.

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

[11] D. J. Lilja. *Measuring computer performance: a practitioner's guide*. Cambridge university press, 2005.

[12] T. L. Lo. The evolution of workload management in data processing industry: a survey. In *Proceedings of 1986 ACM Fall joint computer conference*, pages 768–777, 1986.

[13] B. Wilder. *Cloud architecture patterns: using microsoft azure*. " O'Reilly Media, Inc.", 2012.

Teaching Software Performance Evaluation to Undergrads: Lessons Learned and Challenges

Diwakar Krishnamurthy
University of Calgary
dkrishna@ucalgary.ca

ABSTRACT

Recent high-profile performance-related outages and problems in industry clearly establish the importance of imparting performance evaluation skills to students at the undergrad level. Yet, performance engineering is rarely a required course in most software engineering programs around the world. The typical undergrad student is naturally drawn towards coding courses and courses on topics that they think are likely to be in demand in industry, e.g., data science. While sympathetic, curriculum designers often cite student pressures and other factors such as accreditation requirements from engineering bodies to argue against mandating a performance evaluation course. As a long time instructor of a mandatory, undergrad software performance evaluation course, I describe some of my experiences operating in such a climate. Specifically, I outline key strategies I have followed to motivate students and overcome their resistance to the somewhat analytical nature of performance analysis. I also offer my observations on how undergrad curriculums can be tuned to instil a performance-aware mindset into students. Finally, I point out ongoing challenges to stimulate future solutions.

1. INTRODUCTION

Performance considerations are critical in real-world software systems. Recently, there have been many instances of systems failing because performance considerations were not addressed adequately. A well-known example is the failure of the heathcare.gov Web site, which was plagued by many problems including poor performance. For example, there were reports that the Web site could not even handle 500 concurrent users when it went live [1]. More recently, performance problems have affected even Fortune 100 companies that have access to large scale computational resources and state of the art scaling techniques [3].

Since ignoring performance considerations can clearly have adverse real-world implications, it is imperative that organizations have access to engineers with performance evaluation skills. Specifically, organizations need personnel that can design systems that reduce the likelihood of operational problems related to performance. They need engineers who can build effective scaling strategies and who can trouble shoot performance problems on the rare occasions they occur.

Unfortunately, most organizations hire undergrads [5] and most undergrad software engineering curriculums do not prescribe mandatory performance evaluation courses. Performance courses are more common at the graduate level. Some universities may offer a performance course at the undergrad level, but such offerings are typically not mandatory. In effect, we as educators are leaving performance-related training to chance. I argue that this is quite reckless given how critical the topic is!

In the remainder of the paper, I discuss the main challenges in including performance evaluation courses in undergraduate software engineering curriculums. I will then focus on one of these challenges – student buy-in. I will describe some strategies I have observed to be effective in getting students interested in performance evaluation. Finally, I will present a discussion of open pedagogical problems that need to be addressed by the SIGMETRICS community.

2. PERFORMANCE EDUCATION AT UNDERGRAD LEVEL: CHALLENGES

At the outset, the lack of performance evaluation training might seem like an easy problem to fix. All that needs to be done is to create the appropriate course and decree that it should be taken by all students. However, it is a bit more complicated to operationalize this idea for several reasons. One key reason is that past curriculum standardization efforts do not explicitly include performance evaluation as mandatory material in their recommendations. For example, the IEEE/ACM reference curriculum [4] mentions performance adjacent topics such as software quality but does not explicitly emphasize topics such as performance modeling and measurement. This silence has the unintended consequence of making performance evaluation and engineering sound like a niche field thereby complicating their inclusion in undergrad programs.

Another issue that complicates curriculum design is that in some jurisdictions such as Canada engineering degrees, including software engineering degrees, must be accredited by an external body. In the case of Canada, this is the Canadian Engineering Accreditation Board (CEAB). The CEAB, for example, requires engineers to have depth in their own domain but also breadth in subjects such as math, physics, other engineering disciplines (e.g., thermodynamics, and statics), and complementary topics (e.g., role of engineers in society). While the pursuit of breadth and linkages to other engineering disciplines is laudable, in practice, such requirements make it very hard to carve out space for courses that focus on topics such as performance evaluation.

To illustrate this, software engineering students at my university only take one programming course in year 1! Year 2 has a 50-50 split between software (e.g., Java programming) and non-software (e.g., optics and digital circuits) courses. Consequently, students arrive in year 3 with a lot of ground to cover in core software engineering (e.g., software design) and computer science (e.g., operating systems) courses. In the final year, students are required to take a project management course, complete a design project, and choose from a list of elective courses that provide them an opportunity to pursue specific interests.

Thus, a curriculum designer faces the difficult choice of balancing between depth in software engineering and breadth in other topics. In essence, there are only a limited number of slots available for core software engineering and computer science courses and these are typically filled with topics recommended in standard curriculums.

Finally, a crucial issue that needs to be addressed is student buy-in for a course in performance evaluation. I have been an undergrad curriculum administrator for a decade, and I get to speak to students regularly. Most of them are anxious to make up for the time they have spent taking non-software courses. They prefer courses that will land them their dream jobs. Not surprisingly, courses on data science and game development, not performance evaluation, dominate their thinking. When I offered performance evaluation as an optional course for the first time several years ago, student feedback was overall lukewarm. They felt it was a very niche course and that it was ”grad” material. They were somewhat put off by even the modest level math in the course. Somewhat disturbingly, they felt that the material was not relevant to them and that they do not see the course knowledge being useful to them as practicing software engineers.

Much to my satisfaction, a course in performance evaluation was made mandatory in the software engineering program at my university. My focus in the rest of the paper is the key strategies I used to foster student interest while teaching this course. Specifically, these strategies strive to convince students that performance evaluation is a critical topic that is going to be useful to them in industry. They seek to emphasize practical skills that students can use in their industrial practice.

3. COURSE CONTENTS

The course I teach is organized as follows. I start by motivating the importance of performance using several industrial case studies. Then, to coax students to come out of a functional mindset to a performance mindset, I review some of the previous courses they have taken with a performance lens. For example, I consider computer architecture and discuss how things such as cache hierarchy, processor, and memory organization impact performance. I talk about multi-core architectures and the importance of software level parallelism to take advantage of hardware parallelism. I discuss operating system level issues such as concurrency, synchronization, and threading models and how they impact performance.

Next, I switch to describing the software engineering lifecycle and how performance analysis fits there. I introduce different performance evaluation techniques such as analysis, simulation, and measurements and discuss how these techniques differ and complement each other. The rest of the course focuses on two main aspects namely, modeling and measurements. In modeling, I teach operational analysis and product form models. I also introduce some advanced modeling techniques, mainly Mean Value Analysis (MVA) extensions to study queuing for software resources. The final module emphasizes measurement concepts such as workload modeling, load testing, and performance monitoring. I also introduce experiment design and associated statistical analyses.

4. STUDENT ENGAGEMENT STRATEGIES

On reflection, the main strategy I used is to explicitly motivate performance analysis and its importance to a practicing software engineer. A particularly effective technique to obtain student buy-in is to narrate case studies or situations that I have encountered either in industry or in my industry-oriented research projects. One of the case studies I narrate early in the course was one I was personally involved in while I was at HP Labs. This was a study where there was a production Web site operated by a Fortune 100 company. The Web site had horrendous performance problems such as low throughput and high page load times. Initially, the Web site’s engineers tried to throw hardware at the problem. They increased the system’s horizontal scaling but to no avail. While space constraints preclude me from going into the details, at a high level, the problem occurred due to one method that had a very long critical section. When the method was modified to have a shorter critical section, the performance problems disappeared.

I use this case study to drive home several important points to students. First, performance problems are real, and they can have serious economic implications if left unchecked. Second, I emphasize that just conducting functional testing is not sufficient. The offending method in this case passed functional tests but scaled very poorly and this was not caught prior to deployment due to inadequate performance testing. Finally, throwing hardware at the system will not fix many performance problems. In this scenario, the system was suffering from a software bottleneck. This motivates the need to design systems that do not have such insidious performance problems.

The next strategy I used is to complement narration of case studies with active learning to promote student engagement. Specifically, I describe a certain performance problem and ask students to solve it in groups during the classroom. For example, one of these sessions is based on a paper we published on the scalability of a multi-core web server [2]. This exercise occurs during the part of the course where I establish linkages with computer architecture. In this session, I give groups snippets of information such as the architecture of the server we used, the concept of multiple sockets, multiple cores, local memory access, and socket inter-connect bottlenecks. I then tell them that we installed an Apache Web server on the system and noticed that the throughput scaled linearly with cores when only cores from a single socket were used. However, sublinear scaling was noticed when cores from both sockets were allocated to the application. Students are asked to brainstorm the source of the bottleneck and techniques to mitigate it. Students typically ask a lot of follow up questions but are in general able to figure things out, which in this case relates to poor operating system scheduling decisions that trigger a socket interconnect bottleneck. Many groups correctly suggested that running two Web server replicas with each replica pinned to its own dedicated socket might mitigate this problem.

This exercise promotes student engagement with performance evaluation in many ways. For example, it shows that performance evaluation and debugging can be fun since it resembles forensics. Furthermore, it reinforces the real-life importance of performance analysis by highlighting the fact that even production grade software (Apache) can have poor scalability. Finally, it reiterates that throwing hardware (in this case cores) at a system may not be effective in solving all performance problems. Careful design of software components such as operating system schedulers combined with thoughtful application configuration is crucial for fully leveraging hardware level parallelism.

