CHAPTER 1

Queuing Networks as Applications Models

In this chapter: characteristics of enterprise applications and their key performance indicators; what is application sizing and tuning; why queuing models are representative abstractions of enterprise applications; what is transaction response time and transaction profile.

1.1. ENTERPRISE APPLICATIONS—WHAT DO THEY HAVE IN COMMON?

Enterprise applications have a number of characteristics essential from a performance engineering perspective.

1. Enterprise applications support vital corporate business functions, and their performance is critical for successful execution of business tasks. Consider as an example the failure of a company to deliver a timely quarterly earnings report to its shareholders and Wall Street due to a bottleneck in one of the system servers, which had crashed and brought the application down.

Solving Enterprise Applications Performance Puzzles: Queuing Models to the Rescue, First Edition. Leonid Grinshpan.

2. Corporations inherently tend to grow by expanding their customer base, opening new divisions, releasing new products, as well as engaging in restructuring, mergers, and acquisitions. Business dynamics directly affects a number of application users, as well as the volume and structure of data loaded into databases. That means that tuning and sizing must be organic and indispensable components of the application life cycle, ensuring its adaptation to an ever-changing environment.
3. Each company is unique in terms of operational practice, customer base, product nomenclature, cost structure, and other aspects of business logistics; as such, enterprise applications cannot be deployed right after being purchased as they must undergo broad customization and be tested and tuned for performance before being released in production.
4. The typical enterprise application architecture represents server farms with users connected to the system from geographically distributed offices over corporate and virtual private networks.
5. Enterprise applications deal with much larger and complex data per a user's request as opposed to Internet applications because they sift through megabytes and even terabytes of business records and often implement massive online analytical processing (OLAP) in order to deliver business data rendered as reports, tables, sophisticated forms, and templates.
6. The number of enterprise application users is significantly lower than that of Internet application users since their user communities are limited to corporation business departments. That number can still be quite large, reaching thousands of users, but it never even comes close to millions as in the case of Internet applications.
7. End users work with enterprise applications not only through their browsers, as with Internet applications, but also through a variety of front-end programs (for example, Excel or PowerPoint, as well as interface programs specifically designed for different business tasks). Often front-end programs do large processing of information that is delivered from servers before making it available to the users.
8. A significant factor influencing the workload of enterprise applications is the rate of the requests submitted by the users—a

number of requests per given time interval, usually per one work hour. Pacing defines an intensity of requests from the users and by the same token utilization of system resources.

Enterprise applications have a client-server architecture [1.1], where the user (client) works with a client program; through this program the user demands a service from a server program by sending a request over the corporate network (Fig. 1.1). The client program resides on the user's computer; the server program is hosted on a server.

Figure 1.1 represents a two-tier implementation of client-server architecture. Today's complex enterprise applications are hosted on multi-tier platforms; the most common is the three-tier platform (Fig. 1.2). The functional logic of the application is performed by software hosted on a middle tier; data are stored in the database tier.

Three-tier architecture has a fundamental performance advantage over the two-tier structure because it is *scalable*; that means it can support more users and a higher intensity of requests by increasing the number of servers and their capacity on a functional tier.

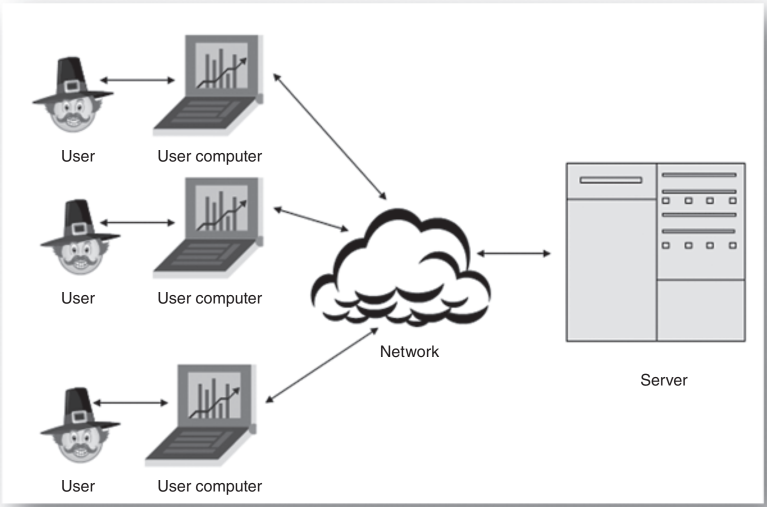

Figure 1.1. Client-server architecture.

