Script **generated by TTT**

Title: Distributed_Applications (20.05.2014)

Date: Tue May 20 14:31:09 CEST 2014

Duration: 91:29 min

Pages: 28

External data representation

Heterogeneous environment means different data representations

⇒ requirement to enable data transformation.

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

XML as common data representation

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

primitive datatypes → XSD equivalent

boolean, byte, unsignedShort (used for char), int, long, float, string, ...

[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 API for creating custom mapping.

e.g. writeSchema to specify an XML schema definition

high	Application specific data encoding language	XML
↑	General data encoding language	ASN.1
low	Network data encoding language	Sun XDR

Level of Abstraction

Example: primitive datatypes

Request for the invocation of the Java method

```
echoString("cat")
```

SOAP body of request that sends a string.

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

XML as common data representation

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

- primitive datatypes → XSD equivalent
  - boolean, byte, unsignedShort (used for char), int, long, float, string, ...
- **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 API for creating custom mapping.
  - e.g. writeSchema to specify an XML schema definition

high	Application specific data encoding language	XML
↑	General data encoding language	ASN.1
low	Network data encoding language	Sun XDR

Level of Abstraction

Generated by Targeteam

Example: array datatype

Request for Java method invocation

```
echoInts([1, 2, 3])
```

SOAP body of request that sends an array.

```
<soap:Body>
  <n:echoInts
  xmlns:n='http://tempuri.org/mapping.server.Array'>
  <ints href='#id0'>
  </n:echoInts>
  <id0 id='id0'
     soapenc:root='0'
     xsi:type='soapenc:Array'
     soapenc:ArrayType='xsd:int[3]'>
     <i xsi:type='xsd:int'>1</i>
     <i xsi:type='xsd:int'>2</i>
     <i xsi:type='xsd:int'>3</i>

  </id0>
</soap:Body>
```

Generated by Targeteam

XML as common data representation

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

- primitive datatypes → XSD equivalent
  - boolean, byte, unsignedShort (used for char), int, long, float, string, ...
- **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 API for creating custom mapping.
  - e.g. writeSchema to specify an XML schema definition

high	Application specific data encoding language	XML
↑	General data encoding language	ASN.1
low	Network data encoding language	Sun XDR

Level of Abstraction

Generated by Targeteam

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
- 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

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

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
```

example 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

to serialize an object

create an instance of the class `ObjectOutputStream` and invoke `writeObject`

to deserialize an object

open an `ObjectInputStream` use its `readObject` method to reconstruct the original object.

External data representation

Heterogeneous environment means different data representations

⇒ requirement to enable data transformation.

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

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?

Time

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

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

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

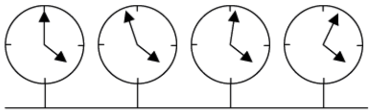
   $C_i (t)= a H_i (t) + b$

   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_i

Skew between clocks



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.

Coordinated Universal Time (UTC)

International Atomic Time is based on very accurate physical clocks (drift rate 10^{-13}).

- 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 millisecond
- Signals from GPS are accurate to about 1 microsecond

Time

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

Synchronizing physical clocks

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

Clock correctness

Synchronization in a synchronous system

Cristian's method for an asynchronous system

Network Time Protocol (NTP)

Precision Time Protocol (PTP)

External - internal synchronization

External synchronization

- 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.

Clock correctness

A ***hardware clock H*** is said to be correct if its drift rate is within a bound q > 0. (e.g. 10^{-6} secs/ 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 $C_i(t) = a H_i(t) + b$
- 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.

Synchronization in a synchronous system

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 p1 sends its local time t to process p2 in a message m

p2 could set its clock to $t + T_{trans}$ where T_{trans} is the time to transmit m

T_{trans} is unknown but

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

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

$t + (max - min)/2$

then skew $\leq u/2$.

Synchronizing physical clocks

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

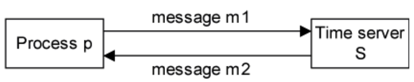
Approach

A time server S receives signals from a UTC source

- Process p requests time in m1 and receives t in m2 from S.
- p sets its clock to: $t + T_{round}/2$

Accuracy is +/- $(T_{round}/2 - min)$.

- because the earliest time S puts t in message m2 is min after p sent m1: t + min
- the latest time was min before m2 arrived at p: $t + T_{round} - min$
- the time by S's clock when reply message m2 arrives is in the range $[t + min, t + T_{round} - min]$



Discussion

The approach has several problems. It is only suitable for deterministic LAN environment or Intranet.

- a single time server might fail
  - $\Rightarrow$ redundancy through group of servers, multicast requests
- it does not deal with faulty time servers
  - how to decide if replies vary (byzantine agreement problems)
- imposter providing false clock readings

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

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

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:

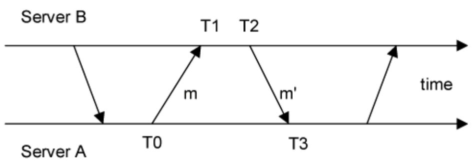
- A server accepts requests from other computers (like Cristian's algorithm). Higher accuracy. Useful if no hardware multicast.

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



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 o between the two clocks and a round-trip delay d_i (total transmission time for the two messages m and m', which take t and t')

$T_{i-2} = T_{i-3} + t + o$ and $T_i = T_{i-1} + t' - o$

This gives us the delay (by adding the equations)

$d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$

Also the offset (by subtracting the equations)

$o = o_i + (t' - t)/2$, where $o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2$

Estimate of offset

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

- $o_i - d_i/2 \le o \le o_i + d_i/2$.
- Thus o_i is an estimate of the offset and d_i is a measure of the accuracy

NTP servers filter pairs $< o_i , d_i >$

- retains the 8 most recent pairs
- estimates the offset o

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

Accuracy

- over Internet: tens of ms
- over a LAN: 1 ms

Accuracy of NTP

$T_{i-2} = T_{i-3} + t + o$ and $T_i = T_{i-1} + t' - o$

This gives us the delay (by adding the equations)

$d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$

Also the offset (by subtracting the equations)

$o = o_i + (t' - t)/2$, where $o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2$

Estimate of offset

Using the fact that $t, t' > 0$ it can be shown that

$o_i - d_i/2 \leq o \leq o_i + d_i/2$.

Thus o_i is an estimate of the offset and d_i is a measure of the accuracy

NTP servers filter pairs $<o_i , d_i>$

retains the 8 most recent pairs

estimates the offset o