I also consciously prioritized applications over theory given the overall objective of establishing industrial relevance. Specifically, for the modeling part, I decided to cover techniques that are mathematically simple yet have good practical applications. So, operational laws and product form models are the main modeling topics I cover in the course. This is probably not an ideal choice. For example, product form models have obvious limitations, which might limit their applications in many real-world contexts. However, I decided to trade off mathematical sophistication for student buy-in and motivation. In keeping with this strategy, the course focus is always on coming up with or building a proper model for a given scenario rather than the mechanics of solving the model. I allow students to use formulae cheat sheets during exams and solvers for projects. The emphasis is not on solving the model but rather on coming up with the model. I also noticed that the wording of assignment and exam problems can influence student engagement. Problems should be worded to have a practical flavour so that they come across as plausible scenarios that could happen in practice.

The final strategy I emphasize is the building of strong linkages between the lab and lecture components. While I give students the freedom to propose their own performance evaluation project, I introduce activities that force them to apply concepts covered in lecture. For example, consider a project that involves load testing. Students are required to provide a rationale for the synthetic workloads selected. They are directed to use operational laws to ensure that the load test environment has been setup properly, e.g., verify that measurements of the number of concurrent users, response time, and throughput follow Little's law. Students are given exercises that show how operational laws can be used to derive performance model parameters such as resource demands. Students are encouraged to build a model of the system using the derived parameters and explore how the model can be used to answer various "what-if" performance questions. Students find this portion of the course enjoyable due to its hands-on nature. The key idea is to introduce activities within the project that demonstrate how modeling approaches can complement measurement exercises.

5. CONCLUDING REMARKS

While student feedback suggests that the strategies outlined here are effective, there remain several challenges that need to be addressed. For example, in my opinion, the way software engineering programs are typically structured leads to siloed teaching efforts. Specifically, many programs have standalone courses on requirements elicitation, design, architecture, coding, functional testing, and performance analysis with no link to one another. Since all these activities are inter-related, there is a need for more integrated teaching efforts. For example, requirements gathering should include performance requirements. Design should include building models from design to see if the designs are likely to meet performance requirements. Testing courses should also introduce load testing to discuss aspects such as concurrency and scalability.

There are other aspects that need a thoughtful discussion in the SIGMETRICS community. For example, given the prevalence of machine learning based performance models in industry, how should queuing theory be introduced and taught to undergrad students? How much emphasis should be placed on theory given many students are intimidated by the rigorous math involved in classical performance analysis?

6. REFERENCES

[1] C. Brooks. Documents show healthcare.gov couldn't handle 500 users before launch, Nov 2013.

[2] R. Hashemian, D. Krishnamurthy, M. Arlitt, and N. Carlsson. Characterizing the scalability of a web application on a multi core server. Concurrency and Computation: Practice and Experience, 26(12):2027–2052, 2014.

[3] N. Statt. Amazon's website crashed as soon as prime day began, Jul 2018.

[4] The Joint Task Force on Computing Curricula. Curriculum guidelines for undergraduate degree programs in software engineering. Technical report, New York, NY, USA, 2015.

[5] U.S. Bureau of Labor Statistics. Software developers, quality assurance analysts, and testers, Feb 2023.

Trade-off Analysis in Learning-augmented Algorithms with Societal Design Criteria

Mohammad H. Hajiesmaili

University of Massachusetts Amherst
hajiesmaili@cs.umass.edu

ABSTRACT

Traditionally, computer systems are designed to optimize classic notions of performance such as flow completion time, cost, etc. The system performance is then typically evaluated by characterizing theoretical bounds in worst-case settings over a single performance metric. In the next generation of computer systems, societal design criteria, such as carbon awareness and fairness, becomes a first-class design goal. However, the classic performance metrics may conflict with societal criteria. Foundational understanding and performance evaluations of systems with these inherent trade-offs lead to novel research questions that could be considered new educational components for performance analysis courses. The classic techniques, e.g., worst-case analysis, for systems with conflicting objectives may lead to the impossibility of results. However, a foundational understanding of the impossibility of results calls for new techniques and tools. In traditional performance evaluation, to understand the foundational limits, typically, it is sufficient to derive lower-bound arguments in worst-case settings. In the new era of system design, lower bounds are inherently about trade-offs between different objectives. Characterizing these trade-offs in settings with multiple design criteria is closer to the notion of Pareto-optimality, which is drastically different from classic lower bounds. With the impossibility of results using classic paradigms, one possible direction is to (re)design systems following the emerging direction of learning-augmented algorithms. With this approach, it might be possible to remove/mitigate the foundational conflict between classic vs. societal metrics using the right predictions. However, the performance evaluation of learning-augmented algorithms calls for a new set of technical questions, which we highlight in this paper.

1 Introduction

The traditional approach for algorithm design targets classic objective functions that model some notions of *efficiency*, such as performance or cost. The performance of the proposed algorithms is then typically analyzed by characterizing theoretical bounds, e.g., approximation ratio, competitive ratio, regret, etc., in worst-case settings. With the wide deployment of algorithmic ideas in society, it is essential to systematically add *societal* criteria, such as fairness, carbon awareness, safety, privacy, etc., into the system design.

However, the classic efficiency metrics may conflict with societal criteria in several scenarios. We outline two examples of such conflicts in the context of fairness and carbon awareness in computing systems.

Example 1: The trade-off between fairness and competitiveness in the online knapsack problem. The online knapsack problem (`OKP`) [1, 2, 3] (formally introduced in Section 2.2) is well-studied in the literature on online algorithms. In its basic version of `OKP`, one provider allocates a limited resource (i.e., the knapsack's *capacity*) to users arriving sequentially to maximize the total value of admitted users. In `OKP`, as in many other online decision problems, there is a trade-off between *efficiency*, i.e., maximizing the value of the packed items, and *fairness*, i.e., ensuring equitable "treatment" for different items across some desirable criteria [4]. To illustrate the importance of these considerations in the context of `OKP`, it is perhaps best to start with an example.

Consider a cloud computing resource accepting heterogeneous jobs online from clients sequentially. Each job includes a bid the user is willing to pay and the resource requirement. The cloud resource is limited – there are not enough resources to service all incoming requests. Consider the *quality* of a job as the ratio of the bid price paid by the client to the quantity of resources required for it. Note that the limit on the resource implies that the problem of accepting and rejecting items reduces precisely to the online knapsack problem. If we cared only about the overall quality of accepted jobs, we would intuitively be solving the unconstrained online knapsack problem. However, simultaneously, it might be desirable for an algorithm to apply a fair *quality criteria* to each job that arrives. But, adding fair criteria will come at the expense of degrading competitiveness. In Section 4, we formally explore the trade-off between fairness and competitiveness in this context.

Example 2: The trade-off between energy efficiency and carbon efficiency. While the first example was concretely about a specific problem, in the second example, we focus on a more high-level trade-off concept. Motivating by the goal of reducing the carbon footprint of computing systems, recently, there has been attention to elevate the importance of *carbon efficiency*—the ability to do more work when and where low-carbon and clean energy is available—relative to *energy efficiency*—the ability to do the most work for the least amount of energy. While optimizing energy efficiency has been a focus of research in sustainable computing for decades, optimizing carbon efficiency is new and largely under-explored. Technically speaking, optimizing carbon

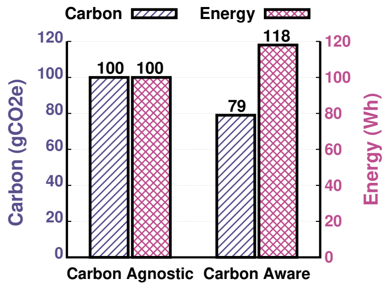

Figure 1: ***Trade-off between energy- vs. carbon-efficiency.***

efficiency is more closely related to the concept of energy flexibility—the degree to which a workload can be shifted temporally or spatially—than energy efficiency. The relationship between energy flexibility and energy efficiency may conflict, i.e., increasing energy flexibility can decrease energy efficiency [5]. A representative example in Figure 1 demonstrates that by while a carbon-aware workload scheduling decreases the carbon footprint by 11%, it increases the energy consumption by 18% (more details on setup and traces in [5]). As another example, data centers are most energy-efficient at high utilization, so leveraging their energy flexibility to reduce carbon emissions by periodically reducing their utilization and power usage makes them less energy-efficient. Another example on theoretical understanding of the trade-off between carbon and energy efficiency is studied in [6].

More broadly, the addition of societal design objectives could lead to multiple other conflicts, such as between safety and regret in online learning [7, 8, 9], and other notions of fairness and learning performance [10, 11].

Learning-augmented algorithms. Besides the emerging topic of societal algorithms design, recently, there has been extensive work on the systematic integration of algorithm design with advice from machine learning. The main motivation is that the classic algorithms (particularly online algorithms, which are the focus of examples in this paper) designed purely with guarantees of the worst-case performance tend to ignore predictions outright. Thus they often have poor performance in common average-case scenarios. In practice, however, for most application scenarios, abundant historical data could be leveraged by machine learning (ML) tools for generating some predictions of the unknown future input, e.g., item values in the online knapsack problem. Then, the possibility of leveraging ML predictions in algorithmic design has led to the recent development of learning-augmented algorithms [12, 13], where the goal is to leverage predictions to improve the performance when predictions are accurate and preserve the robust worst-case guarantees when facing erroneous ML predictions. This high-level idea has led researchers to revisit a wide range of online problems, including but not limited to caching [12], rent-or-buy problems [13, 14, 15], facility location [16, 17], secretary matching [18], metrical task systems [19], bin packing [20], and beyond.

Consistency-robustness Trade-off in learning-augmented algorithms. In the framework of learning-augmented algorithms, there is a natural trade-off between consistency and robustness [12], where consistency represents the competitive ratio (formally defined in (1)) when the prediction is accurate, and robustness is the competitive ratio regardless of the prediction error. The ultimate design goal is to develop an algorithm that can achieve the Pareto-optimal trade-off between consistency and robustness, i.e., no other learning-augmented algorithms can simultaneously achieve better consistency and robustness than the proposed algorithm. A few examples regarding the Pareto-optimality are for ski-rental problem [14], online conversion problem [21], and online matching problem [22].