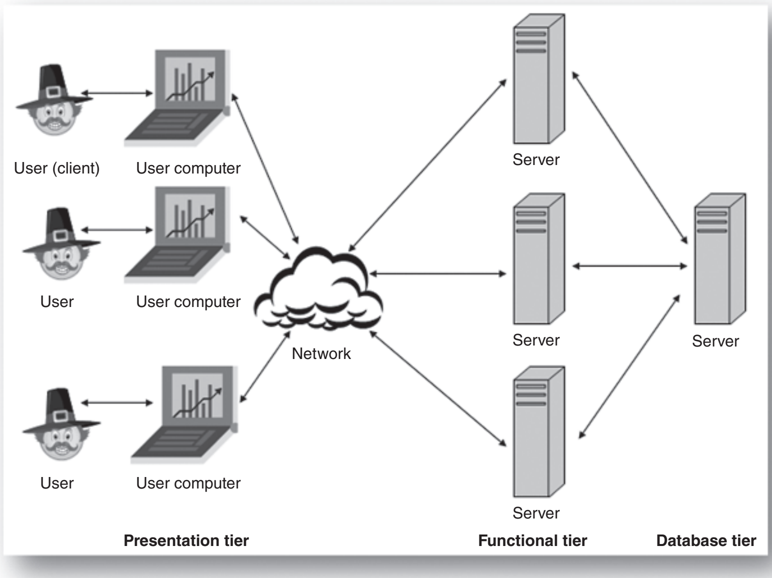

Figure 1.2. Three-tier architecture.

The next level of scalability can be achieved by separation of Web servers from the functional tier (Fig. 1.3). This is possible for enterprise applications where the presentation tier communicates to the system over the Internet. The Web server tier is scalable as well and can comprise multiple Web servers.

Ultimate scalability is delivered by network-like architecture where each functional service of the enterprise application is hosted on dedicated computers. This architecture is illustrated in Fig. 1.4, where different hardware servers host services like data integration, business logic, financial consolidation, report printing, data import/export, data storage, etc. The architecture in Fig. 1.4 can support practically unlimited growth in the number of business users and complexity or volume of data by deploying additional servers that host different services.

In a network of servers, a request from a user sequentially visits different servers and is processed on each one for some time until it returns to a user delivering business data (rendered, for example, as a report or spreadsheet). We can imagine a request from a user as a car traveling across network of highways with tollbooths. A tollboth is

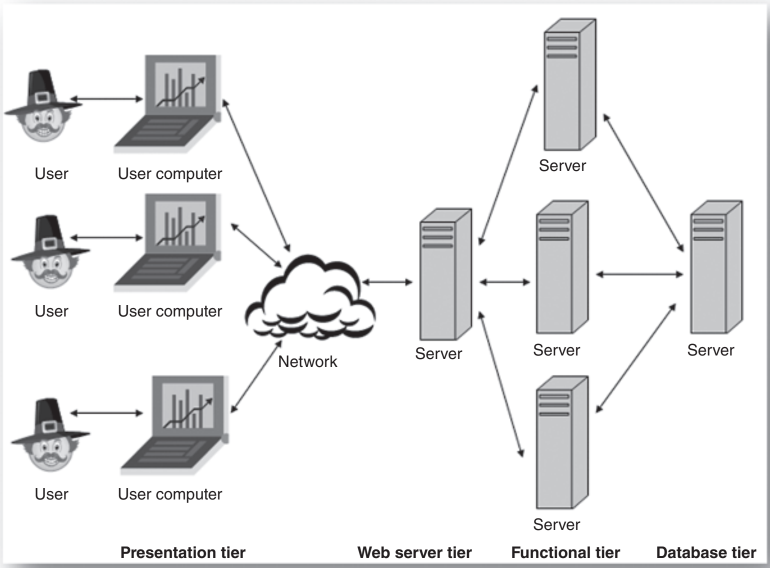

Figure 1.3. Four-tier architecture.

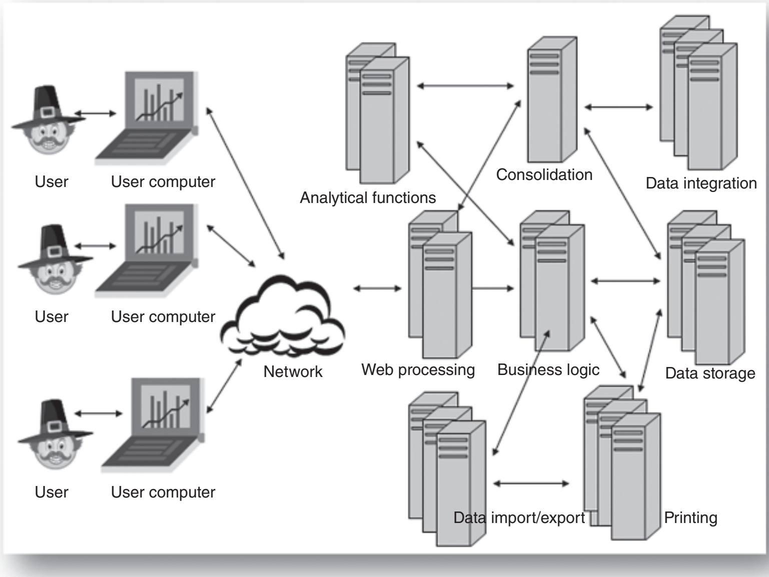

Figure 1.4. Network of servers.

serving cars by accepting payments and in that respect it can be compared to a hardware server working on users' requests.

A user's request to an application can be imagined as a car and a highway's tollbooth as a hardware server. A car's travel on highway with tollbooths is a representative metaphor of a request served in hardware servers.

