Complex data types can be mapped to XML for transmission across the network.

- **Example: primitive data types**
  - SOAP provides built-in support for encoding arrays.

- **Example: array datatype**
  - Complex data types are mapped to XML schema types;
  - SOAP platforms provide API for creating custom mapping.

  e.g. `writeSchema` to specify an XML schema definition

```
<soap:Body>
  <n:echoString
    xmlns:n='http://tempuri.org/mapping-server.Primitive'>
    <value xsi:type='xsd:string' cat</value>
  </n:echoString>
</soap:Body>
```

Heterogeneous environment means different data representations

- Requirement to enable data transformation.

**Independence** from hardware characteristics while exchanging messages means: use of external data representation.

- **Marshalling and unmarshalling**
- **Centralized transformation**
- **Decentralized transformation**
- **Common external data representation**
- **XML as common data representation**
- **Java Object Serialization**
XML as common data representation

Complex data types can be mapped to XML for transmission across the network.

**Example: primitive datatypes**
SOAP provides built-in support for encoding arrays.

**Example: array datatype**
Complex data types are mapped to XML schema types;
SOAP platforms provide APIs for creating custom mapping.

<table>
<thead>
<tr>
<th>high</th>
<th>low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application specific data encoding language</td>
<td>XML</td>
</tr>
<tr>
<td>General data encoding language</td>
<td>ASN 1</td>
</tr>
<tr>
<td>Network data encoding language</td>
<td>Sun XDR</td>
</tr>
</tbody>
</table>

**Example**
public class Person implements Serializable {  
    private String name;  
    private int year;  
    // methods such as constructor, etc  
    example object Person p = new Person("Smith", 1984);  
    serialized form:  
    Person | 8-byte version number | h0  
    3 | int year | java.lang.String name  

Java Object Serialization

Serialization means flattening an object (or a set of objects) into a serial form that is suitable for storing the object on disk or transmitting the object in a message.

**Serialize an Object**
The following information is written out:
- class information with class name and version number
- number, types and names of instance variables
- values of instance variables

```java
public class Person implements Serializable {  
    private String name;  
    private int year;  
    // methods such as constructor, etc  
    example object Person p = new Person("Smith", 1984);  
    serialized form:  
    Person | 8-byte version number | h0  
    3 | int year | java.lang.String name  
```
number, types and names of instance variables
values of instance variables

Example
public class Person implements Serializable {
    private String name;
    private int year;
    // methods such as constructor, etc
}

element object Person p = new Person("Smith", 1984);
serialized form:
    Person | 8-byte version number | h0
    3 | int year | java.lang.String name
    1984 | 5 Smith

Java Methods

create an instance of the class ObjectOutputStream and invoke writeObject

to deserialize an object
	on an ObjectInputStream use its readObject method to reconstruct the original object.

Basic mechanisms for distributed applications

Issues

The following section discusses several important basic issues of distributed applications.

- Data representation in heterogeneous environments.
- Discussion of an execution model for distributed applications.
- What is the appropriate error handling?
- What are the characteristics of distributed transactions?
- What are the basic aspects of group communication (e.g., algorithms used by ISIS)?
- How are messages propagated and delivered within a process group in order to maintain a consistent state?

External data representation

Time

Distributed execution model

Failure handling in distributed applications

Distributed transactions

Group communication

Distributed Consensus

Authentication service Kerberos

Heterogeneous environment means different data representations

requirement to enable data transformation.

Independence from hardware characteristics while exchanging messages means: use of external data representation.

Marshalling and unmarshalling

Centralized transformation

Decentralized transformation

Common external data representation

XML as common data representation

Java Object Serialization

Time is an important and interesting issue in distributed systems

We need to measure time accurately:

- to know the time an event occurred at a computer
- to do this we need to synchronize its clock with an authoritative external clock

Algorithms for clock synchronization useful for

- concurrency control based on timestamp ordering
- authenticity of requests e.g. in Kerberos

Three notions of time:

- time seen by an external observer → global clock of perfect accuracy.
  However, there is no global clock in a distributed system
- time seen on clocks of individual processes.
- logical notion of time: event a occurs before event b.

Introduction

Synchronizing physical clocks
Each computer in a distributed system (DS) has its own internal clock used by local processes to obtain the value of the current time. Processes on different computers can timestamp their events but clocks on different computers may give different times. Computer clocks drift from perfect time and their drift rates differ from one another.