1.1 Paper Organization

The goal of the remaining sections of this paper is to provide more concrete examples of the trade-off analysis in both societal algorithm design and learning-augmented algorithms to further motivate the need for different notions of trade-off analysis in the broader context of performance evaluation courses. Towards this, in Section 2, we provide a brief background of online algorithms and introduce the online knapsack problem as the running example used in the rest of the paper. Second, in Section 3, we present the consistency-robustness trade-off analysis for learning-augmented online knapsack algorithms. Third, in Section 4, we provide a trade-off analysis between fairness and competitiveness for a notion of time fairness in the knapsack problems. We provide concluding remarks in Section 5.

2 Background

In this section, we provide a brief background on competitive online algorithms and then introduce the classic online knapsack problem.

2.1 Online Algorithms

Decision-making under uncertainty is one of the most challenging issues that real-world computational problems face. In the context of sustainable computing problems, for example, questions like "*will it be enough solar in the next few hours to run the workload later?*" to "*which location will have the lowest carbon intensity to move the workload?*", cannot be answered reliably due to unpredictability of the inputs. Competitive design [23, 24] is a remarkably successful framework for tackling decision making under uncertainty scenarios by developing worst-case optimized algorithms. This framework assumes no stochastic modeling of the input, and its ultimate goal is to devise algorithms with the best possible competitive ratio. Competitive ratio refers to the maximum ratio of the cost incurred by an online algorithm A and the optimal cost incurred by solving the problem in an offline manner under any feasible input instance ω, i.e.,

$$\mathtt{CR}(A) = \max_{\omega\in\Omega} \frac{\mathtt{cost}(A(\omega))}{\mathtt{cost}(\mathtt{opt}(\omega))}, \tag{1}$$

where Ω is the set of all feasible instances, and $\mathtt{cost}(A(\omega))$ and $\mathtt{cost}(\mathtt{opt}(\omega))$ are the cost of A and the offline optimal cost under input ω. This framework has been successfully applied to numerous systems and networking applications such as TCP acknowledgement [25], renting cloud servers [26], dynamic capacity provisioning of data centers [27, 28], energy optimization problems [29, 30, 15], scheduling [31, 32, 33, 34], to name a few.

In the next section, we introduce the online knapsack problem and review the existing competitive algorithms with optimal competitive ratios for this problem.

2.2 The Online Knapsack Problem

The online knapsack problem (OKP) is a classic problem that has been studies extensively in the context of competitive online algorithms. In the basic version of OKP, the goal is to pack items that are arriving online into a knapsack with unit capacity such that the aggregate value of admitted items is maximized. In each round, item $i \in [n] = \{1, \dots, n\}$, with value v_i and weight w_i, arrives, and an online algorithm must decide whether to admit or reject i with the objective of maximizing the total value of selected items while respecting the capacity. More formally, given items' values and weights $\{v_i, w_i\}_{i \in [n]}$, OKP can be formulated as

$$\begin{aligned} [\texttt{OKP}] \quad & \max && \textstyle\sum_{i\in[n]} v_i x_i, \\ & \text{s.t.,} && \textstyle\sum_{i\in[n]} w_i x_i \le 1, \\ & \text{vars.,} && x_i \in \{0,1\}, \quad i \in [n], \end{aligned}$$

where the binary variable $x_i = 1$ denotes the admission of item i and $x_i = 0$ represents a decline. In an online setting, the admission decision x_i for item i must be made only based on the information of current and past items. It is straightforward to show that without any assumptions on the item value and weights, it is impossible to design online algorithms with a bounded competitive ratio for the above formulation of OKP [1]. Hence, in the literature [26, 1, 2, 3], the following two standard assumptions are made to design online algorithms with bounded competitive ratios.

Assumption 1. *The weight of each individual item is much smaller than the unit capacity of the knapsack, i.e.,* $w_i \ll 1, \forall i \in [n]$.

Assumption 2. *The value-to-weight ratio (or value density) of each item is lower and upper bounded between* L *and* U*, i.e.,* $L \le v_i/w_i \le U, \forall i \in [n]$.

Assumption 1 naturally holds in large-scale systems where the capacity of the entire system is way larger than individual requests. Assumption 2 is to eliminate the potential for rare items that have extremely high or low-value densities and again is reasonable from practical perspective. This version of OKP has been used in numerous applications, including online cloud resource allocation [35, 36], budget-constrained bidding in keyword auction [1], online routing [37], and electric vehicle charging scheduling [38, 34, 39].

Prior work on OKP has resulted in an optimal deterministic algorithm for the problem described above, shown by [1] in a seminal work using the framework of online threshold-based algorithms (OTA). In OTA, a carefully designed *threshold function* is used to facilitate the decisions made at each time step. This threshold is specifically designed so that greedily accepting inputs whose values meet or exceed the threshold at each step provides a competitive guarantee. This algorithmic framework has seen success in the related online search and one-way trading problems [38, 40, 41] as well as OKP [1, 2, 3].

The ZCL algorithm: Prior literature [1] proposed a deterministic threshold-based algorithm that achieves a competitive ratio of $\ln(U/L) + 1$. The authors also show that this is the optimal competitive ratio for any deterministic or randomized algorithm. We henceforth refer to this algorithm as the ZCL algorithm. In the ZCL algorithm, items are admitted based on the monotonically increasing threshold function $\Phi(z) = (Ue/L)^z \, (L/e)$, where $z \in [0, 1]$ is the current utilization. The jth item in the sequence is accepted iff it satisfies $v_j/w_j \ge \Phi(z_j)$, where z_j is the utilization at the time of the item's arrival. This algorithm is optimally competitive [1, Theorems. 3.2, 3.3].

In what follows, we provide two examples of trade-off analysis for learning-augmented algorithms (in Section 3) and societal algorithms (in Section 4) for the online knapsack problem.

3 Learning-augmented Algorithms for the Online Knapsack Problem

In this section, we overview a recent consistency-robustness trade-off results for the 1-max search problem [42], which is a simplified version of the online knapsack problem. A 1-max search problem considers how to convert one asset (e.g., dollars) to another (e.g., yens) over a trading period $[N] := \{1, \dots, N\}$. At the beginning of step $n \in [N]$, an exchange rate (or price), v_n, is announced, and a decision maker must immediately determine the amount of dollars, x_n, to convert and obtains $v_n x_n$ yens. The trading horizon N is unknown to the decision maker, and if there are any remaining dollars after $N - 1$ trading steps, all of them will be compulsorily converted to yens at the last price v_N. The 1-max search problem is a special case of OKP in the sense of setting item sizes equal to the capacity of the knapsack and the goal of picking the top most valuable item. If the asset is allowed to convert fraction-by-fraction over multiple transactions, the decision $x_n \in [0, 1]$ is a continuous variable, and this fractional version is referred to as *one-way trading* [43]. Similar to that of OKP, we assume the prices $\{v_n\}_{n\in[N]}$ are bounded, i.e., $v_n \in [L, U], \forall n \in [N]$, where L and U are known parameters, and define $\theta = U/L$ as the price fluctuation.

The optimal algorithm for 1-max-search. There is a simple threshold-based algorithm, which determines a threshold function as a constant $\Phi = \sqrt{UL}$, where Φ is also called a reservation price. Then the algorithm selects the first price that is at least Φ. In [43], it has been shown that this algorithm achieves the optimal competitive ratio $\sqrt{\theta}$.

3.1 1-max search with prediction

In this section, we review an existing algorithm with a Pareto-optimal trade-off between consistency-robustness for the 1-max search problem. We refer to [21] for the full explanation of the results. First, we assume that a prediction of the maximum price P is given to the learning-augmented algorithm. The goal is to design the reservation price Φ_P given a prediction P. We denote η as the consistency and γ as the robustness. set $\eta := \eta(\lambda)$ and $\gamma := \gamma(\lambda)$ as

$$\gamma(\lambda) = [\sqrt{(1-\lambda)^2 + 4\lambda\theta} - (1-\lambda)]/(2\lambda), \text{ and } \eta(\lambda) = \theta/\gamma(\lambda), \tag{3}$$

where $\lambda \in [0, 1]$ is the robustness parameter. In other words, parameter λ determines the level of trust on prediction P, where $\lambda = 0$ means full trust; and $\lambda = 1$ means no trust at all. and η and γ are predetermined parameters for designing Φ_P that represent the consistency and robustness that we target to achieve. In particular, η and γ are designed as the solution of

$$\eta(\lambda) = \theta/\gamma(\lambda), \text{ and } \eta(\lambda) = \lambda\gamma(\lambda) + 1 - \lambda. \tag{4}$$

The first equation is the desired trade-off between robustness and consistency and thus represents a Pareto-optimal trade-off. The second equation sets η as a linear combination of 1 and γ. In this way, as λ increases from 0 to 1 , η increases from the best possible ratio 1 to the optimal competitive ratio $\sqrt{\theta}$, and γ decreases from the worst possible ratio θ to $\sqrt{\theta}$. Taking η and γ as inputs, we design the reservation price Φ_P as follows:

$$\text{when } P \in [L, L\eta),\ \Phi_P = L\eta; \tag{5a}$$

$$\text{when } P \in [L\eta, L\gamma),\ \Phi_P = \lambda L\gamma + (1-\lambda)P/\eta; \tag{5b}$$

$$\text{when } P \in [L\gamma, U],\ \Phi_P = L\gamma. \tag{5c}$$

The following theorem provides robustness and consistency bounds for this algorithm.

