Distributed Systems Concepts

Ch. 10 and 14-17

Figure 10.1
Skew between computer clocks in a distributed system

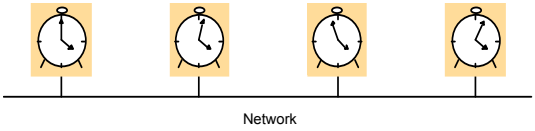
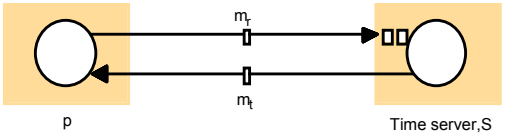
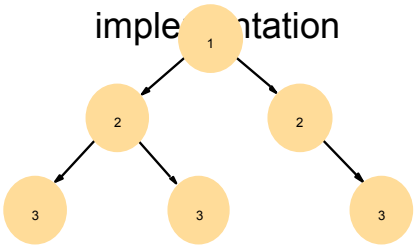


Figure 10.2
Clock synchronization using a time server



An example synchronization subnet in an NTP implementation



Note: Arrows denote synchronization control, numbers denote strata.

Figure 10.4
Messages exchanged between a pair of NTP peers

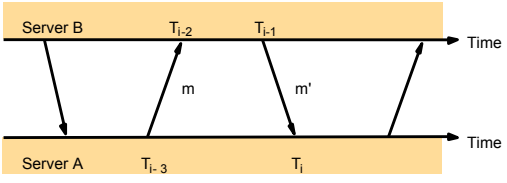


Figure 10.5
Events occurring at three processes

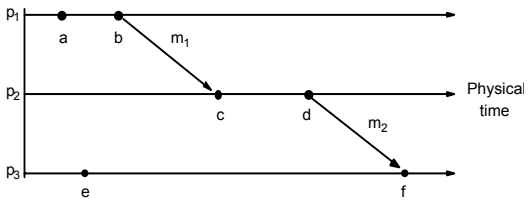


Figure 10.6
Lamport timestamps for the events shown in Figure 10.5

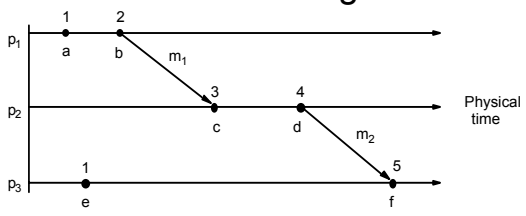


Figure 10.7
Vector timestamps for the events shown in Figure 10.5

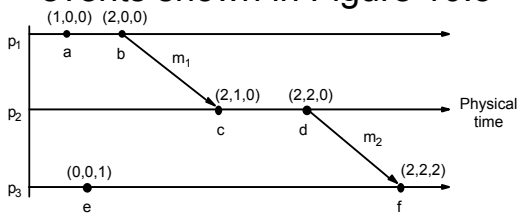


Figure 10.8
Detecting global properties

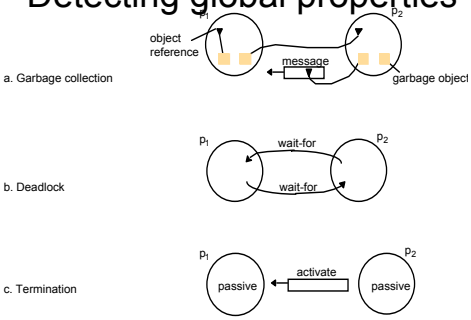


Figure 10.9
Cuts

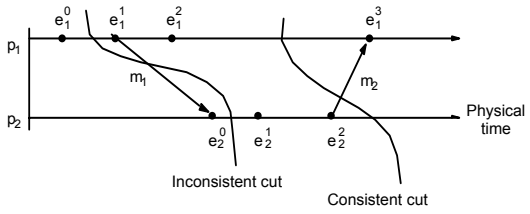


Figure 10.10
Chandy and Lamport's 'snapshot' algorithm

Marker receiving rule for process p_i

On p_i's receipt of a *marker* message over channel c:

if (p_i has not yet recorded its state) it

records its process state now;

records the state of c as the empty set;

turns on recording of messages arriving over other incoming channe

else

p_i records the state of c as the set of messages it has received over c since it saved its state.

end if

Marker sending rule for process p_i

After p_i has recorded its state, for each outgoing channel c:

p_i sends one marker message over c

(before it sends any other message over c).

