To understand the important properties of distributed systems, we use a model, an abstraction based on

two critical components: processes and communication channels. A process is a program in execution,

and a thread is a lightweight process. A thread of execution is the smallest unit of processing that can

be scheduled by an operating system.

A process is characterized by its state; the state is the ensemble of information we need to restart

a process after it was suspended. An event is a change of state of a process. The events affecting the

state of process p1 are numbered sequentially as e1

i , e2

i , e3

i , . . ., as shown in the space-time diagram in

Figure 2.1(a). A process p1 is in state σ

j

i immediately after the occurrence of event e j

i and remains in

that state until the occurrence of the next event, e j+1

i .

A process group is a collection of cooperating processes; these processes work in concert and

communicate with one another to reach a common goal. For example, a parallel algorithm to solve

a system of partial differential equations (PDEs) over a domain D may partition the data in several

segments and assign each segment to one of the members of the process group. The processes in the

group must cooperate with one another and iterate until the common boundary values computed by one

process agree with the common boundary values computed by another.

A communication channel provides the means for processes or threads to communicate with one

another and coordinate their actions by exchanging messages.Without loss of generality,we assume that

communication among processes is done only by means of send (m) and receive (m) communication

events, where m is a message.We use the term message for a structured unit of information, which can

be interpreted only in a semantic context by the sender and the receiver. The state of a communication

channel is defined as follows: Given two processes pi and p j , the state of the channel, ξi , j, from pi to

p j consists of messages sent by pi but not yet received by p j .

These two abstractions allow us to concentrate on critical properties of distributed systems without

the need to discuss the detailed physical properties of the entities involved. Themodel presented is based

on the assumption that a channel is a unidirectional bit pipe of infinite bandwidth and zero latency, but

unreliable; messages sent through a channel may be lost or distorted or the channel may fail, losing its

ability to deliver messages.We also assume that the time a process needs to traverse a set of states is of

no concern and that processes may fail or be aborted.

A protocol is a finite set of messages exchanged among processes to help them coordinate their

actions. Figure 2.1(c) illustrates the case when communication events are dominant in the local history

of processes, p1, p2, and p3. In this case only e5

1 is a local event; all others are communication events.

The particular protocol illustrated in Figure 2.1(c) requires processes p2 and p3 to send messages to

the other processes in response to a message from process p1.

The informal definition of the state of a single process can be extended to collections of communicating

processes. The global state of a distributed system consisting of several processes and communication

channels is the union of the states of the individual processes and channels [34].

Call h j

i the history of process pi up to and including its j -th event, e j

i , and call σ

j

i the local state

of process pi following event e j

i . Consider a system consisting of n processes, p1, p2, . . . , pi , . . . , pn

with σ

ji

i the local state of process pi ; then the global state of the system is an n-tuple of local states

 ( j1, j2,..., jn ) =

σ

j1

1 , σ

j2

2 , . . . , σ

ji

i , . . . , σ

jn

n

. (2.10)

The state of the channels does not appear explicitly in this definition of the global state because the

state of the channels is encoded as part of the local state of the processes communicating through the

channels.

The global states of a distributed computation with n processes form an n-dimensional lattice. The

elements of this lattice are global states  ( j1, j2,..., jn )

σ

j1

1 , σ

j2

2 , . . . , σ

jn

n

.

Figure 2.2(a) shows the lattice of global states of the distributed computation in Figure 2.2(b) This

is a two-dimensional lattice because we have two processes, p1 and p2. The lattice of global states for

the distributed computation in Figure 2.1(c) is a three-dimensional lattice, and the computation consists

of three concurrent processes, p1, p2, and p3.

The initial state of the system in Figure 2.2(b) is the state before the occurrence of any event and it is

denoted by (0,0); the only global states reachable from (0,0) are (1,0) and (0,1). The communication

events limit the global states the system may reach; in this example the system cannot reach the state

 (4,0) because process p1 enters state σ4 only after process p2 has entered the state σ1. Figure 2.2(b)

shows the six possible sequences of events to reach the global state  (2,2):

e1

1, e2

1, e1

2, e2

2

,

e1

1, e1

2, e2

1, e2

2

,

e1

1, e1

2, e2

2, e2

1

,

e1

2, e2

2, e1

1, e2

1

,

e1

2, e1

1, e2

1, e2

2

,

e1

2, e1

1, e2

2, e2

1

.

(2.11)

An interesting question is how many paths does it take to reach a global state. The more paths exist,

the harder it is to identify the events leading to a state when we observe an undesirable behavior of the

system. A large number of paths increases the difficulty of debugging the system.

We conjecture that in the case of two threads in Figure 2.2(b) the number of paths from the global

state  (0,0) to  (m,n) is

N(m,n)

p

= (m + n)!

m!n! . (2.12)

We have already seen that there are six paths leading to state  (2,2); indeed,

N(2,2)

p

= (2 + 2)!

2!2!

= 24

4

= 6. (2.13)

To prove Equation 2.12 we use a method resembling induction; we notice first that the global state

 (1,1) can only be reached from the states  (1,0) and  (0,1) and that N(1,1)

p = (2)!/1!1! = 2. Thus, the

formula is true for m = n = 1. Then we show that if the formula is true for the (m − 1, n − 1) case it

will also be true for the (m, n) case. If our conjecture is true, then

N[(m−1),n]

p

=

[(m − 1) + n]!

(m − 1)!n! . (2.14)

and

N[m,(n−1)]

p

=

[m + (n − 1)]!

m!(n − 1)! . (2.15)

We observe that the global state  (m,n), ∀(m, n)   1 can only be reached from two states,  (m−1,n)

and  (m,n−1) (see Figure 2.3), thus:

N(m,n)

p

= N(m−1,n)

p

+ N(m,n−1)

p . (2.16)

It is easy to see that indeed,

[(m − 1) + n]!

(m − 1)!n!

+

[m + (n − 1)]!

m!(n − 1)!

= (m + n − 1)!

1

(m − 1)!n!

+ 1

m!(n − 1)!

= (m + n)!

m!n! .

(2.17)

This shows that our conjecture is true; thus, Equation 2.12 gives the number of paths to reach the global

state  (m,n) from  (0,0) when two threads are involved. This expression can be generalized for the case

of q threads; using the same strategy, it is easy to see that the number of path from the state  (0,0,...,0)

to the global state  (n1,n2,...,nq ) is

N(n1,n2,...,nq )

p = (n1 + n2 +     +nq )!

n1!n2! . . . nq ! . (2.18)

Indeed, it is easy to see that

N(n1,n2,...,nq )

p = N(n1−1,n2,...,nq )

p + N(n1,n2−1,...,nq )

p +     + N(n1,n2,...,nq−1)

p . (2.19)

Equation 2.18 gives us an indication of how difficult it is to debug a system with a large number of

concurrent threads.

Many problems in distributed systems are instances of the global predicate evaluation problem

(GPE), where the goal is to evaluate a Boolean expression whose elements are functions of the global

state of the system.