THEOREM 1. *Given $\lambda \in [0,1]$,* OTA *with the reservation price in Equation* (5) *for* 1*-max-search is $\gamma(\lambda)$-robust and $\eta(\lambda)$-consistent, where $\gamma(\lambda)$ and $\eta(\lambda)$ are given in Equation* (3)*.*

THEOREM 2. *Any γ-robust learning-augmented online algorithms for 1-max-search must have consistency $\eta \geq \theta/\gamma$. Thus, the algorithm proposed with the reservation price* (5) *is Pareto-optimal.*

For additional insights on the algorithm design and Pareto-optimal trade-off, we refer to [21]. Putting together the results in the above two theorems, we conclude that the consistency-robustness trade-off of the above algorithm is Pareto-optimal. As a concluding remark for this section, we further note that the Pareto-optimal trade-off analysis in the context of learning-augmented algorithms is an emerging topic, and to the best of our knowledge, finding a Pareto-optimal learning-augmented algorithm for the general online knapsack is still an open problem.

It is worth noting that the main purpose of presenting the Pareto-optimality trade-off results was to highlight the contrast with respect to classic competitive analysis where the optimality of results reduces to only showing a lower bound on a single criterion instead of two (or multiple) criteria as in learning-augmented algorithm design.

4 Trade-offs in Social Algorithm Design

To show the potential trade-offs between societal vs. classic design criteria in performance analysis of the algorithms, we demonstrate the results in [4] as a running example. The purpose of this example is just to provide an example of a trade-off between fairness (as a societal criterion) and competitiveness (as an efficiency metric). The high-level concept could be applicable to other societal concerns such as carbon awareness as we mentioned in the introduction.

4.1 Fairness in Online Knapsack Problems

In this section, we briefly explore this trade-off in the context of the online knapsack problem. We refer to [4] for a comprehensive statement of the results.

4.1.1 Trade-off Results

Fairness definition. The example in the introduction presented a specific type of time fairness that was explored in the context of similar problems such as prophet inequalities [44]. It is reasonable to ask that the probability of an item's admission into the knapsack should depend solely on its value density x, and not on its arrival time j. We begin by generalizing the definition of Time-Independent Fairness proposed in [44] to OKP. Motivated by these results, in Definition 3, we present a slightly revised notion, which relaxes this constraint and narrows the scope of fairness to consider items that arrive while the knapsack's utilization is in some subinterval of the knapsack's capacity. In the following, we formally define the notion of α-Conditional Time-Independent Fairness (α-CTIF) for OKP.

DEFINITION 3. *For $\alpha \in [0,1]$, an OKP algorithm* ALG *is said to satisfy α-CTIF if there exists a subinterval $\mathcal{A} = [a,b] \subseteq [0,1]$ where $b - a = \alpha$, and a function $p : [L,U] \to [0,1]$ such that:*

$$\Pr\left[\textsf{ALG} \textit{ accepts item } j \textit{ in } \mathcal{I} \mid \left(\frac{v_j}{w_j} = x\right) \wedge (z_j + w_j \in \mathcal{A})\right] = p(x),$$

$$\forall \mathcal{I} \in \Omega, j \in [|\mathcal{I}|], x \in [L,U].$$

In particular, if $\alpha = 1$, then $\mathcal{A} = [0,1]$, and any item that arrives while the knapsack still can admit it is considered. Using Definition 3, in this section we present algorithms that satisfy CTIF constraints while remaining competitive and leveraging predictions for better performance. We start with a result that captures the essence of the trade-offs inherent to this problem.

THEOREM 4. *Any constant threshold-based algorithm for OKP satisfies* 1*-CTIF. Furthermore, any constant threshold-based deterministic algorithm for OKP cannot be better than (U/L)-competitive.*

We can now extend these results to general values of α.

Extended Constant Threshold (ECT). We define a threshold function $\Phi^\alpha(z)$ on the interval $z \in [0,1]$, where z_j is the knapsack utilization when the jth item arrives, and $\alpha \in [1/(\ln(U/L)+1), 1]$ is the *fairness parameter.* Φ^α is defined as follows:

$$\Psi^\alpha(z) = \begin{cases} L & z \in [0,\alpha], \\ Ue^{\beta(z-1)} & z \in (\alpha, 1], \end{cases} \tag{6}$$

where $\beta = \frac{W\left(\frac{U(1-\alpha)}{L\alpha}\right)}{1-\alpha}$. The following result shows the achieved trade-off between fairness and competitiveness in the above algorithm.

THEOREM 5. ECT$[\alpha]$ *satisfies α-CTIF. Furthermore, for any instance $\mathcal{I} \in \Omega$, we have*

$$\textsf{OPT}(\mathcal{I}) \leq \textsf{ECT}[\alpha](\mathcal{I}) \cdot \frac{U[\ln(U/L)+1]}{L\alpha[\ln(U/L)+1] + (U-L)(1-\ell)}.$$

Thus, ECT$[\alpha]$ *is $\frac{U[\ln(U/L)+1]}{L\alpha[\ln(U/L)+1]+(U-L)(1-\ell)}$-competitive.* ECT$[\alpha]$*, in fact,* ***exactly*** *achieves the Pareto-optimal competitiveness trade-off.*

While the above results provide a tight trade-off between fairness and competitiveness, in the following, we show how to improve the trade-off by using simple predictions.

4.1.2 *Learning-augmented Design Helps*

Prediction model. Consider an offline approximation algorithm APX for OKP, which sorts items by non-increasing value density and packs them in this order. Let $x \in [L, U]$ denote the smallest value density of any packed item, and V is the total value obtained by APX. Then, if the total value of items with value density x in the knapsack is $\geq V/2$, define $d^\star := x$. Otherwise, define $d^\star := x^+$, where x^+ is the next highest value density in $\mathcal{I}$. We assume that our algorithm receives a single prediction $\hat{d} \in [L, U]$ for each instance, where the prediction is perfect if $\hat{d} = d^\star$.

Learning-Augmented Extended Constant Threshold (LA-ECT). Fix a *trust parameter* $\gamma \in [0, 1]$. We define the threshold function $\Psi^{\gamma,\hat{d}}(z)$:

$$\Psi^{\gamma,\hat{d}}(z) = \begin{cases} (Ue/L)^{\frac{z}{1-\gamma}}\,(L/e) & z \in [0, \kappa], \\ \hat{d} & z \in (\kappa, \kappa+\gamma), \\ (Ue/L)^{\frac{z-\gamma}{1-\gamma}}\,(L/e) & z \in [\kappa+\gamma, 1], \end{cases} \tag{7}$$

where κ is the point where $(Ue/L)^{(z/1-\gamma)}(L/e) = \hat{d}$. Call the resulting threshold algorithm $\mathsf{LA\text{-}ECT}[\gamma]$. The following theorem characterizes the fairness as well as the trade-off between consistency and robustness for this algorithm.

Theorem 6. $\mathsf{LA\text{-}ECT}[\gamma]$ *satisfies γ-CTIF. Also, for any $\mathcal{I} \in \Omega$,*

- *For any accurate prediction $\hat{v} \in [L, U]$, we will have* $\mathsf{ORACLE}(\mathcal{I}) \leq \mathsf{LA\text{-}ECT}[\gamma](\mathcal{I}) \cdot \frac{\varrho+2}{\gamma}$.
- *For any prediction $\hat{v} \in [L, U]$, we have*

$$\mathsf{OPT}(\mathcal{I}) \leq \mathsf{LA\text{-}ECT}[\gamma](\mathcal{I}) \cdot (1/1-\gamma)\ln(U/L) + 1.$$

Thus, $\mathsf{LA\text{-}ECT}[\gamma]$ *is* $\left(\frac{1}{1-\gamma}\ln(U/L) + 1\right)$*-robust. For most instances, $\varrho = O(1)$, and so* $\mathsf{LA\text{-}ECT}[\gamma]$ *is $O(1/\gamma)$-consistent.*

The proposed learning-augmented algorithm substantially improves the performance in practice, as shown in the experiments in [4]. Hence, interestingly the learning-augmented algorithm design paradigm is an appropriate tool to improve the conflicting trade-offs between classic and societal design criteria.

5 Concluding Remarks

In this paper, we highlighted two notions of trade-off analysis in the context of (1) learning-augmented algorithms design, where the trade-off is between consistency and robustness; and (2) algorithms design with societal criteria, e.g., fairness and carbon awareness, where the trade-off is between classic performance notions, e.g., competitive ratio, and societal metrics. Interestingly, leveraging learning-augmented design could be considered as a potential tool to improve the trade-offs in societal algorithm design. Lastly, these trade-offs are inherent to both emerging topics and could be considered as new teaching elements to classic performance evaluations and algorithm design and analysis courses.

Acknowledgment

The author acknowledges that the technical parts of this paper are from collaborative research projects with Shahin Kamali, Adam Lechowicz, Russell Lee, Rik Sengupta, Bo Sun, and Adam Wierman.

6 References

[1] Y. Zhou, D. Chakrabarty, and R. Lukose, "Budget constrained bidding in keyword auctions and online knapsack problems," in *Proc. of WINE*, pp. 566–576, 2008.

[2] B. Sun, L. Yang, M. Hajiesmaili, A. Wierman, J. C. Lui, D. Towsley, and D. H. Tsang, "The online knapsack problem with departures," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 3, pp. 1–32, 2022.

[3] L. Yang, A. Zeynali, M. H. Hajiesmaili, R. K. Sitaraman, and D. Towsley, "Competitive algorithms for online multidimensional knapsack problems," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 3, pp. 1–30, 2021.

[4] A. Lechowicz, R. Sengupta, B. Sun, S. Kamali, and M. Hajiesmaili, "Time fairness in online knapsack problems," *arXiv preprint arXiv:2305.13293*, 2023.

[5] W. A. Hanafy, R. Bostandoost, N. Bashir, D. Irwin, M. Hajiesmaili, and P. Shenoy, "The war of the efficiencies: Understanding the tension between carbon and energy optimization," in *The 2nd Workshop on Sustainable Computer Systems (HotCarbon'23)*, 2023.