Figure 10.11
Two processes and their initial states

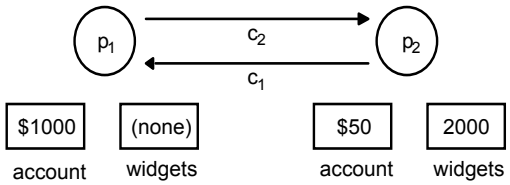


Figure 10.12
The execution of the processes in Figure 10.11

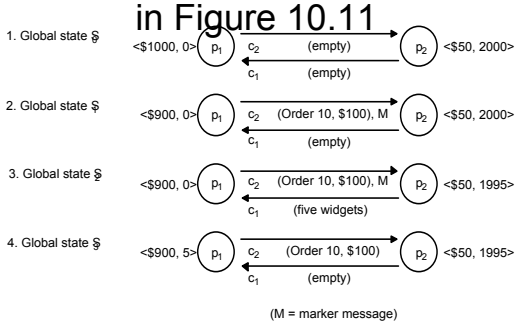


Figure 10.13
Reachability between states in the snapshot algorithm

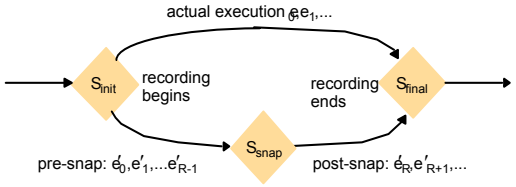
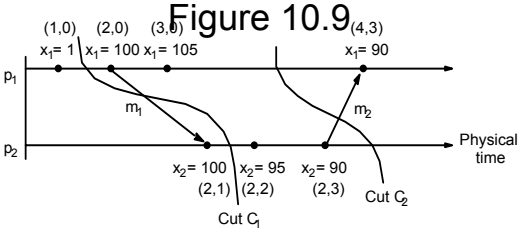


Figure 10.9
Vector timestamps and variable values for the execution of Figure 10.9



Chapter 11,12, and 13

- Are on transaction an concurrency control that are typically covered in a data base course.