1.2. KEY PERFORMANCE INDICATOR—TRANSACTION TIME

The users interact with the application by sending requests to execute particular business functions, for example:

- Generate a report on the sales of computers in the northeast region of the United States in the year 2000
- Consolidate financial data on bonuses paid to employees in the first quarter of the current year
- Load into the database data on all the items sold in January across company stores in New York State

A user's request forces the application to perform a particular amount of work to generate a response. This unit of work comprises the transaction; the time needed to complete the work is called *transaction response time* or *transaction time*. Transaction time is the most important characteristic of system performance from the user's perspective. If the users feel comfortable with the application's agility to process requests, then no additional efforts have to be made to speed up the application, no matter how long transaction times are. The acceptability of a transaction time by a business deviates greatly because it depends solely on the business users' conscious and unconscious anticipation of system agility. Business users in general have a fairly objective stance on transaction time. For example, they expect to get a reply in a matter of seconds when requesting the number of company stores located in New York City but are still pretty comfortable with more than a 1-minute processing of their request on the number of DVD players of different models sold in all the New York stores in January–March of the current year. The difference between two transactions is that the first one returns a number saved in a database, while the second one is actually an ad hoc request requiring analytical calculations on the fly.

On the other hand, if the system appears to the users to be slow and getting even slower and slower as more users log in, then the performance analyst has to use the tools of his trade to fix the issues. Application sizing and the tuning goal is to create an illusion for each user that he/she is the only person working with a system no matter how many other users are actively interacting with the application.

Application performance can be qualified as satisfactory or not only by the application's users; there is no formal metric to make such a distinction. If the users are comfortable with transaction times no efforts should be wasted to improve them (as a quest for the best usually turns out an enemy of the good because of the associated cost).

Applications can slow down not only when the number of active users increases, but also, in some instances, when additional data are loaded into database. Data load is a routine activity for enterprise applications, and it is scheduled to happen over particular time intervals. Depending on the application specifics and a company's logistics, a data load might be carried out every hour, every business day, once per week, once per month, or a data load schedule can be flexible. In several cases, loading new data makes the database larger and information processing more complex; all that has an impact on transaction times. For example, a transaction generating a report on sales volume in the current month might take an acceptable time for the first 10 days of the month, but after that it will take longer and longer with every following day if the data load occurs nightly.

Transaction time degradation can happen not only when the number of users or volume of data exceeds some thresholds, but also after the application is up and running for a particular period. A period can be as short as hours or as long as days or weeks. That is an indication that some system resources are not released after transactions are completed and there is a scarcity of remaining resources to allocate to new transactions. This phenomenon is called "resource leak" and usually has its roots in programming defects.

The natural growth of a business leads to an increase in rate (intensity) of requests from users as well as in data volume and data complexity; this makes periodic performance tuning and sizing an integral and mandatory part of the application life cycle.

Indicators of performance problems:

- ✓ Transaction response time is getting longer as more users are actively working with the system
- ✓ Transaction response time is getting longer as more data are loaded into system databases
- ✓ Transaction response time is getting longer over time even for the same number of users or data volume; that is a sign of "resource leak"

1.3. WHAT IS APPLICATION TUNING AND SIZING?

Application tuning is a course of action aimed at finding and fixing the bottlenecks in deployed systems for a given workload, data volume, and system architecture.

Application sizing takes place for planned applications as well as for existing production applications when they have to be scaled to accommodate a growth in the number of users and volume of data. Sizing delivers estimates of hardware architecture that ensures the quality of the requested service under an anticipated workload.

The sizing of the planned-for deployment systems as well as the tuning of the deployed systems are actually processes of removing limiting boundaries. There are two kinds of limits in enterprise applications:

1. Hardware limits: number of servers, number of CPUs per server, CPU speed, I/O speed, network bandwidth, and similar specifications of other hardware components
2. Software limits: parameter settings that define "throughput" of the logical "pipelines" inside the application (for example, number of threads, number of database connections, number of Web server connections, Java virtual machine memory size, etc.)

Performance engineering is the profession and obsession of improving an application's agility and users' satisfaction by removing software and hardware limitations; by removing boundaries it maximizes system throughput in a very cost-effective way. System tuning and sizing can be compared to processing plant optimization when different means and tools are applied in concert to increase a plant's output.

The complexity of enterprise applications makes their capacity planning and tuning projects effortful and demanding and even more so when they have to be executed in a limited time. Queuing network models clarify and interpret happenings in applications, framing their performance troubleshooting and sizing as logically organized processes that can be interpreted formally and therefore carried out successfully and efficiently.

1.4. QUEUING MODELS OF ENTERPRISE APPLICATION

In dealing with such intricate objects as enterprise applications we have to see the forest, not the trees. That is exactly what models help us to do: they shield our brains from numerous nonimportant and distracting details and allow us to concentrate on the fundamental processes in applications.

Models are capable of factoring in system architecture, the intensity of users' requests, processing times on different servers, as well as the parameters of hardware and a user's workload that are meaningful for performance analysis. Models also can assess the effects and the limitations of software parameters such as the number of threads, size of memory, connections to system resources, etc. At the same time models help to abstract from application specifics that might be substantial from a functionality perspective but irrelevant to sizing and tuning.

Why we are going to use **queuing** models to understand and solve application performance puzzles? The answer is that any system that provides services to the users has the users' requests waiting in queues if the speed of a service is slower than the rate of incoming requests. Consider a grocery store: people are waiting for checkout during peak hours. Another example: have you ever heard while dialing a bank: "All agents are busy serving other customers, please stay on the line and someone will answer your call in the order it was received"? In that case your call is put in a queue. If there is no waiting space in a queue you might hear: "All lines are busy now, try again later."