[6] A. Lechowicz, N. Christianson, J. Zuo, N. Bashir, M. Hajiesmaili, A. Wierman, and P. Shenoy, "The online pause and resume problem: Optimal algorithms and an application to carbon-aware load shifting," *arXiv preprint arXiv:2303.17551*, 2023.

[7] L. Yang, M. Hajiesmaili, M. S. Talebi, J. Lui, W. S. Wong, *et al.*, "Adversarial bandits with corruptions: Regret lower bound and no-regret algorithm," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19943–19952, 2020.

[8] T. Lykouris, V. Mirrokni, and R. Paes Leme, "Stochastic bandits robust to adversarial corruptions," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 114–122, 2018.

[9] A. Gupta, T. Koren, and K. Talwar, "Better algorithms for stochastic bandits with adversarial corruptions," in *Conference on Learning Theory*, pp. 1562–1578, PMLR, 2019.

[10] D. Freund, T. Lykouris, E. Paulson, B. Sturt, and W. Weng, "Group fairness in dynamic refugee assignment," *arXiv preprint arXiv:2301.10642*, 2023.

[11] S. R. Sinclair, S. Banerjee, and C. L. Yu, "Sequential fair allocation: Achieving the optimal envy-efficiency tradeoff curve," *ACM SIGMETRICS Performance Evaluation Review*, vol. 50, no. 1, pp. 95–96, 2022.

[12] T. Lykouris and S. Vassilvtiskii, "Competitive caching with machine learned advice," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 3296–3305, PMLR, 10–15 Jul 2018.

[13] M. Purohit, Z. Svitkina, and R. Kumar, "Improving online algorithms via ml predictions," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31,

Curran Associates, Inc., 2018.

[14] A. Wei and F. Zhang, "Optimal robustness-consistency trade-offs for learning-augmented online algorithms," *Advances in Neural Information Processing Systems*, vol. 33, pp. 8042–8053, 2020.

[15] R. Lee, J. Maghakian, M. Hajiesmaili, J. Li, R. Sitaraman, and Z. Liu, "Online peak-aware energy scheduling with untrusted advice," in *Proc. of ACM -e-Energy*, 2021.

[16] D. Fotakis, E. Gergatsouli, T. Gouleakis, and N. Patris, "Learning augmented online facility location," *arXiv preprint arXiv:2107.08277*, 2021.

[17] S. H.-C. Jiang, E. Liu, Y. Lyu, Z. G. Tang, and Y. Zhang, "Online facility location with predictions," in *International Conference on Learning Representations*, 2021.

[18] A. Antoniadis, T. Gouleakis, P. Kleer, and P. Kolev, "Secretary and online matching problems with machine learned advice," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7933–7944, 2020.

[19] A. Antoniadis, C. Coester, M. Elias, A. Polak, and B. Simon, "Online metric algorithms with untrusted predictions," in *International Conference on Machine Learning*, pp. 345–355, PMLR, 2020.

[20] S. Angelopoulos, S. Kamali, and K. Shadkami, "Online bin packing with predictions," in *Proc. of IJCAI*, 2022.

[21] B. Sun, R. Lee, M. Hajiesmaili, A. Wierman, and D. Tsang, "Pareto-optimal learning-augmented algorithms for online conversion problems," in *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[22] B. Jin and W. Ma, "Online bipartite matching with advice: Tight robustness-consistency tradeoffs for the two-stage model," *arXiv preprint arXiv:2206.11397*, 2022.

[23] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, "Competitive snoopy caching," in *Proc. of IEEE FOCS*, pp. 244–254, 1986.

[24] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. cambridge university press, 2005.

[25] A. R. Karlin, C. Kenyon, and D. Randall, "Dynamic TCP acknowledgement and other stories about e/(e-1)," in *Proc. of ACM STOC*, pp. 502–509, 2001.

[26] Z. Zhang, Z. Li, and C. Wu, "Optimal posted prices for online cloud resource allocation," in *Proc. of ACM SIGMETRICS*.

[27] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1378–1391, 2012.

[28] L. Yang, M. H. Hajiesmaili, R. Sitaraman, A. Wierman, E. Mallada, and W. S. Wong, "Online linear optimization with inventory management constraints," *Proc. of ACM SIGMETRICS*, 2020.

[29] L. Yang, M. H. Hajiesmaili, H. Yi, and M. Chen, "Hour-ahead offering strategies in electricity market for power producers with storage and intermittent supply," in *Proc of. ACM SIGMETRICS*, 2017.

[30] Y. Zhang, M. H. Hajiesmaili, S. Cai, M. Chen, and Q. Zhu, "Peak-aware online economic dispatching for microgrids," *IEEE Transactions on Smart Grid*, vol. 9, no. 1, pp. 323–335, 2018.

[31] B. Alinai, M. S. Talebi, M. H. Hajiesmaili, A. Yekkehkhany, and N. Crespi, "Competitive online scheduling algorithms with applications in deadline-constrained EV charging," in *Proc. of IEEE/ACM IWQoS*, 2018.

[32] B. Alinia, M. H. Hajiesmaili, Z. J. Lee, N. Crespi, and E. Mallada, "Online EV scheduling algorithms for adaptive charging networks with global peak constraints," *IEEE Transactions on Sustainable Computing*, 2020.

[33] L. Yang, W. S. Wong, and M. H. Hajiesmaili, "An optimal randomized online algorithm for QoS buffer management," in *Proc. ACM SIGMETRICS*, 2018.

[34] R. Bostandoost, B. Sun, C. Joe-Wong, and M. Hajiesmaili, "Near-optimal online algorithms for joint pricing and scheduling in ev charging networks," in *Proc. of ACM e-Energy*, 2023.

[35] S. R. M. Amarante, F. M. Roberto, A. R. Cardoso, and J. Celestino, "Using the multiple knapsack problem to model the problem of virtual machine allocation in cloud computing," in *Proc. of IEEE CSE*, 2013.

[36] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. Lau, "Online auctions in IaaS clouds: Welfare and profit maximization with server costs," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1034–1047, 2017.

[37] N. Buchbinder and J. S. Naor, "The design of competitive online algorithms via a primal–dual approach," *Foundations and Trends® in Theoretical Computer Science*, vol. 3, no. 2–3, pp. 93–263, 2009.

[38] B. Sun, A. Zeynali, T. Li, M. Hajiesmaili, A. Wierman, and D. H. Tsang, "Competitive algorithms for the online multiple knapsack problem with application to electric vehicle charging," *Proc. of ACM SIGMETRICS*, 2021.

[39] A. Zeynali, B. Sun, M. Hajiesmaili, and A. Wierman, "Data-driven competitive algorithms for online knapsack and set cover," in *Proc. AAAI*, 2021.

[40] J. Lorenz, K. Panagiotou, and A. Steger, "Optimal algorithms for k-search with application in option pricing," *Algorithmica*, vol. 55, no. 2, pp. 311–328, 2009.

[41] R. El-Yaniv, A. Fiat, R. M. Karp, and G. Turpin, "Optimal search and one-way trading online algorithms," *Algorithmica*, vol. 30, no. 1, pp. 101–139, 2001.

[42] R. El-Yaniv, A. Fiat, R. M. Karp, and G. Turpin, "Optimal search and one-way trading online algorithms," *Algorithmica*, vol. 30, pp. 101–139, May 2001.

[43] R. El-Yaniv, A. Fiat, R. M. Karp, and G. Turpin, "Optimal search and one-way trading online algorithms," *Algorithmica*, vol. 30, no. 1, pp. 101–139, 2001.

[44] M. Arsenis and R. Kleinberg, "Individual fairness in prophet inequalities," in *Proc. of ACM EC*, 2022.

The Role of Advanced Math in Teaching Performance Modeling

Ziv Scully
Cornell University
School of Operations Research and Information Engineering
zivscully@cornell.edu

ABSTRACT

How should we teach performance modeling without assuming a deep mathematical background? One approach is to focus on rigorously studying relatively simple stochastic models that do not require too much math background. But this may leave students underprepared to reason about systems in practice. They have multiple servers, bursty arrivals, heavy tails, and other features that demand more complex stochastic models. Reasoning about these phenomena calls for advanced tools from performance modeling theory, but rigorously learning such tools requires more math background than many computer science and engineering students have.

In my view, the main obstacle to teaching advanced theoretical tools is the mathematical rigor. I believe we can teach such tools accessibly by dispensing with some of the rigor. In this (opinionated!) abstract, I argue that students would be well-served by advanced theoretical tools, and I outline what teaching those tools with less rigor might look like.

1. THE NEED FOR ADVANCED MATH IN PERFORMANCE MODELING

In my view, the main goals of performance modeling are to *describe* and *analyze* systems in service of improving their design. Math plays a critical role in both goals.

- We formally describe systems as mathematical models, typically stochastic models.
- We analyze said models either in theory, using tools from applied probability; or in simulation, in which case probability and statistics help us decide what to simulate and interpret results.

A typical path for performance modeling courses is to introduce students to a mathematical modeling framework, then teach strategies for analyzing models in that framework.

1.1 Markov Chains Fall Short

A modeling framework of choice for many texts on performance modeling and stochastic processes is Markov chains on discrete state spaces [4, 5, 12, 16, 23, 25, 30]. Markov chains are appealing for several reasons.

- One can define and prove fundamental theorems about Markov chains with elementary tools. For example, on finite state spaces, one can define the transition kernel as a matrix, and one can frame questions about convergence and mixing times in terms of linear algebra.
- One can often exactly analyze Markov chains with elementary tools. For example, while guessing a Markov chain's stationary distribution can be difficult, confirming a guess requires only checking balance equations.

This combination makes it feasible for students to learn to define and exactly analyze Markov chain models fully rigorously.

