Clock Synchronization

- Why do we need clock synchronization?
  - process coordination
    - e.g. real-time process control systems require that accurate timestamps be assigned to sensor values to aid in correct data interpretation.
  - performance monitoring
    - e.g. performance statistics based on elapsed time
  - deadline detection
    - e.g. determination if deadlines have been violated
  - distributed agreement
    - e.g. assumed loose synchronization for atomic broadcast

Clock Synchronization

- Clocks
  - atomic clocks
    - extremely accurate
    - but: too expensive, too big, too unreliable and complex
  - crystal oscillator
    - typical computer clock
    - small, cheap, simple
    - fairly reliable with fail rates of about 10^{-6}/h
    - accuracy of 10^{-5} to 10^{-6}
      - resulting in drifts of 1 to 10 μs/s
      - 3.6 to 36 ms/h
  - clocks based on power-line frequency
    - power grid in the Northern US typically drifts 4 to 6 seconds from real time over the course of an evening

Clock Synchronization

- Synchronization
  - external synchronization
    - maintain processor clock within some given maximum derivation from a time reference external to the system
  - internal synchronization
    - keep processor clocks synchronized within some maximum relative derivation of each other

Clock Synchronization

- Definitions
  - "Understanding Protocols for Byzantine Clock Synchronization", Fred Schneider, 87-859, Aug. 1987, CS Dept., Cornell University.
  - Real-Time (t)
    - unobservable -- can only be approximated
    - "observes" events of different processors
      - observes and records all events
      - all observation delays are identical, i.e. there is no time skew
      - all events are immediately time-stamped, i.e. there is no processing delay
  - Real-Clock $C_p(t)$ of processor *p*
    - function mapping real time *t* into a clock reading $C_p(t)$
    - thus $C_p(t)$ is the value of the clock in processor *p* at real time *t*

Clock Synchronization

- $C_p(t)$ is characterized by μ, ρ and κ
  - constant μ: defines the range of initial values
    - hardware initial value

$$0 \le C_p(0) \le \mu$$

  - constant κ is the interval between ticks, i.e. the length of a tick, also defining the granularity
  - constant ρ is the upper bound on the clock drift rate
  - thus at each tick a clock is incremented (advanced) by a varying real number value *v*, with

$$(1-\rho)\kappa \le v \le (1+\rho)\kappa$$

  - hardware rate

$$0 < (1-\rho) \le \frac{C_p(t+\kappa) - C_p(t)}{\kappa} \le (1+\rho) \text{ for } 0 \le t$$

Clock Synchronization

- physical clock
  - hardware clock
  - simple counter
  - constant rate (+/- ρ)
  - no correction mode (independent of protocol)
  - source of clock drift
- virtual clock $\hat{C}_p$
  - clock synchronization protocols implement virtual clocks $\hat{C}_p$ at each processor *p*.
  - can be started, stopped, corrected

Clock Synchronization

» mapping

$$t \rightarrow C_p(t) \rightarrow \hat{C}_p(t)$$

t: real time

$C_p(t)$: hardware clock reading at t

$\hat{C}_p(t)$: virtual clock reading at t

» Similarly to real clocks virtual clocks have
- $\hat{\mu}$ defines the range of initial values
- $\hat{\kappa}$ is the tick length, may be >> κ
- $\hat{\rho}$ is the upper bound on the clock drift rate

Clock Synchronization

- Objective
  - Virtual Synchronization
    » given two processors *p* and *q*

$$|\hat{C}_q(t) - \hat{C}_p(t)| \le \hat{\delta} \qquad (2.3)$$

» here $\hat{\delta}$ defines how close the virtual clocks are synchronized

  - Virtual Rate

$$0 < (1-\hat{\rho}) \le \frac{\hat{C}_p(t+\hat{\kappa}) - \hat{C}_p(t)}{\hat{\kappa}} \le (1+\hat{\rho}) \text{ for } 0 \le t \qquad (2.4)$$

Clock Synchronization

- Reliable Time Source (RTS)
  - a mechanism which periodically makes the "correct" time available to all processors so that all processors can adjust their local virtual clock to the RTS.
  - requirements
    - time is distributed frequently enough
    - processor clocks will not drift too far apart in the interval between adjustment
    - no processor has to adjust its clock by too much
    - the adjustment can be spread over the interval that precedes the next resynchronization
  - if these requirements can be met, a reliable time source can solve the synchronization problem.

