Association for Computing Machinery
Advancing Computing as a Science & Profession

Ubiquity, an ACM publication
August 2016

Rethinking Randomness

An Interview with Jeff Buzen, Part II

by Peter J. Denning

Editor's Introduction

In Part 1, Jeff Buzen discussed the basic principles of his new approach to randomness, which is the topic of his book Rethinking Randomness. He continues here with a more detailed discussion of models that have been used successfully to predict the performance of systems ranging from early time sharing computers to modern web servers.

Peter J. Denning
Editor in Chief

Rethinking Randomness

An Interview with Jeff Buzen, Part II

by Peter J. Denning

Peter Denning: Tell us a bit more about the models that are actually used to evaluate computer system performance. What do they look like, and what types of problems are they used to solve?

Jeff Buzen: Let's begin with Figure 5, which is a more detailed version of the diagram in Figure 2. It depicts a system and a group of *N* active users who transmit requests for processing from their desktops, laptops and hand held devices.

The box labeled System can represent anything from a single e-commerce server to an entire server farm of the type deployed by Google or Amazon to process incoming queries. In principle, the system box can also contain components of the network that connects these servers to end users. Thus, the idea of a system is both flexible and general.

To keep things simple for this discussion, we'll just combine all the processing that takes place within the system into a single value. This value becomes the "service time" at the system's processor (represented by a circle). The system also contains a queue (represented by a rectangle) that holds requests submitted by users and waiting for service by the processor. In this simplified model, only one request can be processed at a time. All others must wait in the queue until their turn comes.

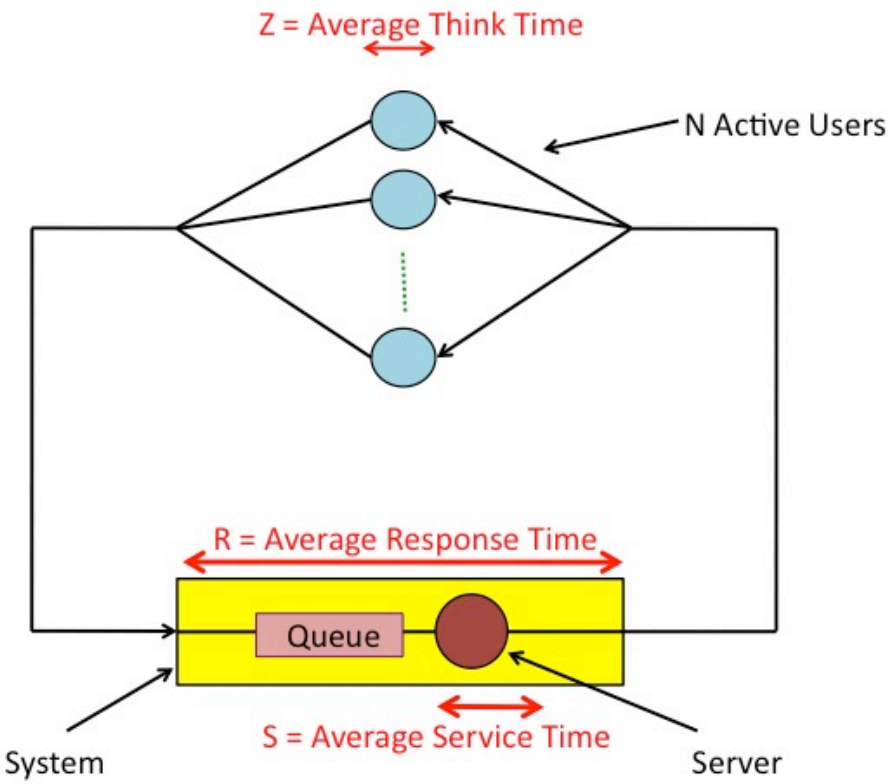

Figure 5 – Model of a Simple System

Essentially, we want to understand how response time varies with the number of active users who are accessing this system. In particular, we'd like to know how many active users the system can support before response time becomes unacceptably long … or buffer overflow causes the entire system or website to crash.

To begin, let's consider the relationship between *N*, the number of active users, and *U*, the utilization of the server. We need two other parameters at this stage of the analysis. We've already seen one of these before: *S*, the average processing time per request. The second is *Z*, the average think time per interaction. Essentially, *Z* represents the time it takes the user to examine the results that were obtained from the previous request, think about what to do next, and then send off the next request.