However, I see two main downsides to focusing a performance modeling course around Markov chains. The first downside is concrete: Markov chains have limited modeling power. It is difficult to model queueing systems with general service time distributions with Markov chains, outside of specific cases like the M/G/1 and G/M/1 where an embedded discrete-time approach is possible. Phase-type distributions can approximate general distributions, but their tails never resemble power-law tails, which are ubiquitous in computer systems and beyond [7, 22, 26, 31]. One can work around this by adding more states to the Markov chain so that jobs have infinitely many possible phases. But this yields Markov chains that are challenging or impossible to exactly analyze.

This brings us to a second downside of focusing on Markov chains: they tempt a focus on exact analysis, as opposed to approximations or bounds. It is true that many Markov-chain queueing models admit exact analysis, such as the famously general Jackson networks [16]. But exact analysis is intractable for most models, particularly multiserver models like the M/G/k [11, 17]. Why do courses focus so heavily on exact analysis? I suspect it is because exact analysis is often easier than approximations. Verifying the balance equations for Jackson networks, for example, is just algebra. Approximately analyzing systems like the M/G/k requires more advanced machinery [13, 27].

In summary: Markov chains are an appealing because they can be taught fully rigorously without assuming an extensive math background. But the set of Markov chains that can be rigorously analyzed with only elementary tools is limited. I believe this status quo can leave students unprepared to reason about the systems they will encounter in practice.

1.2 Tools Students Need to Build and Analyze Realistic Models

What features of practical systems might students need to reason about, and what mathematical tools do they need to do so? Below are four tools I think would prove useful, but this is not an exhaustive list.

The first tool is *general state spaces*. Even very simple models require complex state spaces. Consider an M/G/1 queue where we wish to track the remaining work of each job. The system state is a list $[r_1, \ldots, r_n]$, where r_i is the remaining work of job i. The state space is thus $\bigsqcup_{i=0}^{\infty} \mathbb{R}_+^i$, which is not even finite-dimensional. Computer systems, with their many layers of abstraction, have even more complicated state spaces. For instance, a data center has many physical servers, each of which hosts many virtual machines, each of which runs many programs, each of which has its own internal software state.

The second tool is *drift* of a system's state, or the drift of functions of the state. An especially important example is drift of the total remaining work of all jobs in a system, which is related to the system's *load*. Drift is important because they give students a first indication about how a system might behave. For example, if load is greater than the average work rate, the system will be unstable. Practical factors like parallelism [3] and garbage collection [14] can make figuring out a computer system's load difficult. Learning to reason about drift is one way to understand load and stability more broadly. Drift methods, as pioneered by Eryilmaz and Srikant [10], are also one of the principal tools queueing theorists have for approximately analyzing otherwise intractable systems.

The third tool is *expectations from varying perspectives*. There are many ways we might look at a system's average behavior, such as averaging over time, averaging over sample paths, and averaging over jobs. Exactly which average is appropriate depends on what metrics we are trying to analyze in theory or measure in simulation. To name two recent examples:

- Kumar et al. [18] investigate the degree to which websites rely on a small number of centralized services, such as content delivery networks. Much of their analysis is in terms of website averages, namely the fraction of websites from a given list satisfy a criterion. But there are other types of averages that one could investigate. One example is website-visit averages, namely the fraction of website visits that satisfy a criterion; or, equivalently, a website average where each website is weighted by the number of visits it gets.
- Atre et al. [1] design an algorithm for caching with bursty requests. Types of averages that could be relevant include time averages, request averages, and request-burst averages. They use request-burst averages in their algorithm, but one could imagine a different average would yield different results, and it is not a priori obvious which type of average is the right choice.

The fourth tool is *large-scale approximations*, such as *mean-field models*. Today's computer systems are certainly large-scale, and mean-field models can quickly give some insight into a large-scale system's behavior. This insight might be from numerically evaluating the mean-field model, or from proving theorems about it. Famously, mean-field models have been used to study dispatching [20]. I think mean-field models could give students insight into metastability [9], a phenomenon that is behind many recent failures in computer systems [6, 14].

1.3 What Makes These Tools Advanced

All four of the tools above are typically considered mathematically advanced. When taking a fully rigorous approach, this is certainly the case.

- Studying stochastic processes on general state spaces raises measure theoretic concerns.
- In continuous-time models, defining the drift of a Markov process involves the process's infinitesimal generator, an operator whose domain is hard to define.
- Defining expectations from certain perspectives involves what seems to be conditioning on a probability-zero event. For example, one cannot simply define an arrival average as a time average conditional on an arrival occurring. *Palm calculus* [2] is a rigorous method of overcoming this, but the formalism can be intimidating even to experts (myself included!), let alone students.
- One typically derives mean-field models as infinite-size limits of large but finite-size models. But this requires reasoning about limits of stochastic processes and raises concerns about when large-time and large-size limits may be exchanged.

Teaching these tools rigorously thus seems infeasible in the context of an engineering course.

My view is that the main thing that makes these tools advanced is the rigor. I think there is a way to teach them that dodges much of the rigor while still providing value to students. This is the subject of the next section.

2. HOW WE MIGHT TEACH THE MATH PERFORMANCE MODELING NEEDS

The conclusion of the previous section is that students would be well served by advanced theoretical tools, but that there is not time to teach them rigorously in the context of an engineering course. The natural question is: how do we teach such tools less rigorously? In this section, I outline the beginnings of an approach for doing so. Given the amount of hand-waving involved, I will refer to my proposed approach using the acronym *WAVE*.

The main issue with hand-waving without a rigorous foundation is that it can leave students unsure about exactly when hand-waving is allowed. In WAVE, I hope to state precise rules for hand-waving, which I refer to as *principles* (as distinct from theorems), that work "most of the time". To help students solidify their understanding of when principles apply, I plan to "prove" most principles in some way. This may be by picture, by computation, or even by appeal to empirical data. To further guide students, WAVE provides some common patterns for applying principles, which I refer to as *recipes*.

A downside of teaching using high-level principles and recipes, as opposed to more rigorous low-level statements, is that each individual topic might need many principles and recipes. To get around this, I hope to focus WAVE on a small number of flexible principles and recipes that can be applied in many different contexts. The first step is to introduce a flexible modeling framework to which those principles apply.

2.1 Model with Markov Processes

I propose we teach students to model systems as Markov processes on general state spaces in either discrete or continuous time. The discrete time version of this is not so different from the current Markov chain approach. The main difference is one of emphasis: rather than focusing attention on easily tractable chains like birth-death processes, the main goal is to capture the key dynamics of a system. A learning goal for students should be to decide what aspects of a system should be tracked as part of its state in order to fully specify its transition dynamics.

I focus hereafter on continuous time, as that is the setting where technical issues arise in a fully rigorous treatment. An immediate question is: how should students specify the dynamics of a continuous-time Markov process? In WAVE, students specify dynamics in an informal but clear pseudocode. For example, Algorithm 2.1 below describes an M/G/1 with arrival rate λ and job size distribution S, tracking the size and attained service of each job as part of the state. There are two ways the state can change: *continuously*, such as a job being served; and through *jumps*, such as a job arriving or departing. This is a natural way to describe piecewise-deterministic Markov processes [8], and I suspect it will suffice for most queueing applications.

Algorithm 2.1 M/G/1 Queue with Known Job Sizes

```
STATE: a list $X = [(s_1, a_1), \dots, (s_n, a_n)]$, where $n \in \mathbb{N}$ and
$0 \le a_i \le s_i$ for all $i \in \{1, \dots, n\}$
  • $s_i$ represents the size of job $i$
  • $a_i$ represents the attained service of job $i$
DYNAMICS:
 (A) Continuously while $n \ge 1$:
         Increase $a_1$ at rate 1
 (B) Jump at hazard rate $\lambda$:
         Sample $s_{n+1}$ from $S$
         Set $a_{n+1}$ to 0
         Append $(s_{n+1}, a_{n+1})$ to $X$
 (C) Jump when $n \ge 1$ and $a_1 = s_1$:
         Delete $(s_1, a_1)$ from $X$          (and shift indices)
```

The main choice students make when modeling a system is what information to include in the system state. One aspect of this choice is deciding what information should be considered known or unknown. One can view Algorithm 2.1 as being an M/G/1 with known job sizes, because the job sizes s_i are tracked in the system state. A variation with unknown job sizes, which makes use of the hazard rate $h_S(\cdot)$ of the job size distribution, is given in Algorithm 2.2 below.

Algorithm 2.2 M/G/1 Queue with Unknown Job Sizes

```
STATE: a list $X = [a_1, \dots, a_n]$, where $n \in \mathbb{N}$ and $a_i \ge 0$ for
all $i \in \{1, \dots, n\}$
  • $a_i$ represents the attained service of job $i$
DYNAMICS:
 (A) Continuously while $n \ge 1$:
         Increase $a_1$ at rate 1
 (B) Jump at hazard rate $\lambda$:
         Set $a_{n+1}$ to 0
         Append $a_{n+1}$ to $X$
 (C) Jump at hazard rate $h_S(a_1)$:
         Delete $a_1$ from $X$          (and shift indices)
```

Using pseudocode does not completely shield students from learning math. For instance, hazard rates feature in both Algorithms 2.1 and 2.2. But I believe some version of pseudocode could be intuitive, especially to students with programming background.

A question one might ask is what would need to be done to rigorize pseudocode descriptions like Algorithms 2.1 and 2.2. I suspect the main obstacle would be to formalize what conditions are admissible for jumps. One would likely want to check for non-explosiveness, meaning the process has probability zero of having infinitely many jumps in a finite time interval. I believe students can benefit from pseudocode descriptions without worrying about explosiveness in most cases.