A few examples of the systems that can be presented by queuing models:

1. Bridge toll. Cars are requests; booths are servers.
2. Grocery store. Visitors are requests; counters are servers.
3. Cellular phone network. Phone calls are requests; phone towers are servers.
4. Internet. Click on a link initiates a request; networks and computers are servers.

Enterprise applications, like other systems serving multiple concurrent requests, have to manage different queues; they put incoming requests into waiting queues if particular services are busy processing previous requests. To size and tune applications we have to understand why queues are building up and find out how to minimize them. Queuing models are time-proven logical abstractions of real systems, and they clarify causes and consequences of queues on performance of multiuser systems [1.2, 1.3, 1.4]. Queuing models help us to understand event and processes relevant for sizing and tuning of computer systems: competition for resources among concurrent requests (CPU, memory, I/O, software threads, database connections, etc.) waiting in queues when resources are not available, the impact of the waits on transaction response times, and so on.

Queuing models represent applications by employing two constructs: transactions and nodes. A user's request initiates a transaction that navigates across a network of nodes and spends some time in each node receiving a service. Various publications on queuing models do not distinguish between the terms "request" and "transaction," assuming they mean the same thing. In such a case, the previous sentence can be rephrased: "A request navigates across a network of nodes and spends some time in each node receiving a service." We predominantly differentiate the two terms, but when we talk about queuing systems and queuing networks as mathematical objects, but not as application models, we use term "request" (this mostly relates to Chapter 2).

In this book we are dealing with two types of nodes. One type consists of two entities: queue and processing units (Fig. 1.5a).

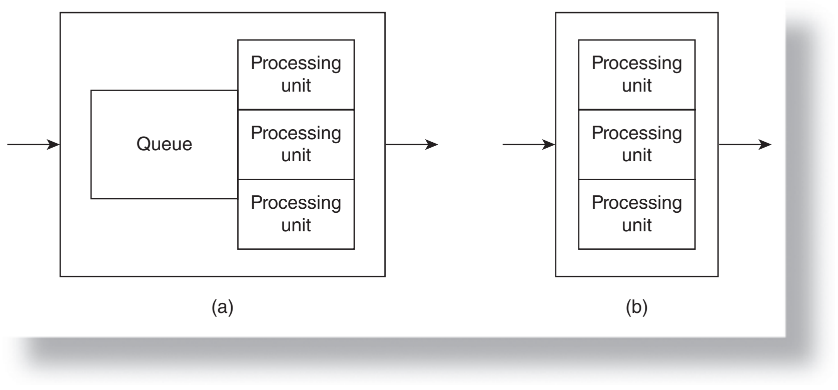

Figure 1.5. (a) A node with queue and processing units; (b) a node with processing units.

Processing units serve incoming transactions; if they are all busy, transactions will wait in the node's queue. The second type does not have a waiting queue but only processing units (Fig. 1.5b).

With a little imagination we can envision a transaction initiated by a user as a physical object visiting different hardware servers. A symbolic metaphor for a transaction is its representation as a car traveling on a highway with tollbooths. A tollbooth, in turn, is a metaphor for a hardware server.

Figure 1.6 depicts a relationship between an application and its queuing model. This is just one of many possible models of the same computer system; the model can represent a system on the different levels of abstraction. The model in Fig. 1.6 embodies system architecture; it has three nodes corresponding to the users, network, and hardware server.

Below are the relationships between the components of a real system and the components of its model:

Component of Application	Matching Component in Queuing Model
Users and their computers	Node "users"
Network	Node "network"
Server	Node "server"
Transactions initiated by users	Cars

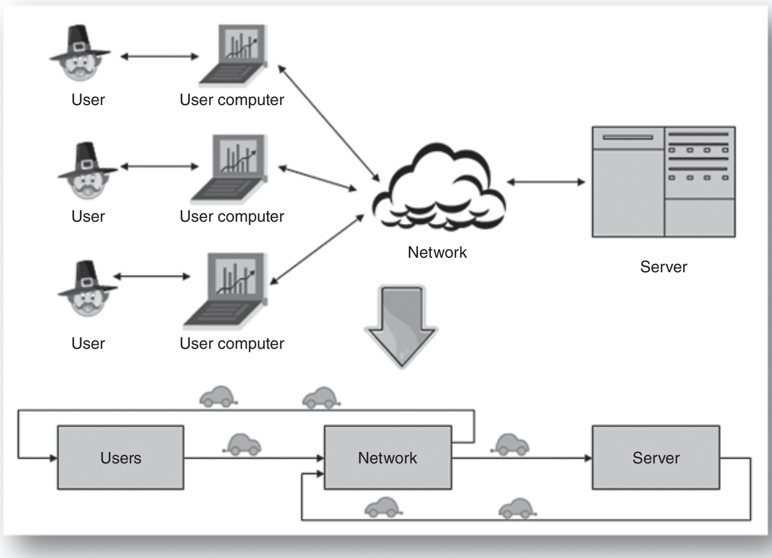

Figure 1.6. An application and its queuing model.