Ch 14: Fault Tolerance

Figure 14.1
A basic architectural model for the management of replicated data

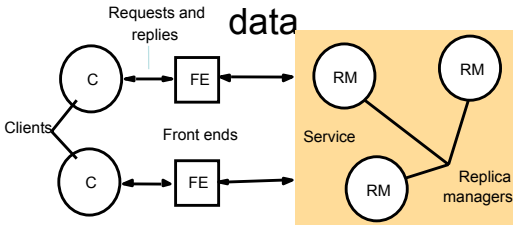


Figure 14.2
Services provided for process groups

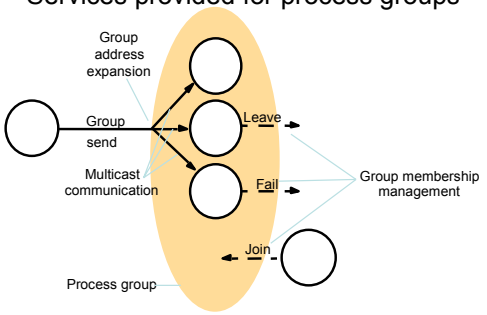


Figure 14.3
View-synchronous group communication

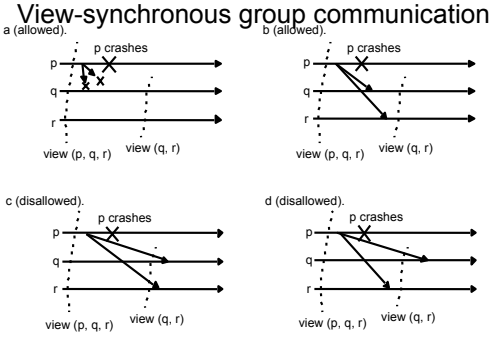


Figure 14.4
The passive (primary-backup) model for fault tolerance

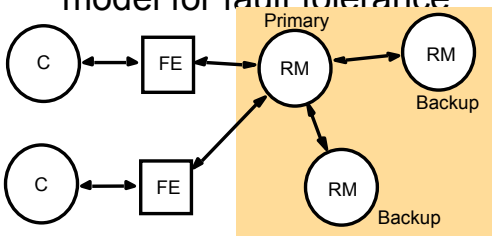
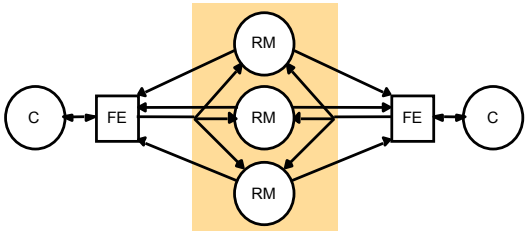
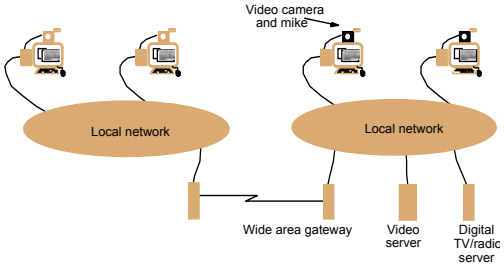


Figure 14.5
Active replication



Ch.15: Distributed Multimedia Systems

Figure 15.1
A distributed multimedia system



The window of scarcity for computing and communication

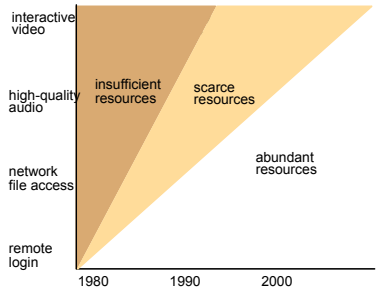
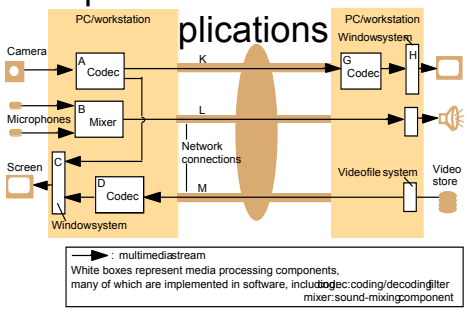


Figure 15.3 Characteristics of typical multimedia streams

	Data rate (approximate)	*Sample or frame frequency*	*size*
Telephone speech	64 kbps	8 bits	8000/sec
CD-quality sound	1.4 Mbps	16 bits	44,000/sec
Standard TV video (uncompressed)	120 Mbps	up to 640x 480 pixelsx 16 bits	24/sec
Standard TV video (MPEG-1 compressed)	1.5 Mbps	variable	24/sec
HDTV video (uncompressed)	1000–3000 Mbps	up to 1920x 1080 pixelsx 24 bits	24–60/sec
HDTV video MPEG-2 compressed)	10–30 Mbps	variable	24–60/sec

Typical infrastructure components for multimedia plications



QoS specifications for components of the application shown in Figure 15.4

Component		*Direction*		*Latency*	*Loss rate*	*Resources required*
	Camera	Out:	10 frames/sec, raw video 640x480x16 bits		Zero	
A	Codec	In:	10 frames/sec, raw video	Interactive	Low	10 ms CPU each 100 ms;
		Out:	MPEG-1 stream			10 Mbytes RAM
B	Mixer	In:	2 44 kbps audio	Interactive	Very low	1 ms CPU each 100 ms;
		Out:	1 44 kbps audio			1 Mbytes RAM
H	Window system	In:	various	Interactive	Low	5 ms CPU each 100 ms;
		Out:	50 frame/sec framebuffer			5 Mbytes RAM
K	Network connection	In/Out:	MPEG-1 stream, approx. 1.5 Mbps	Interactive	Low	1.5 Mbps, low-loss stream protocol
L	Network connection	In/Out:	Audio 44 kbps	Interactive	Very low	44 kbps, very low-loss stream protocol