PD: You are saying that you can calculate the response time just knowing two external parameters (*N* and *Z*) and one internal measure (*S*)? Don't you need to know the layout of the network and the way the jobs flow through?

JB: No, those details don't matter. Let me show you. If only one user is accessing the system, each complete "think-wait" cycle takes an average of $S + Z$ seconds. Since the system is busy for an average of S seconds per cycle, system utilization is clearly equal to $S/(S + Z)$.

As more users access the system, utilization will obviously increase. Under "best case" assumptions, users will be perfectly synchronized and never interfere with each other: while one user is waiting for a request to be processed by the system, all others will be thinking about their next requests. Thus, if there are two perfectly synchronized users, utilization will double to $2S/(S + Z)$. In general, if there are n perfectly synchronized users, utilization will grow to $nS/(S + Z)$.

It's clear that perfect synchronization cannot go on forever. Utilization will necessarily reach 100 percent once the number of perfectly synchronized users exceeds $(S + Z)/S$. Beyond this point, a queue will inevitably build up at the system and utilization will level off. This general behavior is shown in Figure 6.

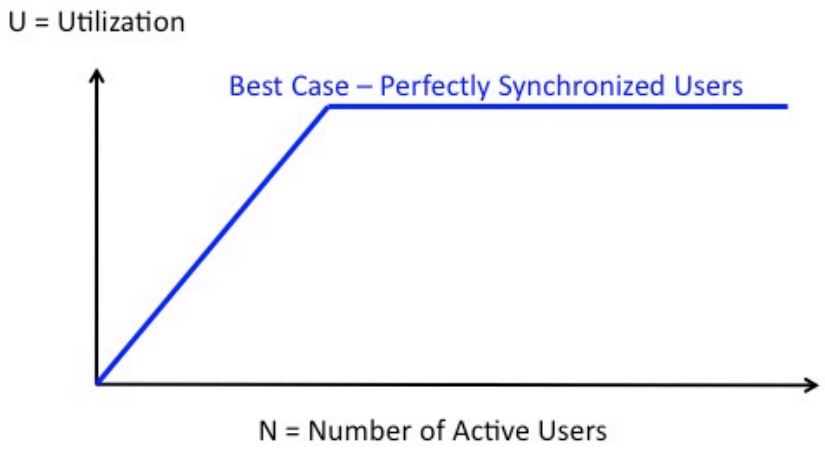

Figure 6 – "Best Case Utilization Curve

PD: Can you do any better than an upper bound? What would you actually expect to see in practice?

JB: This is where randomness and variability come in. These factors make perfect synchronization impossible, even when the number of active users is relatively small.

In traditional queueing models, this type of randomness is represented by assuming that each individual think time and each individual service time can be regarded as a sample drawn from a probability distribution. The exact forms of the think time distribution and the service time distribution are often assumed to be negative exponential.

It's not necessary for us to look into the equations that define the negative exponential distribution. The main point for our purposes is that "realistic" utilization curves such as the one shown in red in Figure 7 can be derived if you assume exponential service times.

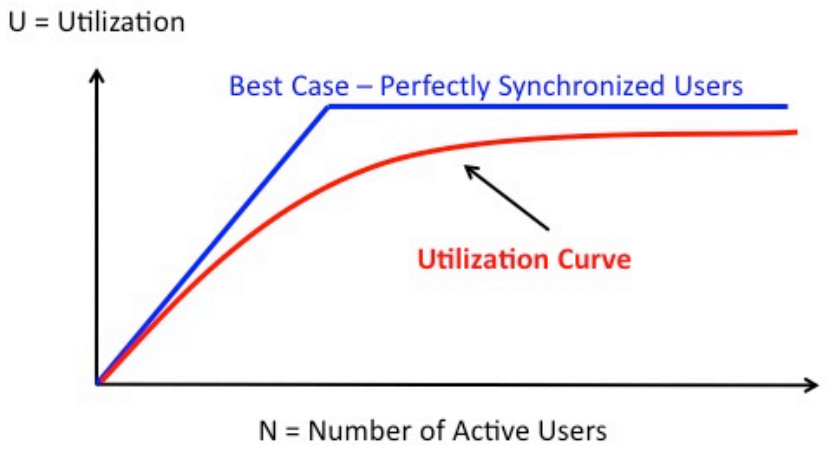

Figure 7 – Realistic Utilization Curve

Utilization curves based on the negative exponential distribution have been used successfully to predict computer system performance for more than half a century. The first validation of the

model in Figure 5 was carried out by Allan Scherr as part of his 1965 Ph.D. dissertation at MIT. Many successful validations of similar (but more complex) models have been carried out since.

