A March 16, 2012, posting on ZDNet reveals that EC2 was made up of 454,600 servers; if we add the

number of servers supporting other AWS services, the total number of Amazon systems dedicated to

cloud computing is much larger. An unofficial estimate puts the number of servers used by Google in

January 2012 close to 1.8 million; this number is expected to be close to 2.4 million by early 2013

[289].

The complexity of such systems is unquestionable and raises questions such as:What are the generic

properties of complex systems?Howcan we manage such systems?Dowe have to consider radically new

ideas, such as self-management and self-repair, for future clouds consisting of millions of components?

Should we migrate from a strictly deterministic view of such systems to a nondeterministic one when

the desirable properties of a system are statistically assured? These are some of the questions examined

in this chapter.

When we think about complex systems, the human brain comes immediately to mind; the number

of neurons in the human brain is estimated to be between 80 and 120 billion. Technology systems with

a very large number of components, such as the space shuttle,1 a modern microprocessor with several million transistors,2 or the Internet with some 800 million hosts as of January 2010, are examples of

complex man-made systems.

A very large number of interaction channels among the components of a system is another defining

characteristic of a complex system. Indeed, the behavior of any system cannot be explained without precise

knowledge of the interactions among its components. Even systems with a relatively small number

of components can be complex; for example, the complexity of interactions between the gravitational

fields of the planets make our solar system a very complex one.

Discovering all possible interactions among the components of a system is a formidable task. Such

an effort is undertaken whenever we need to make progress in our understanding of the behavior of a

system we engineer or of the physical world surrounding us. Sometimes we are forced to discover such

interactions to build better systems. For example, the investigations following the failure of an aircraft,

the space shuttle, or any other system aim to discover the interaction channels that contributed to the

failure.

Predicting all possible interactions among the components of a system during the design process

is an even more daunting task. A good illustration of this problem is the so-called “death grip” effect

manifested by a smartphone released not long ago. The signal suffered a significant attenuation when

the device was gripped in a certain way, because the gain of the antenna embedded in the smartphone’s

case was drastically reduced. In retrospect, such an effect should have been foreseen, but during the

design process it was either overlooked or considered very unlikely.

Discovery of the interactions among the basic building blocks of a physical system is required for

the advancement of our knowledge. A major step in understanding the human brain is the development

of a comprehensive map of neural connections in the brain. The goal of the Human Connectome Project

sponsored by the National Institutes of Health (NIH) is to build a network map of the human brain in

healthy individuals.

Can we increase the number of components of a system while limiting the number of interaction paths

among its components? Though there is no positive answer to this question for most complex systems, a

solution applicable in the case of some computing and communication systems exists. In Section 7.10we

showed that systems interconnected by scale-free networks have this highly desirable property. Because

the degrees of the nodes enjoy a power-law distribution, the number of highly connected components is

very small, whereas the vast majority of components have one or a few connections. Thus, scalability

of large-scale systems can, in principle, be assured.

Interaction with the environment is an important dimension of system complexity; unfortunately

this dimension is often ignored. The more complex the interactions with the environment, the more

difficult it is to satisfy the often contradictory requirements of a dynamic environment, and the more

complex the system becomes. Moreover, changes in a dynamic environment are difficult to predict and

to accommodate in the original design of a system. For example, a system might not be able to perform

additional checks when the environment acquires new capabilities to challenge the system’s integrity.

Whenever we simulate a system, the complexity of the simulation increases with the number of

components and with the number of interaction paths among them. Indeed, the simulation program has to describe the properties of each component as well as the effect of each interaction on all components

involved. Symmetry and regularity help decrease the complexity of the simulation as multiple

components and multiple interactions share the same description; irregularities have the opposite effect.

The length of the description of all the system components is captured by the concept of Kolmogorov

complexity, discussed in Section 10.3.

Symmetry and regularity of computing and communication systems imply homogeneity; the hardware

and the software of individual components are identical or very similar. Large-scale systems

designed as collections of homogeneous systems are easier to assemble, their properties are better

understood, and they are more effectively managed than collections of heterogeneous systems. In

Section 1.3 we cited the hardware and software homogeneity of a cloud as one of the reasons we believe

that cloud computing could be successful, whereas heterogeneous large-scale distributed computing

systems could not be.

Because we want to increase the versatility and usefulness, as well as the performance of manmade

systems, we add new features and we increase the complexity of the system. This is especially

true whenever we try to accommodate changes that were not envisioned by the original design. For

example, the Internet was originally designed as a data network based on a best-effort model where

routers could freely discard packets. Major changes were necessary to support data streaming and

other types of communication with real-time delivery constraints when the retransmission of discarded

packets could not be tolerated. These changes have increased the complexity of the Internet. We also

saw in Chapter 6 that the management of cloud resources increases in complexity when we attempt to

mix best-effort applications with applications that have soft real-time constraints and when we add to

the mix applications with real-time constraints.