Figure 15.6 The QoS manager's task

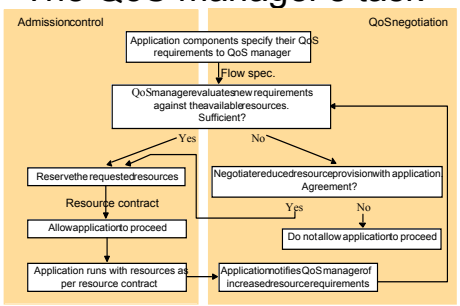


Figure 15.7 Traffic shaping algorithms

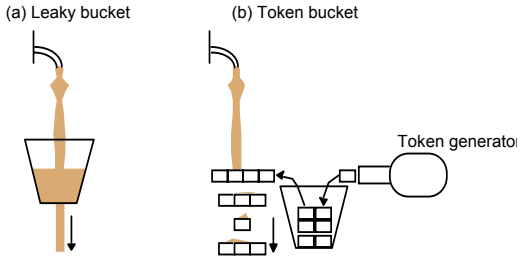
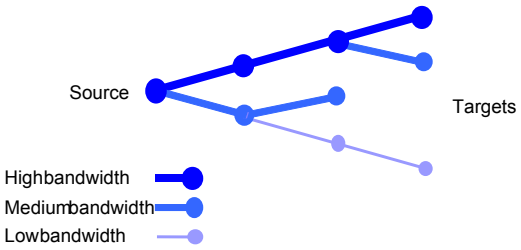


Figure 15.8
The RFC 1363 Flow Spec

	Protocol version
Bandwidth:	Maximum transmission unit
	Token bucket rate
	Token bucket size
	Maximum transmission rate
Delay:	Minimum delay noticed
	Maximum delay variation
Loss:	Loss sensitivity
	Burst loss sensitivity
	Loss interval
	Quality of guarantee

Figure 15.9
Filtering



Highbandwidth
Mediumbandwidth
Lowbandwidth

Figure 15.10
Tiger video file server hardware configuration

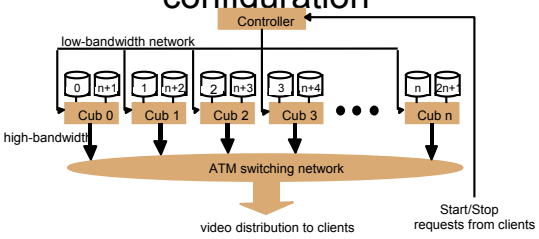
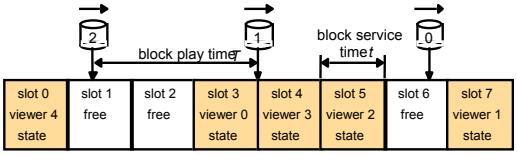


Figure 15.11
Tiger schedule



16: Distributed Shared Memory

Figure 16.1
The distributed shared memory abstraction

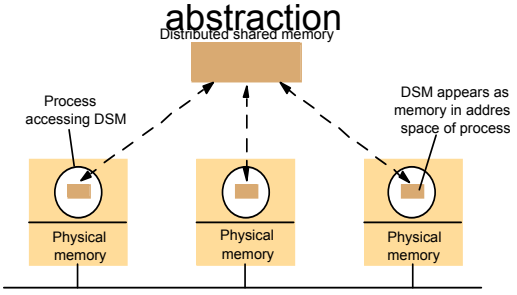


Figure 16.2
Mether system program - slide 1

```
#include "world.h"
struct  shared { int a,b; };

Program Writer:
main()
{
    struct shared *p;
    methersetup();                  /* Initialize the Mether run-time */
    p = (struct shared *)METHERBASE;
                                    /* overlay structure on METHER segment */
    p->a = p->b = 0;                /* initialize fields of struct */
    while(TRUE)  {                  /* continuously update structure fields */
        p –>a = p –>a + 1;
        p –>b = p –>b - 1;
    }
}
```

Continued on next slide...