As I mentioned earlier, I've always found these successful validations a bit puzzling since the assumptions that traditional models require seem unlikely to be satisfied in practice.

These concerns led me to develop a new framework for thinking about uncertainty, unpredictability, and randomness. As I've already mentioned, the key idea is to base the models on observable trajectories rather than abstract stochastic processes.

The next challenge is to introduce assumptions about variability and randomness that can be expressed in terms of observable properties of trajectories. In general, these assumptions concern the rates at which state-to-state transitions take place. These transition rates are important because they determine the overall flows into and out of individual states. It's critically important to balance such flows during an analysis.

In this particular model, my assumptions concern the rate at which new requests arrive at the system, and the rate at which requests currently in the system are completed. I'm able to derive utilization curves that have exactly the same mathematical form as those derived through traditional queueing models by making reasonable assumptions about how these transition rates vary as the number of requests currently in the system changes. Anyone with a strong background in high school algebra should be able to follow the analysis. See *Rethinking Randomness* for details.

PD: How do you use the utilization you just calculated to get the response time?

JB: The response time law expresses R, the average response time of any system, as a function of the number of users N, the average think time Z, and the overall throughput rate X.

$$R = N/X - Z$$

Response Time Law

This law has a simple explanation. Consider a single user who thinks for an average of Z seconds, waits an average of R seconds for the system to respond, and then repeats these think-wait cycles over and over again. On average, the amount of time this user requires to complete one entire cycle is $Z + R$. This implies that the average number of requests a typical user completes per second is $1/(Z + R)$. Note that R represents total response time, including

service time S as well as the queueing delays that arise because the *N* users in the model are not perfectly synchronized.

Since there are a total of *N* users, the overall throughput rate is $N/(Z + R)$ requests per second. But the overall throughput rate is, by definition, equal to *X*. Thus $X = N/(Z + R)$. Just solve this for *R*.

Note that this law applies not only to the simple system shown in Figure 5, but to all systems, regardless of the number of servers, queues and other components they may contain. It does not depend on any assumptions about probability distributions or their long term limiting values.

By combining the response time law with the utilization law I mentioned earlier, it's possible to generate a graph of the type shown in Figure 8. Graphs such as these are used routinely to estimate the number of users a system can support before response time gets too high.

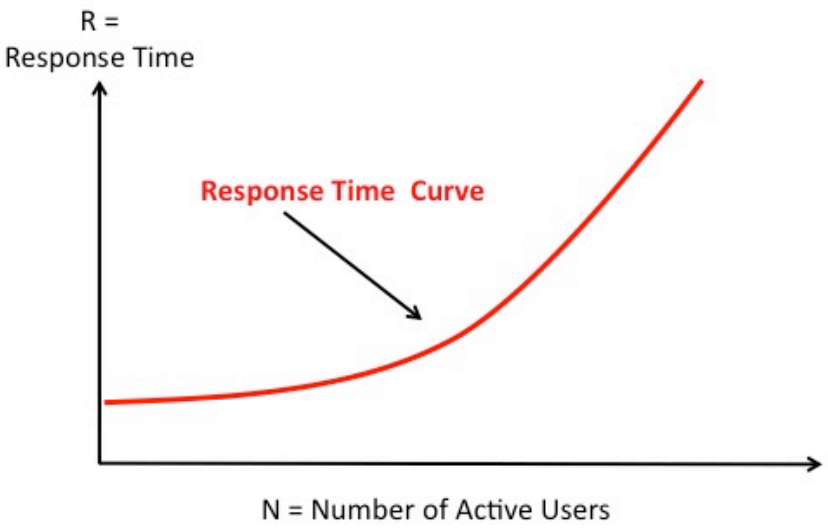

Figure 8 – Response Time vs Number of Users

There are of course many options to consider once response time exceeds that ceiling. There may be a bottleneck somewhere in the system that can be eliminated by upgrading one minor component or rewriting a small section of code. On the other hand, it may be necessary to upgrade the entire system to one with a faster CPU, more memory, and SSDs rather than rotating disks.

Of course, upgrades don't necessarily have to involve physical systems anymore. Many apps run on virtual servers that reside in the cloud. However, virtual doesn't mean free. Upgrades still cost real money, and it's still necessary to understand both the costs and the benefits of any proposed upgrade. The models we've just been discussing here play an important role in this process.