Clock drift rate: the relative amount that a computer clock differs from a perfect clock.

Even if clocks on all computers in a DS are set to the same time, their clocks will eventually vary quite significantly unless corrections are applied.

### Timestamp

**Skew between clocks**

**Coordinated Universal Time (UTC)**

Computer clocks are not generally in perfect agreement. Skew: the disagreement between two clocks (at any instant).

Computer clocks are subject to clock drift (they count time at different rates).

Clock drift rate: the difference per unit of time from some ideal reference clock. Ordinary quartz clocks drift by about 1 sec in 11-12 days.

To timestamp events, we use the computer's clock:

1. At real time $t$, the operating system reads the time on the computer's hardware clock $H_i(t)$
2. It calculates the time on its software clock $C_i(t)$
   
   $C_i(t) = a H_i(t) + b$

   e.g., a 64 bit number giving nanoseconds since some "base time". In general, the clock is not completely accurate.

   but if $C_i$ behaves well enough, it can be used to timestamp events at $p$.

---

**International Atomic Time** is based on very accurate physical clocks (drift rate $10^{-12}$).

UTC is an international standard for time keeping. It is based on atomic time, but occasionally adjusted to astronomical time.

It is broadcast from radio stations on land and satellite (e.g., GPS). Computers with receivers can synchronize their clocks with these timing signals.

Signals from land-based stations are accurate to about 0.1-10 milliseconds.

Signals from GPS are accurate to about 1 microsecond.
Time is an important and interesting issue in distributed systems. We need to measure time accurately to know the time an event occurred at a computer. To do this, we need to synchronize its clock with an authoritative external clock.

Algorithms for clock synchronization useful for concurrency control based on timestamp ordering and authenticity of requests, e.g., in Kerberos.

Three notions of time:
- Time seen by an external observer is global clock of perfect accuracy.
- However, there is no global clock in a distributed system.
- Time seen on clocks of individual processes; logical notion of time; event a occurs before event b.

Introduction

Synchronizing physical clocks

External - internal synchronization

Clock correctness

Synchronization in a synchronous system

Cristian's method for an asynchronous system

Network Time Protocol (NTP)

Precision Time Protocol (PTP)

A computer's clock $C_i$ is synchronized with an external authoritative time source $S$, so that:

$|S(t) - C_i(t)| < D$ for $i = 1, 2, ..., N$ over an interval $I$ of real time $t$.

The clocks $C_i$ are accurate to within the bound $D$.

Internal synchronization

The clocks of a pair of computers are synchronized with one another so that:

$|C_i(t) - C_j(t)| < D$ for $i, j = 1, 2, ..., N$ over an interval $I$ of real time $t$.

The clocks $C_i$ and $C_j$ agree within the bound $D$.

Internally synchronized clocks are not necessarily externally synchronized, as they may drift collectively. If the set of processes $P$ is synchronized externally within a bound $D$, it is also internally synchronized within bound $2D$.

A hardware clock $H$ is said to be correct if its drift rate is within a bound $q > 0$, e.g., $10^{-5}$ sec/sec.

The error in measuring the interval between real times $t$ and $t'$ is bounded:

$(1 - q)(t' - t) \leq H(t') - H(t) \leq (1 + q)(t' - t)$, where $t' > t$.

No jumps in time readings of hardware clocks.

Weaker condition of monotonicity

$t' > t \rightarrow C(t') > C(t)$

E.g., required by Unix make.

We can achieve monotonicity with a hardware clock that runs fast by adjusting the values of $a$ and $b$ of

$C_i(t) = aH_i(t) + b$

A faulty clock is one that does not obey its correctness condition. Crash failure - a clock stops ticking, arbitrary failure - any other failure e.g., jumps in time.
a synchronous distributed system is one in which the following bounds are defined
the time to execute each step of a process has known lower and upper bounds.
each message transmitted over a channel is received within a known bounded time.
each process has a local clock whose drift rate from real time has a known bound.

**Internal synchronization in a synchronous system**

One process $p_1$ sends its local time $t$ to process $p_2$ in a message $m_2$. $p_2$ could set its clock to $t + T_{trans}$ where $T_{trans}$ is the time to transmit $m_2$. $T_{trans}$ is unknown but:

$$\min \leq T_{trans} \leq \max$$

uncertainty $u = (\max - \min)$. Set clock to:

$$t + (\max + \min) / 2$$

then skew $\leq u/2$.