Figure 16.2
Mether system program - slide 2

```
Program Reader:
main()
{
    struct shared *p;
    methersetup();
    p = (struct shared *)METHERBASE;
    while(TRUE) {               /* read the fields once every second */
        printf("a = %d, b = %d\n", p –>a, p –>b);
        sleep(1);
    }
}
```

Figure 16.5
DSM using write-update

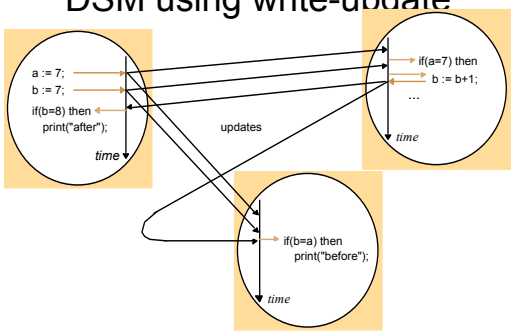


Figure 16.6
Data items laid out over pages

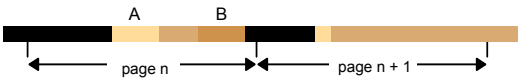


Figure 17.1
IDL interfaces Shape and ShapeList

```
struct
Rectangle{
    long width;
    long height;
    long x;
    long y;
};
interface Shape {
    long getVersion() ;
    GraphicalObject getAllState() ;          // returns state of the GraphicalObject
};

typedef sequence <Shape, 100> All;
interface ShapeList {
    exception FullException{ };
    Shape newShape(in GraphicalObject g) raises (FullException);
    All allShapes();                         // returns sequence of remote object references
    long getVersion() ;
};
```

```
struct GraphicalObject
{
    string type;
    Rectangle enclosing;
    boolean isFilled;
};
```

Figure 17.2
Java interface *ShapeList* generated by *idltojava* from CORBA interface *ShapeList*

```
public interface ShapeList extends org.omg.CORBA.Object {
    Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException;
    Shape[] allShapes();
    int getVersion();
}
```

Figure 17.3
ShapeListServant class of the Java server program for CORBA interface *ShapeList*

```
import org.omg.CORBA.*;
class ShapeListServant extends _ShapeListImplBase{
      ORB theOrb;
      private Shape theList[];
      private int version;
      private static int n=0;
      public ShapeListServant(ORB orb){
            theOrb = orb;
            // initialize the other instance variables
      }
      public Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException {   1
            version++;
            Shape s = new ShapeServant( g, version);
            if(n >=100) throw new ShapeListPackage.FullException();
            theList[n++] = s;                                                            2
                  theOrb.connect(s);
                  return s;
      }
      public  Shape[] allShapes(){ ... }
      public int getVersion() { ... }
}
```

Figure 17.4
Java class *ShapeListServer*

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class ShapeListServer {
      public static void main(String args[]) {
            try{
                  ORB orb = ORB.init(args, null);                                        1
                  ShapeListServant shapeRef = new ShapeListServant(orb);                 2
                  orb.connect(shapeRef);                                                 3
                  org.omg.CORBA.Object objRef =
                        orb.resolve_initial_references("NameService");                   4
                  NamingContext ncRef = NamingContextHelper.narrow(objRef);
                  NameComponent nc = new NameComponent("ShapeList", "");                 5
                  NameComponent path[] = {nc};                                           6
                  ncRef.rebind(path, shapeRef);                                          7
                  java.lang.Object sync = new java.lang.Object();
                  synchronized (sync) {   sync.wait();}
            } catch (Exception e) { ... }
      }
}
```

Figure 17.5
Java client program for CORBA interfaces *Shape* and *ShapeList*

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class ShapeListClient{
      public static void main(String args[]) {
            try{
                  ORB orb = ORB.init(args, null);
      1
                  org.omg.CORBA.Object objRef =
                        orb.resolve_initial_references("NameService");
                  NamingContext ncRef = NamingContextHelper.narrow(objRef);
                  NameComponent nc = new NameComponent("ShapeList", "");
                  NameComponent path [] = { nc };
                  ShapeList shapeListRef =
                        ShapeListHelper.narrow(ncRef.resolve(path));
      2
                  Shape[] sList = shapeListRef.allShapes();
      3
                  GraphicalObject g = sList[0].getAllState();
```