2.2 Measure with Expectations from Varying Perspectives

Having defined a system model, how should we define metrics of interest, such as mean waiting time? WAVE is focused on metrics that can be expressed in the form

$$\mathbf{E}_\ell[f(X)],$$

in which:

- X is the system state.
- f is some numerical function of the system state.
- $\mathbf{E}_\ell[\cdot]$ is a expectation from *perspective* ℓ. We use the letter ℓ because perspectives will often be *lines* or *labels* in the pseudocode description of X's evolution.[1]

I explain all three aspects in more detail below via two examples. Both examples use the M/G/1 with known job sizes from Algorithm 2.1. The metrics of interest are mean waiting time and mean queue length.

We first consider mean waiting time. A job's waiting time as the amount of work in the system when it arrives, so we start defining a function for the amount of work:

$$w\big([(s_1, a_1), \dots, (s_n, a_n)]\big) = (s_1 - a_1) + \dots + (s_n - a_n).$$

That is, $w(x)$ is the amount of work when the system is in state x. The system state X evolves randomly over time, so we write $X(t)$ for the system state at time t. We suppose the system runs for a very long time interval $[0, T]$. If the times jobs arrive are $t_1, \dots, t_N$, where N is the number of arrivals in $[0, T]$, then the mean waiting time of jobs that arrive during the interval is

$$\frac{1}{N} \sum_{i=1}^{N} w(X(t_i)).$$

More generally, we define $\mathbf{E}_{\mathsf{arrival}}[f(X)]$ for function f to be

$$\mathbf{E}_{\mathsf{arrival}}[f(X)] = \frac{1}{N} \sum_{i=1}^{N} f(X(t_i-)).$$

So mean waiting time is $\mathbf{E}_{\mathsf{arrival}}[w(X)]$.

We now consider mean queue length. We start by defining a function for the queue length:

$$q\big([(s_1, a_1), \dots, (s_n, a_n)]\big) = (n-1)^+ = \max\{n-1, 0\}.$$

That is, $q(x)$ is the number of jobs in the queue, not counting the job in service, when the system is in state x. Mean queue length is a time average, so we next define $\mathbf{E}_{\mathsf{arrival}}[f(X)]$ for function f to be

$$\mathbf{E}_{\mathsf{time}}[f(X)] = \frac{1}{T} \int_0^T f(X(t))\,\mathrm{d}t.$$

[1]Such expectations are sometimes called *Palm expectations* in the literature [2, 21].

So mean queue length is $\mathbf{E}_{\mathsf{time}}[q(X)]$.

Above, arrival and time are two examples of a *perspectives*. A perspective is a way of taking a long-run average. I suspect that most useful perspectives will be time and perspectives associated with *jump labels*, meaning lines of pseudocode that specify jumps. The arrival perspective is a special case of this: $\mathbf{E}_{\mathsf{arrival}}[\cdot] = \mathbf{E}_{(\mathsf{B})}[\cdot]$, because (B) is the label corresponding to arrivals in Algorithm 2.1. In general, for jump labels ℓ, we define $\mathbf{E}_\ell[\cdot]$ to be the average taken over the N_ℓ times $t_{\ell,1}, \dots, t_{\ell,N_\ell}$ a jump occurs due to label ℓ:

$$\mathbf{E}_\ell[f(X)] = \frac{1}{N_\ell} \sum_{i=1}^{N_\ell} f(X(t_{\ell,i}-)).$$

One could imagine defining perspectives for continuous labels, too. Again using Algorithm 2.1 as an example, (A) would be the perspective of a busy system, meaning an average that excludes times the server is idle. It is not yet clear to me whether such perspectives would be useful.

2.3 Quantify with WAVE Equality

We have defined expectations as long run averages on a single sample path $X(t)$ for $t \in [0, T]$. But we have not yet said anything about the probability space underlying $X(t)$. WAVE makes no explicit mention of probability spaces, instead focusing on a single long sample path. How, then, can we use probabilistic information, such as the fact that the number of arrivals N during $[0, T]$ should be approximately λT? Traditionally, we would use the law of large numbers to deduce this.

Instead of defining a probability space, I propose we *make the law of large numbers an axiomatic principle*, or a small number of related principles. One such principle (or a special case thereof) would say that when jumps occur at constant hazard rate λ during a union of intervals of total length T, the number of jumps N is[2]

$$N \approx \lambda T.$$

In a rigorous presentation, the $\approx$ above would refer to a type of convergence, such as almost sure convergence as $T \to \infty$. Under WAVE, $\approx$ is a new equivalence relation which we call *WAVE equality*.

Intuitively, WAVE equality means "equal enough for practical purposes if the sample path is long enough". But there is no direct definition of WAVE equality. The closest we can get to giving a formal definition of WAVE equality is to say it is defined inductively via principles. That is, the only way to show that two quantities are WAVE-equal is to use a principle, and two quantities are WAVE-equal if some application of principles can show them to be. Of course, the principles are informal, so this is still not a formal definition.

In WAVE, most interesting principles hold only under WAVE equality, as opposed to ordinary equality. For example, the WAVE version of Little's law [19] is

$$\mathbf{E}_{\mathsf{arrival}}[q(X)] \approx \lambda \mathbf{E}_{\mathsf{time}}[w(X)].$$

We will soon give an argument for Little's law using WAVE. Another example is the PASTA (Poisson Arrivals See Time Averages) principle [32], whose WAVE version says that for all functions f,

$$\mathbf{E}_{\mathsf{arrival}}[f(X)] \approx \mathbf{E}_{\mathsf{time}}[f(X)].$$

2.4 Analyze with the Main WAVE Principle

Having defined metrics like mean waiting time and mean queue length, how do we actually compute them? The starting point is the main WAVE principle.[3]

> For any function f, the long-run integral $\int_0^T f(X(t))\,\mathrm{d}t$ is WAVE-equal to any other sum or integral that computes the same signed area, *ignoring edge effects.*

WAVE's name is a (somewhat clumsy) acronym for this principle: *When Averaging, Vilipend Edges.*[4] I hope defining "edge effect" informally as "only affecting the area near times 0 and T" will suffice. If needed, a more formal definition could define it as a difference between the areas that scales as $o(T)$ in the $T \to \infty$ limit, but it remains unclear when that actually holds.

To demonstrate the value of the main WAVE principle, let us derive Little's law for the M/G/1. The key idea, as in the usual formal proof [19], is to look at $\int_0^T q(X(t))\,\mathrm{d}t$ in two ways. Recall our notation that N arrivals happen during $[0, T]$ at times $t_1, \dots, t_N$.

- By definition, $\int_0^T q(X(t))\,\mathrm{d}t = T\mathbf{E}_{\mathsf{time}}[q(X)]$.
- By definition, $\sum_{i=1}^N w(X(t_i-)) = N\mathbf{E}_{\mathsf{arrival}}[w(X)]$.

By the main WAVE principle, thanks the usual trick of "slicing horizontally", these compute the same area modulo edge effects, so

$$T\mathbf{E}_{\mathsf{time}}[q(X)] \approx N\mathbf{E}_{\mathsf{arrival}}[w(X)].$$

Finally, recall that $N \approx \lambda T$ by a law of large numbers principle, which yields Little's law.

A close relative of the main WAVE principle is the *rate conservation law* [21]. One can view it as saying that for any function f,

$$\frac{f(X(T)) - f(X(0))}{T} \approx 0.$$

The power of the rate conservation law comes from expanding the left-hand side as a combination of sums and integrals which constitute all the changes in $f(X)$. This can be done in a systematic way, though it requires some more notation (which we will not define formally). Consider again the M/G/1 from Algorithm 2.1. Each of the labels contributes one way the state can change.

(A) Service yields $\mathbf{E}_{\mathsf{time}}\big[\mathbf{1}(\mathtt{length}(X) \geq 1)\,\partial_{a_1} f(X)\big]$.

(B) Arrivals yield $\lambda\mathbf{E}_{\mathsf{arrival}}\big[f\big(\mathtt{join}(X, [(S, 0)])\big) - f(X)\big]$.

(C) Departures yield $\lambda\mathbf{E}_{\mathsf{departure}}\big[f\big(\mathtt{dropFirst}(X)\big) - f(X)\big]$.[5]

[2]It may be necessary to divide both sides by T, because $N - \lambda T \approx 0$ seems more likely to lead to errors than $N/T - \lambda \approx 0$.

[3]As in the footnote about the law of large numbers, it may be necessary to say that the areas are WAVE-equal only after dividing by T.

[4]"Vilipend" means to regard something as having little value, a fact I learned from a thesaurus while writing this abstract.

[5]Strictly speaking, one needs to show that the average departure rate is λ. This is intuitive, but for a slightly more detailed argument, one can first write this term with a different rate $\lambda_{\mathsf{departure}}$. Applying the rate conservation law to $f(x) = \mathtt{length}(X)$ then implies $\lambda_{\mathsf{departure}} \approx \lambda$.

The rate conservation law says that the sum of these three terms is WAVE-equal to zero. One can carry out the same process for essentially any pseudocode.

The rate conservation law is very powerful. Applying the rate conservation law to $f(x) = w(x)$ yields, after some computation,

$$\mathbf{E}_{\mathsf{time}}[\mathbf{1}(n(X) \geq 1)] \approx \lambda \mathbf{E}[S],$$

a well-known characterization of an M/G/1's load. Applying it to $f(x) = \frac{1}{2}(w(x))^2$ yields, when combined with PASTA, the PK formula for mean work:

$$\mathbf{E}_{\mathsf{time}}[w(X)] \approx \frac{\frac{1}{2}\lambda \mathbf{E}[S^2]}{1 - \lambda \mathbf{E}[S]}.$$

The above derivations are specific instances of the first of two more general recipes:

- To analyze $\mathbf{E}_\ell[(f(X))^p]$, try applying the rate conservation law to $(f(X))^{p+1}$.
- To analyze $\mathbf{E}_\ell[\exp(\theta f(X))]$, try applying the rate conservation law to $\exp(\theta f(X))$.

