8.1 Introduction

This chapter introduces the PDQ (Pretty Damn Quick) queueing analyzer and explains how to use it. We begin with some guidelines on how to build performance models and then move on to the specifics of the PDQ library in Perl. Finally, we present the actual PDQ codes for the examples discussed in Chaps. 4 and 5. Other PDQ model codes are embedded in the chapters of Part II. Instructions for installing PDQ and creating the corresponding Perl module can be found in Appendix D.

8.2 How to Build PDQ Circuits

Consistent with the notion of circuits presented in Chap. 5, every PDQ program should have a defined set of inputs and a set of outputs. The inputs are performance metrics such as traffic rates, active user population, and service rates. These come from the data you have collected from the system you are analyzing or estimates if no data exists. The role of PDQ is to provide a set of performance metrics, such as utilizations, queue lengths, and residence times, as outputs.

8.3 Inputs and Outputs

As an example of this procedure, recall the response time formula in (4.38) for an $M/M/1$ queue labeled with its respective input and output parameters:

$$R \leftarrow \frac{D \text{ (input)}}{1 - \lambda \text{ (input)} \times D \text{ (input)}}.$$  

This is also a model; an equational model but a model, nonetheless. The corresponding queueing circuit is shown in Fig. 8.1. Equation (8.1) is already
contained in PDQ, so you will never have to write the code for this equation explicitly. To solve (8.1) manually, however, we need the appropriate inputs. These are the parameters on the right side of the formula, viz. the arrival rate $\lambda$ and the service demand $D$. Those inputs are then used to calculate the output $R$ on the left side of (8.1). That is precisely what PDQ does algorithmically.

![Fig. 8.1. Simple M/M/1 queueing circuit with conventional inputs: the arrival rate $\lambda$, the mean number of visits $V$, the mean service time $S$, or the service demand $D = VS$. The typical outputs are the waiting time $W$, the residence time $R$ in (8.1), and the mean queue length $Q = \lambda R$](image)

Of course, things are rarely this simple in real life. We may not have a direct measurement of the arrival rate $\lambda$ or the average service demand $D$, so we have to resort to deducing it from other information we do have. For example, we may have to calculate the arrival rate by counting the number of arrivals $A$ during the measurement period $T$ and using the relation $\lambda = A/T$ from Chap. 4. Combining this calculated $\lambda$ with the measured utilization of the server, we can use Little’s law $\rho = \lambda D$ to determine the input service demands as $D = \rho/\lambda$.

Building PDQ models is merely an extension of this same process. We try to limit the number of input parameters required for the model and let PDQ do the work of computing numerous output metrics. Sometimes, the process of building a PDQ model can surprise you by telling you what parameters need to be measured as inputs that you had not thought of previously. An advantage of PDQ is that you do not have to construct the code for all the performance equations presented in Chaps. 4 and 5. Another advantage of PDQ is that the outputs can be computed for very complex queueing circuits with multiple workloads that would otherwise be debilitating, if not impossible, to carry out by hand.

### 8.3.1 Setting Up PDQ

PDQ is a queueing circuit solver, not a simulator. As part of its suite of solution methods, PDQ incorporates the MVA algorithm discussed in Chap. 5. The purpose is to enable the user to build queueing circuit representations of