Figure 17.6
The main components of the CORBA architecture

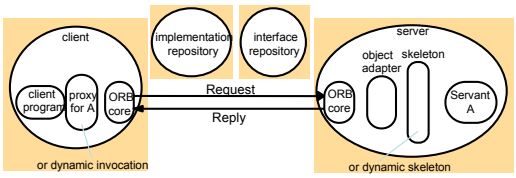


Figure 17.7
IDL module Whiteboard

```
module Whiteboard {
    struct Rectangle{
    ...} ;
    struct GraphicalObject {
    ...};
    interface Shape {
    ...};
    typedef sequence <Shape, 100> All;
    interface ShapeList {
    ...};
};
```

Figure 17.8
IDL constructed types – 1

Type	*Examples*	*Use*
sequence	*typedef sequence <Shape, 100> All;* *typedef sequence <Shape> All* bounded and unbounded sequences of *Shapes*	Defines a type for a variable-length sequence of elements of a specified IDL type. An upper bound on the length may be specified.
string	*String name;* *typedef string<8> SmallString;* unboundedand bounded sequences of characters	Defines a sequences of characters, terminated by the null character. An upper bound on the length may be specified.
array	*typedef octet uniqueId[12];* *typedef GraphicalObject GO[10][8]*	Defines a type for a multi-dimensional fixed-length sequence of elements of a specified IDL type.

this figure continues on the next slide

Figure 17.8
IDL constructed types – 2

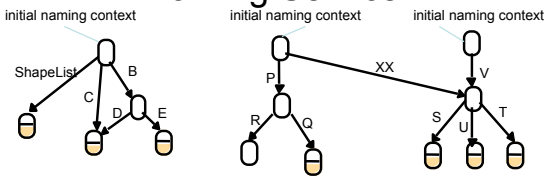
Type	*Examples*	*Use*
record	*struct GraphicalObject { string type; Rectangle enclosing; boolean isFilled; };*	Defines a type for a record containing a group of related entities. *Structs* are passed by value in arguments and results.
enumerated	*enum Rand (Exp, Number, Name);*	The enumerated type in IDL maps a type name onto a small set of integer values.
union	*union Exp switch (Rand) { case Exp: string vote; case Number: long n; case Name: string s; };*	The IDL discriminated union allows one of a given set of types to be passed as an argument. The header is parameterized by an *enum* which specifies which member is in use.

Page 684
CORBA interoperable object references

IOR format

IDL interface type name	Protocol and address details			Object key	
interface repository identifier	IIOP	host domain name	port number	adapter name	object name

Figure 17.9
Naming graph in CORBA Naming Service



Part of the CORBA Naming Service NamingContext interface in IDL

struct NameComponent { string id; string kind; };

typedef sequence <NameComponent> Name;

interface NamingContext {

void bind (in Name n, in Object obj);
binds the given name and remote object reference in my context.

void unbind (in Name n);
removes an existing binding with the given name.

void bind_new_context(in Name n);
creates a new naming context and binds it to a given name in my context.

Object resolve (in Name n);
looks up the name in my context and returns its remote object reference.

void list (in unsigned long how_many, out BindingList bl, out BindingIterator bi);
returns the names in the bindings in my context.

};

Figure 17.11
CORBA event channels