A transaction starts its journey when a user clicks on a menu item or a link that implicitly initiates interaction with the application. In a model it means that a transaction leaves node "users" and is processed in the nodes "network" and "server." At the end of its journey, the transaction comes back to node "users." The total time a transaction has spent in the nodes "network" and "server" is the transaction time.

Transaction time is equal to the total time it has spent in all nodes except node "users."

A transaction (think of it as a car) coming into a node with a queue might either get served in a processing unit right away if there is at least one idle unit, or wait in a queue if all processing units are busy serving other requests (Fig. 1.7).

The total time spent by a transaction-car in a node with a queue is:

Time in queue + time in processing unit

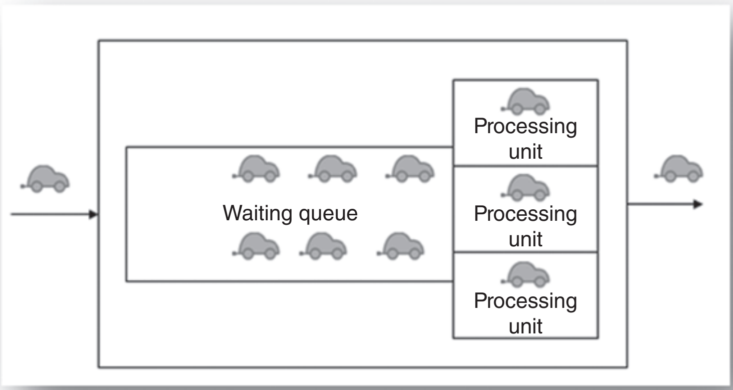

Figure 1.7. Transactions in a node with waiting queue and processing units.

This simple formula demonstrates that time in queues has a major impact on transaction response time. If a transaction does not wait in any queues, then its response time is equal to the time it has spent in all processing units of all nodes it visited. That is always the case when only a single user is working. When many users are active, waiting in queues fundamentally influences system agility because the time in queues can be much longer than the processing time; as such, waiting in queues becomes the prevailing component of system response time. The number and speed of processing units, as well as the rate of incoming requests, are among the factors defining a node's queue length.

The nodes model different hardware components of the computer system; each component can be presented by a node on different levels of abstraction depending on the goal of the modeling project. For example, the following representation of a hardware server can be sufficient for the sizing of enterprise applications:

Hardware Server in Real System	Matching Object in Node
CPU	Processing unit
Number of CPUs	Number of processing units
CPU speed (computer clock speed)	Speed of processing unit

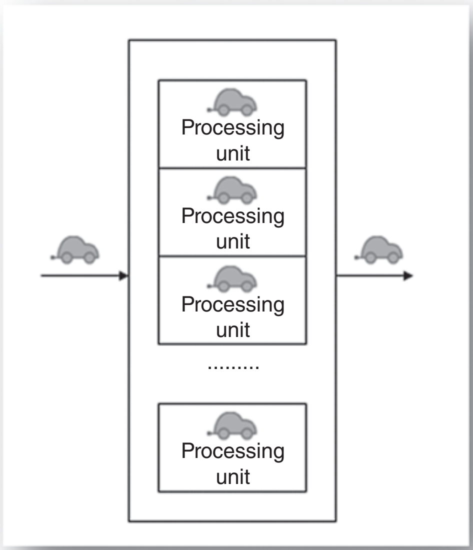

Figure 1.8. A node as a network model.

Representation of a network by a node has to take into account network specifics. A network is a complex conglomerate of controllers, routers, bridges, and other devices interconnected by physical cables or wirelessly, and it can be portrayed by a model on different levels. For the sizing of enterprise applications, a corporate network can be modeled by a node without a queue but with an unlimited number of processing units (Fig. 1.8).

This network model takes into account the most important network parameter—network delay. Network delay for each transaction is equal to the time a transaction is served in a processing unit. Because of an unlimited number of processing units, the node does not have a waiting queue. The interpretation of a network by a node with unlimited processing units is an adequate representation of a corporate network because it always has enough capacity to start processing immediately every incoming transaction initiated by the corporation's users. We have to note, however, that this interpretation might not be suitable for networks with low bandwidth. For example, a network with dial-in connections can be represented by the node with a finite number of processing units equal to the number of connections and without queues. If an incoming transaction finds that all processing units are busy (which means all connections are already allocated), the incoming transaction will be rejected and a user will have to redial.

A node with unlimited processing units is also an adequate model of the users of the most common *interactive* enterprise applications. Consider how a user works with an *interactive* enterprise application (Fig. 1.6):

1. User initiates a transaction (car-transaction leaves node "users")
2. User waits for a reply (car-transaction is in nodes "network" or "server")
3. User receives a reply and analyzes it (car-transaction is in node "users")
4. User kicks off next transaction (car-transaction leaves a node "users")

The sequence of 1–4 depicts a process where the next transaction starts after a reply to the previous one is received by a user. This is the most common scenario for enterprise applications that usually are designed as *interactive* systems (http://en.wikipedia.org/wiki/Interactive) because they have to support execution of business taskflows broken down by a number of steps where each subsequent step depends on the results of the previous ones. A user implements each step by starting a transaction and will launch the next one only after analyzing the system's reply to the previous one. The interactive application's user interface prevents the user from submitting more than one request. Here is an example of a business taskflow "Update sales data for the North region":