Clock Synchronization

- RTS Definitions
  - RTS is periodic
    - sequence of events at real times

$$t_{RTS}^{1}, t_{RTS}^{2}, t_{RTS}^{3}, \ldots$$

  - Periods are stable within a fixed bound. i.e.

$$r_{\min} \le r_{period} \le r_{\max}$$

where $r_{\min}$ and $r_{\max}$ are constants

  - t_p^i is the real-time at which processor p detects event t_{RTS}^{i}

Clock Synchronization

- For a process to qualify as an RTS is must satisfy the following conditions:
  - RTS1:
    - *bounded period*: a reliable time source generates a sequence of events at real times such that

      $$(t_{RTS}^{1}=0)\wedge(\forall i:\ 0<i:\ r_{\min}\leq t_{RTS}^{i+1}-t_{RTS}^{i}\leq r_{\max})$$

    - *bounded reading*: the real time at which a processor *p* detects the event produced at t_{RTS}^{i} satisfies

      $$(t_{P}^{1}=0)\wedge(\forall i:\ 1\leq i:\ 0\leq t_{p}^{i}-t_{RTS}^{i}\leq\beta)$$

      where β is a constant.
    - the first part in the above terms indicate that protocol & clocks start at real time 0

Clock Synchronization

- RTS2:
  - Real Time Source provides a useful value to all non-faulty processors to be used for correction, i.e.

    At t_p^i processor *p* obtains a value V_p^i that can be used in adjusting $\hat{C}_p$ consistent with (2.3) and (2.4)

- Note: it is not implied that the correct time is always available to processors, but it is available periodically
- RTS1 and RTS2 can be easily implemented using a single clock, but this clock is a single-point-of-failure
  - RTS1 achieved by using individual processor clocks to signal periodic resynchronization event
  - RTS2 achieved by each processor producing fault-tolerant average, e.g. median value.

Clock Synchronization

- Resetting a virtual clock $\hat{C}_p$
  - Typically "adjusting" a clock can be thought of as "resetting".
  - At real time 0, a processor uses virtual clock $\hat{C}_p^1$ and starts a new virtual clock $\hat{C}_p^{i+1}$ at real time t_p^{i+1} after detecting t_{RTS}^{i+1}
  - Thus in interval $t_p^i \le t < t_p^{i+1}$ we have $\hat{C}_p(t) = \hat{C}_p^i(t)$

Clock Synchronization

- Implementing virtual clock $\hat{C}_p^i$
  - At processor *p* take hardware clock C_p and add adjustment value resulting from clock synchronization protocol, i.e.

$$\hat{C}_p^i(t) \equiv C_p(t) + FIX_p^i(C_p(t))$$

  - here $FIX_p^i(C_p(t)) = FIX_p^i(T)$ is a correction function.
    - note: *T* is the clock time, whereas *t* is the real-time
    - need to implement a "smooth" correction function to avoid big jumps in $\hat{C}_p^i$,i.e. to not violate the virtual rate (2.4)

Clock Synchronization

- $FIX_p^i(T)$ spreads any change in its correction to C_p over an *adjustment interval* (AI), or AI clock seconds.
- Let adj_p^i be the cumulative adjustment to implement $\hat{C}_p^i$ from C_p
- Then $adj_p^i - adj_p^{i-1}$ is the additional, incremental amount of correction added during period *i*.
- The resulting "gradual" correction function is

$$FIX_p^i(T) \equiv$$

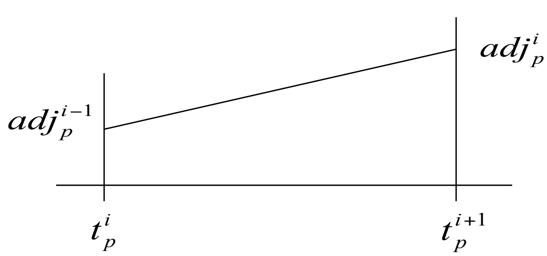
elapsed real clock reading since beginning of period

$$adj_p^{i-1} + \frac{(adj_p^i - adj_p^{i-1})(\min(C_p(t) - C_p(t_p^i), AI))}{AI}$$

previous accumulated correction

Clock Synchronization

- Effects of AI
  - If $AI \le \hat{\kappa}$ then *instantaneous resynchronization*
  - Else *continuous resynchronization*
  - $FIX_p^i(T)$ is a linear interpolation of the adj. function, and is a step-function if clock is discontinuous



