Having two bathrooms ruined the capacity to cooperate.
Margaret Mead

The title of this chapter implies that there is more here than M/G/C queues. The straightforward extension of Chapter 4 allows one to have \( C \) identical servers serving up to \( C \) customers independently and simultaneously. But when we set up those equations we find that they apply to a more general class of systems where the active customers can actually interfere with each other while being served. This can be used as a basis for studying clusters of workstations that must share resources such as a communications channel or central disc. We call such a system a \textit{generalized M/G/C//N queue}, where \( N \) is the total number of customers in the system. Interestingly enough, when \( N = C \) then the steady-state solution is the same as for the single class \textit{Jackson network} [JACKSON63], but when \( N > C \) the well-known \textit{product form} solutions are no longer valid, and one must resort to the matrix techniques described here.

### 6.1 Introduction

In previous chapters when dealing with nonexponential distributions, we always assumed that only one customer was active at a time at \( S_1 \). We did look at multiple servers, but only if \( S_1 \) was exponential, introducing the idea of a load-dependent server (Sections 2.1.5 and 5.4). In doing this, it was not necessary to distinguish between:

1. A subsystem containing a single server that works twice as fast on one customer when a second one is present;
2. A subsystem that has two active servers, one for each customer.

In fact, the only way the two cases can be distinguished is by marking the customers so as to tell if they left in the same order in which they arrived. This has become of interest in recent years, and is called the \textit{resequencing problem}. LAQT has been used successfully in analyzing the departure process of an M/M/C queue where customers must leave in the same order in which they arrived [DING91]. Because we have made our customers indistinguishable, we have not bothered to consider this at all, nor can we consider it here without expanding our state space.
6.1 Introduction

We cannot get away so easily when dealing with nonexponential servers. Case 1 has had few realistic applications, but it can be used in studying queues with, for instance, **discouraged arrivals** or **restricted processor sharing**. It is modeled by multiplying the completion rate matrix $M$ by a constant factor when a second customer arrives, leaving the dimension and internal state description of $S_1$ otherwise unchanged. This turns out to be formally identical to the description of ME/M/C//N loops in Section 4.4, except that the load-dependence factor depends on the number of customers at $S_1$ instead of the number at $S_2$. We discuss this further when we look at processor sharing.

The second case is much more complicated. When a second customer arrives, as always, he begins service by going to phase $i$ with probability $p_i$, but the first customer is already in service and is at some other phase. Furthermore, when one of the customers leaves, the other customer is still in service, in some phase determined by the system’s past. Put differently, a departing customer does not leave behind the empty state. We must therefore set up a formalism that keeps track of where both customers are. This is normally done by building a **direct product space**, the most common convention being the **Kronecker product**. However, if the service times for the two customers are identically distributed and they are not marked, one can use a **Reduced-Product (RP) space**. The direct product spaces have dimension $m^n$ for $n$ iid customers, but the RP space has dimension $D_{RP}(m, C) = \binom{C + m - 1}{C}$.

This amounts to a reduction of dimensions by a factor that approaches $C!$.

We have to introduce new symbols and concepts here, hence it is best to start with the simplest extension possible. Therefore, in the following section we set up the formalism, and find the steady-state solution of a system where $S_1$ has exactly two identical ME servers (i.e., the M/ME/2//N loop). In doing this, we have selected a three-phase ME server as an example. In Section 6.3 we extend this to $C$ servers, for by then it will be easier for the reader to follow the notation.

After that, we show that the formulas are actually applicable to a more general class of systems, which we call “generalized M/G/C//N systems.” With little more than a change of notation and a slight generalization of some parameters, we show that we are suddenly dealing with a network of queues. When $N \leq C$, our generalized network is equivalent to the single-class **Jackson network**, and we spend some time discussing the connection. We then extend the model further to allow $S_2$ to be a load-dependent server, as we did in Section 5.4. This is potentially an important extension, because it is the correct treatment of **timesharing systems** with **population-size constraints** (i.e., when $N > C$).

When doing all this, we find that the equations are still algebraically manipulable but too complex to reduce to simple formulas. Thus we describe in

---

*This has been done recently by Feng Zhang [Zhang07] in examining computer systems where a restricted number of jobs can share the CPU*