- Login into the application
- Initiate the North region database
- Open the data input form for the North region
- Input new sales data and save the updated form
- Execute countrywide data consolidation to update country level sales because of new North region data

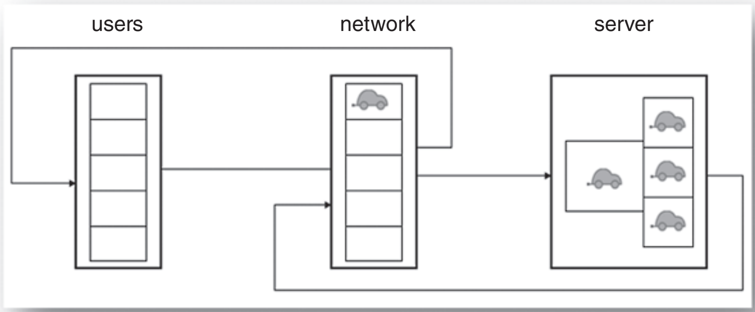

Figure 1.9. Five transactions in both nodes “network” and “server.”

- Run the financial report for country-level sales to make sure that data update for the North region was executed correctly
- Go to the next taskflow or log out

This example means that only one request per user is processed by an application at any given time. If a system has five users and all of them have launched transactions, then all five transactions (cars) will be served in the nodes representing the network and hardware server (Fig. 1.9).

The opposite situation occurs when all five users are analyzing data and are not waiting for the replies from system. In such a case, all five transactions are in node “users” and none are in the other two nodes (Fig. 1.10).

More often there is a situation when some users are waiting for replies and some users are analyzing results of completed previous transactions (Fig. 1.11).

From the examples of Figs. 1.9–1.11 we can conclude that in the queuing model of an *interactive* enterprise application, at any given time the number of transactions in all nodes are equal to the number of users who are logged into the system.

There are exceptions from the predominantly interactive nature of enterprise applications. Some applications allow particular business tasks to be executed in a noninteractive mode, letting users initiate a new transaction while waiting for completion of a previous one. This

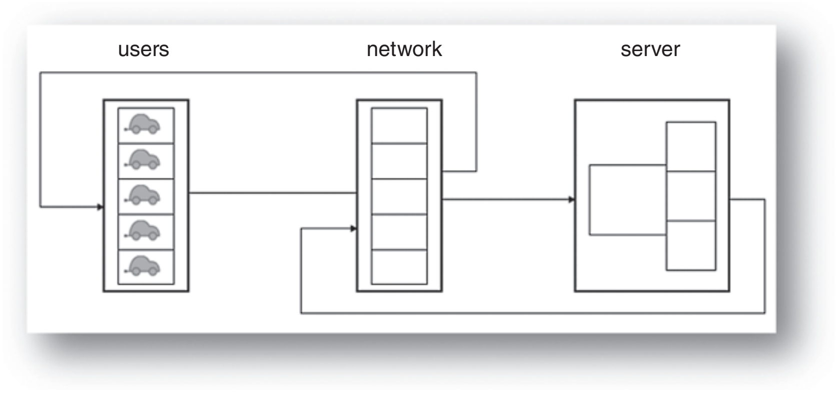

Figure 1.10. Five transactions in node "users."

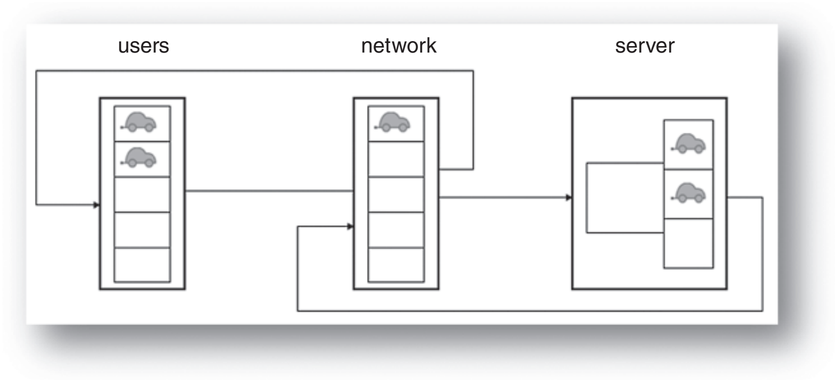

Figure 1.11. Five transactions are distributed among the nodes.

case can also be modeled and here is how. Suppose the transaction "Financial consolidation for the North region" is long and takes dozens of minutes; after requesting it a user, instead of waiting for the reply, can demand the transaction "Financial consolidation for the South region." In such a case we take into consideration additional transactions by increasing the number of processing units in node "users." If we do that, the number of processing units in node "users" actually equals the number of transactions but no longer the number of users.

The finite number of users of enterprise applications represents one of their fundamental distinctions from Internet applications—the latter have practically an unlimited number of users.

Enterprise applications, in respect to number of users, are similar to shopping malls with hundreds and thousands of visitors, while Internet applications are similar to telephone networks with millions of users.

Because of the finite number of users, we model enterprise applications by the ***closed*** queuing model (Fig. 1.6), which always has a constant number of transactions moving from node to node; for *interactive* applications, that number is equal to the number of logged users. Actually, the quantity of logged users fluctuates during the workday and reaches a maximum during the hours of the most intense application usage. These hours are of utmost interest for analysis because the enterprise application has to deliver acceptable performance during such a critical period; it is advisable to evaluate models for the time of maximum application usage.