Clock Synchronization

- Definition of *adj*. function
  - define a *convergence function* CF
  - then
    $adj_p^{i+1} = \mathrm{CF}[p, \hat{C}_1^i(t_p^{i+1}), \ldots, \hat{C}_N^i(t_p^{i+1})] - C_p(t_p^{i+1})$

    note that $\hat{C}_j^i(t_p^{i+1})$ is the virtual time when processor *p* recognizes t_{RTS}^{i+1}
  - Thus function adj_p^{i+1} gives the amount that $C_p(t_p^{i+1})$ differs from $\hat{C}_p(t_p^{i+1})$
  - Note that it is a function of other clock readings

Clock Synchronization

- Clock Synchronization Protocol

$i = 1; adj_p^0 = adj_p^1 = 0$

do forever

1) detect event generated at time t_{RTS}^{i+1};

$t_p^{i+1} =$ real time now

2) $adj_p^{i+1} = CF[p, \hat{C}_1(t_p^{i+1}), \ldots, \hat{C}_N(t_p^{i+1})] - C_p(t_p^{i+1})$

3) calculate $FIX_p^i(C_p(t))$ from adj_p^{i+1}

4) $i = i + 1$

end

Clock Synchronization

- Implementation Issues
  - step 1: how to "detect event generated at time t_{RTS}^{i+1} "
  - step 2: how does one processor read the virtual clocks at another processor
  - step 3: what is a valid CF function

Clock Synchronization

- Detecting Resynchronization Events (step 1)
  - detect event generated at time t_{RTS}^{i+1} by using our own approximately synchronized virtual clock
  - count to some predefined value *R*, i.e. when $\hat{C}_p^i = iR$ start next cycle
  - can be done using timer etc.
  - thus

t_p^{i+1} = time at which processor *p* starts its cycle

t_{RTS}^{i+1} = time at which *earliest* correct clock starts new cycle

Clock Synchronization

- since clock advances with $(1 \pm \hat{\rho})$ we get

$$r_{\min} = \frac{R}{1+\hat{\rho}} \qquad r_{\max} = \frac{R}{1-\hat{\rho}}$$

- recall from RTS1 that $0 \le (t_p^i - t_{RTS}^i) \le \beta$
- furthermore, recall from (2.3) that slowest clock lags fastest clock by at most $\hat{\delta}$
- then the slowest clock must reach *iR* no later than $\frac{\hat{\delta}}{1-\hat{\rho}}$
- thus

$$\beta = \frac{\hat{\delta}}{1-\hat{\rho}}$$

Clock Synchronization

- Reading other Clocks (step 2)
  - Using Correction Table
    - Processor *p* occasionally queries other processors, e.g. *q*
    - Processor *q* responds with time stamped message
    - Processor *p* maintains table $\tau_p^i[1,\ldots,N]$ where $\tau_p^i[q]$ is used to approximate $\hat{C}_q^i(t)$ i.e.

      $$\tau_p^i[q] = C - (C_p(t_{now}) - \Gamma_{\min})$$

      here C is $\hat{C}_q(t_{reply})$
      $\Gamma_{\min}$ is the minimum propagation delay
      $\Gamma_{\max}$ is defined respectively
    - Is assuming the minimum propagation delay realistic?

Clock Synchronization

- Clock Reading Error
  - Let $\lambda_p^i(q)$ be the error in p' s approximation of $\hat{C}_q^i(t)$
  - Then, assuming A is the maximum clock reading error

$$\lambda_p^i(q) \le$$
$$\Gamma_{\max} - \Gamma_{\min} + (\rho + \hat{\rho})(lread_p(q)) \le A$$

where $lread_p(q)$ is the time since $\tau_p^i[q]$ was logged, and A is the max clock reading error.

  - here $\Gamma_{\max} - \Gamma_{\min}$ is the dominating term w.r.t. reading error
  - therefore we focus on minimizing propagation and processing delays
- Note:
  - using periodic queries reduces number of messages by half, but can result in significant higher $\Gamma_{\max} - \Gamma_{\min}$

Clock Synchronization

- Convergence Functions (CF) (step 3)
  - Multiset
    - collection of objects similar in concept to a set
    - different from set in that not all elements need to be distinct
    - number of times a particular object (value) appears in a multiset is called the multiplicity of that object
  - Convergence Function arguments are:
    - processor evaluating CF
    - values x_q, $1 \le q \le N$ of values from processor q

Clock Synchronization

1) Monotonicity of CF

» given two multisets *X* and *Y* (monotonically non-decreasing)

» if $x_i \le y_i,\ \forall i,\ 1 \le i \le N$ implies

$$CF(p, x_1, x_2, \ldots, x_N) \le CF(p, y_1, y_2, \ldots, y_N)$$

2) Translation Invariance

» relative values matter (and not absolute values)

» thus

$$CF(p, x_1 + v, \ldots, x_N + v) = CF(p, x_1, \ldots, x_N) + v$$

» this allows comparison of values computed by CF at different times, i.e. values of CF are not affected by shift in time.