**Observations:**
round trip times between processes are often reasonably short in practice, yet theoretically unbounded
practical estimate possible if round-trip times are sufficiently short in comparison to required accuracy.

**Approach**

- Berkeley algorithm

Both algorithms (Crstian and Berkeley) are not really suitable for Internet.

**Discussion**
The approach has several problems. It is only suitable for deterministic LAN environment or Intranet.
- a single time server might fail
- redundancy through of servers, multicast requests
- it does not deal with faulty time servers
- how to decide if replies vary (byzantine agreement problems)
- impostor providing false clock readings

**Cristian’s method for an asynchronous system**

- Network Time Protocol (NTP)
- Precision Time Protocol (PTP)

physical clocks are used to compute the current time in order to timestamp events, such as:
- modification date of a file
- time of an e-commerce transaction for auditing purposes

**External - Internal Synchronization**

- Synchronization in a synchronous system
- Clock correctness
- Network Time Protocol (NTP)
- Precision Time Protocol (PTP)
Cristian’s method for an asynchronous system

Observations:
round trip times between processes are often reasonably short in practice, yet theoretically unbounded practical estimate possible if round-trip times are sufficiently short in comparison to required accuracy

Approach
Berkeley algorithm

Both algorithms (Cristian and Berkeley) are not really suitable for Internet.

NTP - synchronization of servers

The synchronization subnet can reconfigure if failures occur, e.g.,
a primary that loses its UTC source can become a secondary

a secondary that loses its primary can use another primary

Modes of synchronization
Multicast:
A server within a high speed LAN multicasts time to others which set clocks assuming some delay (not very accurate)

Procedure call:
A server accepts requests from other computers (like Cristian’s algorithm). Higher accuracy. Useful if no hardware multicast.

Symmetric:

Pairs of servers exchange messages containing time information

Used where very high accuracies are needed (e.g., for higher levels)

Messages between a pair of NTP peers

All modes use UDP transport protocol for the message exchange

Server B

\[ T1 \rightarrow T2 \]

Server A

\[ T0 \rightarrow \text{m} \rightarrow \text{m'} \rightarrow T3 \]

Each message bears timestamps of recent events:

Local times of Send and Receive of previous message

Local time of Send of current message

Recipient (Server A) notes the time of receipt T3 (we have T0, T1, T2, T3).

In symmetric mode there can be a non-negligible delay between messages

Accuracy of NTP

For each pair of messages between two servers, NTP estimates an offset \( \epsilon \) between the two clocks and a round-trip delay \( d \) (total transmission time for the two messages m and m', which take \( t \) and \( t' \))

\[ T_{12} = T_{13} + t + \epsilon \quad \text{and} \quad T_{21} = T_{14} + t' + \epsilon \]

This gives us the delay (by adding the equations)

\[ d = (t + t') - (T_{12} - T_{21}) = (T_{12} - T_{13} + T_{21} - T_{14}) \]

Also the offset (by subtracting the equations)

\[ \epsilon = (t - t')/2 \]

\[ \text{Estimate of offset} \]

Using the fact that \( t, t' > 0 \) it can be shown that

\[ \epsilon < \frac{d}{2} \]

Thus \( \epsilon \) is an estimate of the offset and \( d \) is a measure of the accuracy

NTP servers filter pairs \( < \epsilon, d > \)
retains the 8 most recent pairs
estimates the offset \( \epsilon \)

NTP applies peer-selection to identify peer for reliability estimate.

Accuracy

over Internet: tens of ms

over a LAN: 1 ms
\[ T_{12} = T_{3} + t + 0 \text{ and } T_{1} = T_{3} + t - 0 \]

This gives us the delay (by adding the equations)

\[ 0 = t + t = T_{2} \cdot T_{3} + T_{1} \cdot T_{3} \]

Also the offset (by subtracting the equations)

\[ 0 = o + (t - t)/2, \text{ where } o = (T_{2} - T_{3} + T_{3} - T_{1})/2 \]

**Estimate of offset**

Using the fact that \( t, t' > 0 \) it can be shown that

\[ 0 - \frac{d}{2} \leq o \leq 0 + \frac{d}{2} \]

Thus \( o \) is an estimate of the offset and \( d \) is a measure of the accuracy.

NTP servers filter pairs \( <o, d> \)
retains the 8 most recent pairs
estimates the offset \( o \)