Script generated by TTT

Title: Distributed_Applications (26.05.2014)
Date: Mon May 26 09:16:19 CEST 2014
Duration: 47:41 min
Pages: 12

PTP is a protocol to synchronize clocks throughout a computer network. NTP is typically used over the Internet handling large amounts of nondeterministic delays; accuracy in the message. Precision Time Protocol (PTP) is designed for LANs achieving clock accuracy in the sub-microsecond range.

**Synchronization Message Exchange**

PTP supports an algorithm to perform a distributed selection of the best candidate clock.

---

**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)

PTP defines a master-slave hierarchy.

Master

T1

Sync

Follow-Up

Delay

request

Delay

response

Slave

T2

T3

T4

timestamps

known by slave

T2

T1

T3

T1, T3, T4

If d is the transit time for the Sync message and o the constant offset between master and slave clocks:

\[ T2 \cdot T1 = o + d \text{ and } T4 \cdot T3 = o + d \]

\[ o = (T2 \cdot T1 - T4 + T3)/2 \]
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 \( \Rightarrow \) 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

**Rules for “happened-before” after Lamport**

In order to guarantee consistent states among the communicating components, the messages must be delivered in the correct order. The happened-before relation after Lamport may help to determine a message sequence for a distributed application.

The following rules apply:

1. Events within a component are ordered with respect to the before-relation, i.e. \( a \rightarrow b \)

   - if \( a \) is a send event of component TK1, and \( b \) the respective receive event of component TK2, then \( a \rightarrow b \)
   - if \( a \rightarrow b \) and \( b \rightarrow c \), then \( a \rightarrow c \)
   - if \( \neg(a \rightarrow b) \) and \( \neg(b \rightarrow a) \), then \( a \parallel b \); i.e. a and b are concurrent, i.e. they are not ordered.

Utilization of logical clocks to determine the event sequence.

Let

- \( T \): a set of timestamps
- \( C: E \rightarrow T \): a mapping which assigns a timestamp to each event
  
  \( a \rightarrow b \Rightarrow C(a) < C(b) \)

If the reverse deduction is valid, too \( (\Rightarrow) \), then the clock is called strictly consistent.

**Distributed execution model**

Events

- Classes of events
- Rules for “happened-before” after Lamport

Ordering by logical clocks

Logical clocks based on scalar values

- Description
- Example

Logical clocks based on vectors

- Description
- Example for vector clocks

**Characteristics of vector clocks**

Each component manages the following information:

- its local logical clock \( lc \); \( lc \) determines the local progress with respect to occurring events.
- its view on the global logical clock \( gc \); the value of the local clock is determined according to the value of the global clock.

There exist functions for updating logical clocks in order to maintain consistency; the following two rules apply:

**Rules**

- Rule R1 specifies the update of the local clock \( lc \) when events occur.
- Rule R2 specifies the update of the global clock \( gc \).

1. **Sending event**: determine the current value of the local clock and attach it to the message.
2. **Receiving event**: the received clock value (attached to the message) is used to update the view on the global clock.
Events
- Classes of events
- Rules for "happened-before" after Lamport

Logical clocks based on scalar values
- Description
- Example

Logical clocks based on vectors
- Description
- Example for vector clocks
- Characteristics of vector clocks

The time is represented by n-dimensional vectors with positive integers. Each component TK_i manages its own vector v[i] [1...n]. The dimension n is determined by the number of components of the distributed application.

v[i] [j] is the local logical clock of TK_i.

v[i] [k] is the view of TK_i on the local logical clock of TK_k; it determines what TK_i knows about the progress of TK_k.

Example: v[i] [k] = y; i.e., according to the view of TK_k, TK_i has advanced to the state y, i.e., up to the event y.

The vector v[i] [1...n] represents the view of TK_i on the global time (i.e., the global execution progress for all components).

Execution of R1
v[i] [j] := v[i] [j] + Δ

Execution of R2
After receiving a message with vector v[i] from another component, the following actions are performed at the component TK_i:
- update the logical global time: 1 ≤ k ≤ n: v[i] [k] := max (v[i] [k], v[k(i))
- execute R1
- deliver message to the application process of component TK_i

The scalar clock mechanism defines a partial ordering on the occurring events. Scalar clocks are not strictly consistent, i.e., the following is not true: C(a) ≤ C(b) ⇒ a → b

Example for vector clocks
TK1
TK2
TK3

Optimization: omit vector timestamps when sending a burst of multicasts
⇒ missing timestamp means use values of previous vector timestamp and increment the sender's field only.
Comparison of two vector clocks (timestamps) \( v[h][1..n] \) and \( v[k][1..n] \):

\[
\begin{align*}
\forall h \leq v[k] & \implies \forall x : v[h][x] \leq v[k][x] \\
\forall h < v[k] & \implies \forall h \leq v[k] \land \exists x : v[h][x] < v[k][x] \\
\forall h \parallel v[k] & \implies \neg (v[h] < v[k]) \land \neg (v[k] < v[h])
\end{align*}
\]

Let \( a \) and \( b \) be events with timestamps (vector clocks) \( v[a] \) and \( v[b] \), then the following is true:

\[
\begin{align*}
& a \rightarrow b \implies v[a] < v[b] \\
& a \parallel b \implies v[a] \parallel v[b]
\end{align*}
\]

If \( a \) of \( T_k \) and \( b \) of \( T_l \) have been triggered, then the following is true:

\[
\begin{align*}
& a \rightarrow b \implies v[a][] < v[b][] \land v[a][] < v[b][] \\
& a \parallel b \implies v[a][] > v[b][] \land v[a][] < v[b][]
\end{align*}
\]

Vector clocks are strictly consistent.