PD: That's very interesting that you can get such a powerful and widely applicable formula for response time from so few assumptions. This illustrates a point you have been making all along about why your new framework for thinking about randomness has great value. Can you summarize what is different between your approach to randomness and the traditional approach?

JB: The two approaches begin from very different starting points. Traditional stochastic models start with the idea that the state of the system at any instant is uncertain and can be represented by a probability distribution. The objective of the analysis is to derive an expression for this distribution, based on assumptions about other probability distributions that characterize the step-by-step behavior of the system.

My starting point is a trajectory: a time stamped sequence of states that can, in principle, be obtained by observing a real system for a finite interval of time. Intuitively speaking, trajectories are generated when workloads are processed by systems. I present a more formal definition, based on the computer science notion of finite state machines in Chapter 3 of *Rethinking Randomness*.

Note that the state of the system at any instant depends on the workload that the system is processing. The randomness of the system state is a consequence of the uncertainty about the workload that generated a trajectory. In a trajectory-based model, I don't need to know the detailed properties of workloads. In effect, I get around the uncertainty by ignoring it and focusing on how it affects the average time the system is in each state.

In other words, instead of focusing on the state of the system at a given instant, trajectory-based models deal with interval-wide quantities such as utilization, throughput and average

response time. Note that interval-wide quantities are not limited to averages. They also include the proportion of time that queue length is equal to 0, 1, 2 and so on.

These proportions are closely linked to the queue length distributions in traditional stochastic models. However, their derivation does not require probabilistic assumptions. Their derivations depend instead on assumptions about the rates at which various transitions take place. These rate-based assumptions represent a different way to think about and characterize randomness.

The bottom line is that trajectory-based models are useful, easy to understand, and directly applicable to many types of systems—including computer systems, communication networks, and other systems that are studied in engineering and in the physical, biological and social sciences.

PD: Let's finish up on a lighter side, applying your ideas of randomness to weather reports, a modern everyday use of probability. We are told in the morning that the rain chance for the day is 65 percent. We look at weather.com and can find hourly predictions like 40 percent at 8am, 80 percent at noon, and 50 percent at 4pm. What do these numbers mean? Are they operational predictions? Do they mean that on all days with similar weather conditions 65 percent rained? Or that it will rain at 65 percent of maximum possible rain intensity? Or that a bookie would give you 65 percent odds that rain will fall?

JB: I've noticed that weather forecasters on TV are using probability in at least two different ways. When they say that the forecast for tomorrow afternoon is for scattered showers and that viewers have a 40 percent chance of getting wet, it's clear from the weather map they show us that they expect 40 percent of the viewing area to be hit by rain. They simply don't know where that 40 percent will be. This is a forecast that can be verified the following day by simply measuring where rain has actually fallen. This aligns perfectly with the view of probability and chance that I've presented here.

On the other hand, that same forecaster may tell us a few days later that there is a hurricane headed our way with a 40 percent chance of hitting us directly. This is an "all or nothing" forecast that cannot be verified the following day in the same manner. I agree with your suggestion to think of probabilities such as these in terms of the proportion of time a hurricane would hit us under a set of equally likely initial conditions. I have a non-traditional discussion of the law of large numbers in the final chapter of *Rethinking Randomness* that presents an argument along these lines.

Whether or not a bookie would give you the same odds for both forecasts is something I really can't say.

Suggested Readings

J.P. Buzen, *Rethinking Randomness: A New Foundation for Stochastic Modeling*, CreateSpace, 2015.

J.P. Buzen, Fundamental Laws of Computer System Performance, *SIGMETRICS '76: Proc. 1976 ACM SIGMETRICS Conf. on Computer Performance Modeling, Measurement and Evaluation,* April 1976, 200-210.

P.J. Denning and J.P. Buzen, Operational Analysis of Queueing Network Models, *ACM Computing Surveys* 10, 3 (Sept. 1978), 225-261.

R. Suri, G.W. Dielh, S. de Treville and M.J. Tomsicek, From CAN-Q to MPX: Evolution of Queueing Software for Manufacturing, *INFORMS Interfaces* 25, 5 (Oct. 1995), 128-150.

About the Author

Peter J. Denning (pjd@nps.edu) is Distinguished Professor of Computer Science and Director of the Cebrowski Institute for information innovation at the Naval Postgraduate School in Monterey, California, is Editor of ACM *Ubiquity*, and is a past president of ACM. The author's views expressed here are not necessarily those of his employer or the U.S. federal government.

DOI: 10.1145/2986331