A model of Internet application is an ***open*** queuing network (Fig. 1.12) accepting requests from a user community that is permanently changing in size. In the open queuing model, the number of transactions at any given time can be anywhere in a range from "0" to any finite value.

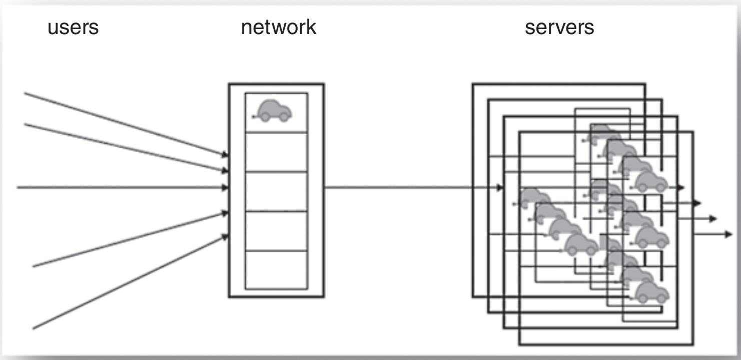

Figure 1.12. Open queuing network as a model of an Internet application.

Mapping application into its queuing model

Application:	*Queuing network*
Server, hardware appliance, users, network:	*Node*
CPU, disk, I/O controller:	*Processing unit*
Transaction:	*Request (imagine a car)*
Waiting for system resource:	*Request in a waiting queue*

1.5. TRANSACTION RESPONSE TIME AND TRANSACTION PROFILE

What the users of enterprise applications care most about (as well as complain most about) is transaction response time. The user's final verdict on application performance is always based on the difference between the transaction time delivered by the application vs. the user's expectation of that time. Models provide great help in understanding the components of transaction time and the factors they depend on. Let's consider a business transaction that retrieves a financial report. The transaction is initiated when a user clicks on an icon labeled "Request Report." At that moment, let's start an imaginary stopwatch to measures transaction time. The initiated transaction [we again depict it as a car (see Fig. 1.13)] starts its journey by moving from one node to another, waiting in queues, and spending time in processing units. Finally the car-transaction will get back to the user, and we will stop the stopwatch at that moment. The time measured by the stopwatch is the transaction time—the sum of all time intervals a car-transaction has spent in waiting queues and processing units of all nodes that represent the system hardware. A "cloud" on Fig. 1.13 encompasses the nodes that contribute to transaction time. Transaction response time is the total of waiting and processing times in the nodes "network," "Web server," "Application server," and "Database server."

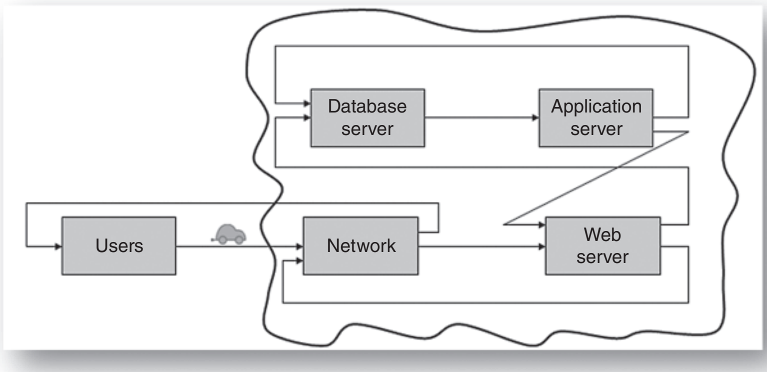

Figure 1.13. Transaction time is time spent in the "cloud."

The table below describes the correlations among the happenings in an application and in its model:

Happenings in System	Happenings in Model
User initiated transaction by clicking on "Request Report" icon	Transaction entered cloud
Network serves transaction	Transaction served in node "network"
Transaction is processed in Web server	Transaction served in node "Web server"
Transaction is processed in Application server	Transaction served in node "Application server"
Processing in Application server requires data from the database; the Application server communicates a number of times with the database to retrieve data	Transaction served in nodes "Application server" and "Database server," visiting them a few times
After the financial report is finally generated by the Application server, the Web server processes the data to send them back over the network to a user	Transaction served in node "Web server"
Report travels back to a user over the network	Transaction served in node "network"
Now it is the user's time to analyze the report	Transaction served in node "users"

Figure 1.13 and the relationships described above indicate that transaction response time depends on the following factors:

- The number of active system users (users initiate requests; as more users initiate requests, there are more transactions in the nodes in a cloud, which means longer waiting queues)
- System architecture (more servers and connections among them allow for more varieties of transaction itineraries to exist in a cloud)
- The speed and numbers of the processing units in each node in a cloud
- Processing algorithms implemented in the application software

Transaction response time is a function of time intervals spent in the nodes that represent servers, networks, and other hardware components. It depends on the number of active users, hardware architecture, software algorithms, as well as the speed and number of hardware components.

The transaction profile is a set of time intervals a transaction has spent in all processing units (but not in queues!) it has visited while served by the application.