Clock Synchronization

3) Precision Enhancement Property

» require convergence

- Consider CF value of two processors *p* and *q* using at least *N-k* similar values. (*k* is the number of faulty elements).
- CF value of *p* and *q* should be closer than x_p and x_q were.

» property:

$$|CF(p, x_1, x_2, \ldots, x_N) - CF(q, y_1, y_2, \ldots, y_N)| \le \pi(\delta, \varepsilon)$$

if

- all non-faulty x_i are within δ from each other, i.e. $\delta = \max|x_i - x_j|$
- for corresponding y_i's $\delta = \max|y_i - y_j|$
  - (recall that δ is the max skew in reading of correct clocks)
- for each non-faulty pair $|x_i - y_i| \le \varepsilon$

» $\pi(\delta, \varepsilon)$ is called the *precision function*

$\pi(\delta, \varepsilon) < \delta \Rightarrow$ convergence

Clock Synchronization

- 4) Accuracy Preservation Property
  - this basically prevents big jumps
  - property:

$$|CF(p, x_1, x_2, \ldots, x_N) - x_p| \leq \alpha(\delta)$$

where $\alpha(\delta)$ is called the *accuracy function*

  - if

$$|adj_p^{i+1} - adj_p^i| \leq \alpha(\delta)$$

then the adjustment is bounded

Clock Synchronization

Examples of functions satisfying 4 conditions

- Egocentric Average:
  - take the average of all values that are no more than δ from x_p
  - note:
    - watch out, there are definitions of egocentric algorithms that replace all values not within the range with your own value.
- Example
  - $CF(p, x_1, x_2, \ldots, x_n)$ = CF(2,4,5,3,4,6,7,11,6,22,4,3,5,5)
  - assume δ = 3, here we have 13 values, $3k+1=N$, thus $k=4$
  - sorted multiset: {3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 7, 11, 22}
  - since $x_2 = 5$ we have to consider all values in the range [2,8]
  - CF = Ave(3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 7) = 4.73

Clock Synchronization

Examples of functions satisfying 4 conditions

- Fast Convergence Algorithm:
  - take average of all values that are within δ from at least *N-k* values
    - So the question to ask for each value is: *is the "neighborhood" of the value large enough, i.e., N-k, to be included?*
  - the degree *k* of fault tolerance is characterized by 3*k*+1=*N*,
  - δ is the range of values
- Example
  - CF(p,x_1,x_2,...,x_n) = CF(2,4,5,3,4,6,7,11,6,22,4)
  - assume δ = k = 3, then N-k = 10-3 = 7
  - sorted multiset: {3, 4, 4, 4, 5, 6, 6, 7, 11, 22}
  - ask: "is value *x* within 3 from at least 7 other values?"
    - e.g., value 4 results in interval [1,7]. Since there are 8 values in the interval value [4-3,4+3] = [1,7] is included.
  - CF = Ave(4, 4, 4, 5, 6, 6)

Clock Synchronization

Examples of functions satisfying 4 conditions

- Fault-tolerant Midpoint:
  - reduce *k* highest and lowest values and average both extreme values
    - Midpoint: (max_value + min_value) / 2 (after reduction of k extremes)
    - note this is **not** the median value in the sorted array of values!!!!
- Example
  - CF(p,x_1,x_2,...,x_n) = CF(2,4,5,3,4,6,7,11,6,22,4,3,5,5)
  - sorted multiset: {3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 7, 11, 22}
  - with k = 3 the reduced multiset is {4, 4, 5, 5, 5, 6, 6}
  - CF = (4+6)/2 = 5

Clock Synchronization

Examples of functions satisfying 4 conditions

- Fault-tolerant Average:
  - reduce *k* highest and lowest values and select average over all remaining
  - more general: MSR

- Example
  - CF(p,x_1,x_2,...,x_n) = CF(2,4,5,3,4,6,7,11,6,22,4,3,5,5)
  - sorted multiset: {3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 7, 11, 22}
  - with k = 3 the reduced multiset is {4, 4, 5, 5, 5, 6, 6}
  - CF = Ave(4, 4, 5, 5, 5, 6, 6) = 5