Variants of these recipes are actually used in current queueing research [15], including my own [28, 29]. For instance, applying the recipe to the M/G/k, one obtains, roughly speaking,

$$\begin{aligned}\mathbf{E}_{\mathsf{time}}[\text{M/G/}k \text{ work}] = \mathbf{E}_{\mathsf{time}}&[\text{M/G/1 work}] \\ &+ \mathbf{E}_{\mathsf{idle}}[\text{M/G/}k \text{ work}],\end{aligned}$$

provided the server speeds are scaled such that the M/G/1 and M/G/k have the same total server speed. While queueing researchers then attempt to explicitly bound terms like $\mathbf{E}_{\mathsf{idle}}$[M/G/$k$ work], one still learns something from just that expression. Whenever a server is idle in the M/G/k, there are $k - 1$ or fewer jobs present, so $\mathbf{E}_{\mathsf{idle}}$[M/G/$k$ work] is, roughly, the "work of at most $k - 1$ jobs".

One can use the main WAVE principle and rate conservation law to prove other helpful principles. Among these are the renewal-reward theorem [12] and its more advanced cousin, the Palm inversion formula [2]. These are especially helpful because they help relate expectations from different perspectives to each other.

2.5 Unresolved Questions

There are, of course, many questions one would need to answer before teaching a course using WAVE. Below are just a few of these questions.

What does a full foundation for WAVE look like? What other definitions, principles, and recipes do we need? Perhaps we need principles that specify what counts as an edge effect. It would be valuable to have recipes for building a mean-field model given a model of one part's dynamics. It may even help to specify a more formal modeling language for writing WAVE pseudocode. Languages for modeling cyber-physical systems [24] could serve as inspiration.

For what audiences is WAVE most appropriate? I plan to teach a small part of an undergraduate stochastic processes course in a style similar to WAVE. Is this too ambitious? In the other direction, could parts of WAVE, such as the idea of expectations from varying perspectives, be valuable for audiences that are even less technical?

What does WAVE lose by sacrificing rigor? Rigorous probability theory exists for a reason. What are the most important dangers to look out for when proceeding non-rigorously? Can we teach students to identify situations that require extra attention to rigor? Considering this question might help decide whether it is worth emphasizing the distinction between WAVE equality $\approx$ and ordinary equality.

ACKNOWLEDGMENTS

Many thanks to Vittoria de Nitto Personè and Y. C. Tay for organizing TeaPACS and inviting me to participate in its 2023 edition. Thanks to Runhan Xie for several helpful discussions about his experience learning stochastic processes. Finally, thanks to Mor Harchol-Balter, my former advisor, for many discussions about teaching and presenting during my PhD years.

This work was done in part while I was an NSF FODSI postdoc at Harvard and MIT, supported by NSF grant nos. DMS-2023528 and DMS-2022448.

References

[1] Nirav Atre, Justine Sherry, Weina Wang, and Daniel S. Berger. 2020. Caching with Delayed Hits. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM 2020)*. ACM, Virtual Event, USA, 495–513. doi:10.1145/3387514.3405883.

[2] François Baccelli and Pierre Brémaud. 2003. *Elements of Queueing Theory: Palm-martingale Calculus and Stochastic Recurrences* (2 ed.). Number 26 in Applications of Mathematics. Springer, Berlin, Germany.

[3] Benjamin Berg. 2022. *A Principled Approach to Parallel Job Scheduling.* Ph. D. Dissertation. Carnegie Mellon University, Pittsburgh, PA.

[4] U. Narayan Bhat. 2008. *An Introduction to Queueing Theory.* Birkhäuser, Boston, MA. doi:10.1007/978-0-8176-4725-4.

[5] Pierre Brémaud. 2020. *Markov Chains: Gibbs Fields, Monte Carlo Simulation and Queues* (2 ed.). Number 31 in Texts in Applied Mathematics. Springer, Cham, Switzerland.

[6] Nathan Bronson, Abutalib Aghayev, Aleksey Charapko, and Timothy Zhu. 2021. Metastable Failures in Distributed Systems. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS 2021)*. ACM, Ann Arbor, MI, 221–227. doi:10.1145/3458336.3465286.

[7] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. 2009. Power-Law Distributions in Empirical Data. *SIAM Rev.* 51, 4 (Nov. 2009), 661–703. doi:10.1137/070710111.

[8] Mark H. A. Davis. 1984. Piecewise-Deterministic Markov Processes: A General Class of Non-Diffusion Stochastic Models. *Journal of the Royal Statistical Society: Series B (Methodological)* 46, 3 (July 1984), 353–376. doi:10.1111/j.2517-6161.1984.tb01308.x.

[9] Jing Dong. 2022. Metastability in Queues. *Queueing Systems* 100, 3-4 (April 2022), 413–415. doi:10.1007/s11134-022-09795-2.

[10] Atilla Eryilmaz and R. Srikant. 2012. Asymptotically Tight Steady-State Queue Length Bounds Implied by Drift Conditions. *Queueing Systems* 72, 3 (Dec. 2012), 311–359. doi:10.1007/s11134-012-9305-y.

[11] Varun Gupta, Mor Harchol-Balter, J. G. Dai, and Bert Zwart. 2010. On the Inapproximability of $M/G/K$: Why Two Moments of Job Size Distribution Are Not Enough. *Queueing Systems* 64, 1 (Jan. 2010), 5–48. doi:10.1007/s11134-009-9133-x.

[12] Mor Harchol-Balter. 2013. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action.* Cambridge University Press, Cambridge, UK.

[13] Per Hokstad. 1978. Approximations for the $M/G/m$ Queue. *Operations Research* 26, 3 (1978), 510–523. doi: `10.1287/opre.27.6.1115`.

[14] Lexiang Huang, Matthew Magnusson, Abishek Bangalore Muralikrishna, Salman Estyak, Rebecca Isaacs, Abutalib Aghayev, Timothy Zhu, and Aleksey Charapko. 2022. Metastable Failures in the Wild. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2022)*. USENIX, Carlsbad, CA, 73–90.

[15] Daniela Hurtado-Lange and Siva Theja Maguluri. 2020. Transform Methods for Heavy-Traffic Analysis. *Stochastic Systems* 10, 4 (Dec. 2020), 275–309. doi:`10.1287/stsy.2019.0056`.

[16] Frank P. Kelly. 2011. *Reversibility and Stochastic Networks* (revised ed.). Cambridge University Press, Cambridge, UK.

[17] John F. C. Kingman. 2009. The First Erlang Century—and the Next. *Queueing Systems* 63, 1 (Nov. 2009), 3. doi: `10.1007/s11134-009-9147-4`.

[18] Rashna Kumar, Sana Asif, Elise Lee, and Fabián E. Bustamante. 2023. Each at Its Own Pace: Third-Party Dependency and Centralization around the World. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 7, 1 (Feb. 2023), 1–29. doi:`10.1145/3579437`.

[19] John D. C. Little. 2011. Little's Law as Viewed on Its 50th Anniversary. *Operations Research* 59, 3 (June 2011), 536–549. doi:`10.1287/opre.1110.0940`.

[20] Michael Mitzenmacher. 2001. The Power of Two Choices in Randomized Load Balancing. *IEEE Transactions on Parallel and Distributed Systems* 12, 10 (Oct. 2001), 1094–1104. doi: `10.1109/71.963420`.

[21] Masakiyo Miyazawa. 1994. Rate Conservation Laws: A Survey. *Queueing Systems* 15, 1 (March 1994), 1–58. doi: `10.1007/BF01189231`.

[22] Jayakrishnan Nair, Adam Wierman, and Bert Zwart. 2022. *The Fundamentals of Heavy Tails: Properties, Emergence, and Estimation*. Number 53 in Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge, UK. doi:`10.1017/9781009053730`.

[23] James R. Norris. 1997. *Markov Chains*. Cambridge University Press, Cambridge, UK. doi:`10.1017/CBO9780511810633`.

[24] André Platzer. 2018. *Logical Foundations of Cyber-Physical Systems*. Springer, Cham, Switzerland. doi: `10.1007/978-3-319-63588-0`.

[25] Sheldon M. Ross. 2014. *Introduction to Probability Models* (11 ed.). Elsevier, Amsterdam, The Netherlands.

[26] Hiroshi Sasaki, Fang-Hsiang Su, Teruo Tanimoto, and Simha Sethumadhavan. 2017. Why Do Programs Have Heavy Tails?. In *2017 IEEE International Symposium on Workload Characterization (IISWC 2017)*. IEEE, Seattle, WA, 135–145. doi:`10.1109/IISWC.2017.8167771`.

[27] Alan Scheller-Wolf and Rein Vesilo. 2006. Structural Interpretation and Derivation of Necessary and Sufficient Conditions for Delay Moments in FIFO Multiserver Queues. *Queueing Systems* 54, 3 (Nov. 2006), 221–232. doi:`10.1007/s11134-006-0068-1`.

[28] Ziv Scully. 2022. *A New Toolbox for Scheduling Theory*. Ph. D. Dissertation. Carnegie Mellon University, Pittsburgh, PA.

[29] Ziv Scully, Isaac Grosof, and Mor Harchol-Balter. 2020. The Gittins Policy Is Nearly Optimal in the M/G/k under Extremely General Conditions. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 3, Article 43 (Nov. 2020), 29 pages. doi:`10.1145/3428328`.

[30] Y. C. Tay. 2014. *Analytical Performance Modeling for Computer Systems* (2 ed.). Morgan & Claypool, San Rafael, CA.

[31] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. 2020. Borg: The next Generation. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys 2020)*. ACM, Heraklion Greece, 1–14. doi:`10.1145/3342195.3387517`.

[32] Ronald W. Wolff. 1982. Poisson Arrivals See Time Averages. *Operations Research* 30, 2 (1982), 223–231.