When analyzing applications' models we will need to know the **transaction profile**—the number of time intervals a transaction has spent in all processing units (but not queues!) it has visited while served by the application. Time in a particular processing unit is often called *service demand*.

This is an example of a transaction profile for a model on Fig. 1.13:

	Time Spent in Processing Units (Seconds)			
Transaction Name	Network	Web Server	Application Server	Database Server
Sales Report	0.05	0.5	3.2	1.0
Profit and Loss Report	0.03	0.8	4.2	2.0

By adding up times in the processing units visited by a single transaction while being served in a system, we have a transaction response time when only a single request was served by the application:

$$\textit{Sales report transaction time} = 0.05 + 0.5 + 3.2 + 1.0 = 4.75 \textit{ seconds}$$

$$\begin{aligned}\textit{Profit and Loss report transaction time} &= 0.03 + 0.8 + 4.2 + 2.0 \\ &= 7.03 \textit{ seconds}\end{aligned}$$

The above calculation is applicable for a transaction of a predominant type that is served sequentially in processing units. In some cases, a transaction's particulars allow its parallelization. There are two parallelization techniques that can be implemented by application software: (1) concurrent execution of the same transaction by a few hardware servers (for example, serving transactions in the application and database servers); and (2) concurrent execution of the same transaction by a few components of the same hardware server (for example, processing transactions by more than one CPU).

We demonstrate in Chapter 9 how to take parallelized transactions into account while modeling applications. In all other chapters we consider the prevailing types of transactions that receive services in processing units sequentially.

1.6. NETWORK OF HIGHWAYS AS AN ANALOGY OF THE QUEUING MODEL

We introduced the car-transaction metaphor previously. It helped to visualize how transactions travel along servers and networks, how they receive "treatment" in servers, and why they have to wait in the queues; it also clarified the meaning of transaction time and transaction profile. We want to "capitalize" on the car-transaction metaphor by making a parallel between the queuing model and a network of highways with tollbooths. That analogy helps performance engineers to illustrate some bottlenecks to nontechnical application business users.

Two things in my life consumed more time than anything else—sizing and tuning applications, and driving. The number of times I found myself stuck in heavy traffic while approaching a tollbooth made me think about the similarities between a queuing model's nodes and the tollbooths (Fig. 1.14).

A toll plaza usually has a few tollbooths that serve each car by taking payment. Payment processing requires some time, which is

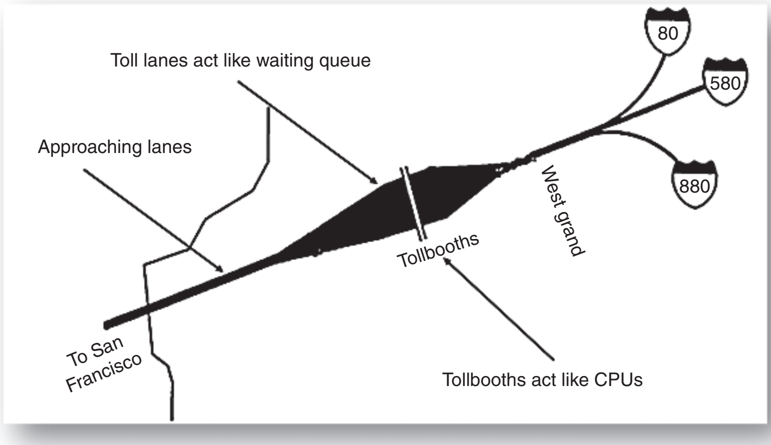

Figure 1.14. Tollbooths as a node.

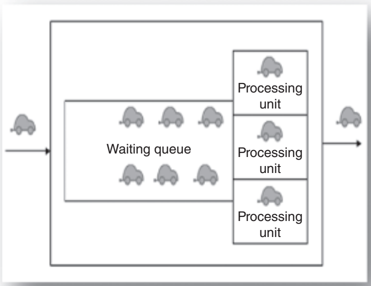

Figure 1.15. Tollbooths with waiting queue and their model (source: http://www.hindu.com/2008/02/09/stories/2008020957590400.htm).

equivalent to time in a processing unit of a node. If all the tollbooths are busy, cars congregate on the approaching lanes of a toll plaza forming waiting queues—exactly like waiting queues in the nodes of queuing models (Fig. 1.15).

Now picture a car traveling from point A to point B and passing through a few tollbooths along the way; this is just like a transaction traveling across a queuing network. We are going to use the highway analogy of a queuing network when we feel that it provides an additional boost to imagination.

TAKE AWAY FROM THE CHAPTER

- *Closed queuing networks with a finite number of requests are representative models of enterprise applications, revealing causes of the bottlenecks and pointing to the right actions to troubleshoot them.*
- *A queuing model's nodes correspond to hardware servers, networks, and users; a node's processing units stand for CPUs; business transactions are characterized by the requests waiting and those being served in the model's nodes.*
- *The most important indicator of a business application's performance is transaction time; it depends on all time intervals a transaction has spent in system servers waiting in queues and being processed. A critical component of transaction time is its wait time in system queues; wait time is a function of the intricate interdependencies of multiple factors: hardware speed, number of users, intensity of a user's transactions, system architecture, settings of software parameters . . . the list goes on and on.*
- *A network of highways is a helpful analogy of queuing models when performance engineers communicate to nontechnical application business users.*