CS514: Intermediate Course in Operating Systems

Professor Ken Birman
Vivek Vishnumurthy: TA

Using real-time

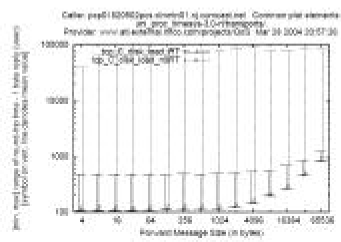
- Consider using a real-time operating system, clock synchronization algorithm, and to design protocols that exploit time
- Example: MARS system uses pairs of redundant processors to perform actions fault-tolerantly and meet deadlines. Has been applied in process control systems. (Another example: Delta-4)

Features of real-time operating systems

- The O/S itself tends to be rather simple
  - Big black boxes behave unpredictably
- They are structured in terms of “tasks”
  - A task is more or less a thread
  - But typically come with expected runtime, deadlines, priorities, “interruptability ”, etc
- User decomposes application into task-like component parts and then expresses goals in a form that RTOS can handle
- Widely used on things like medical devices

RTOS can be beneficial

- Lockheed Martin ATL timed CORBA method invocations
- Variation in response time was huge with a normal Linux OS
- When using a Timesys RTOS the variability is eliminated!



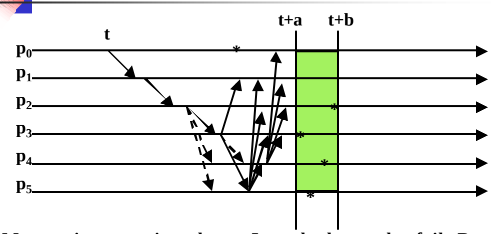
Next add distributed protocols

- Given some degree of real-time behavior in the platform…
- … goal is to offer distributed real-time abstractions programmers can use

Real-time broadcast protocols

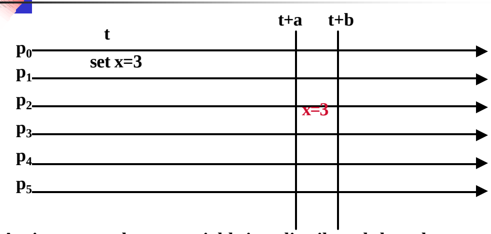
- Can also implement broadcast protocols that make direct use of temporal information
- Examples:
  - Broadcast that is delivered at same time by all correct processes (plus or minus the clock skew)
  - Distributed shared memory that is updated within a known maximum delay
  - Group of processes that can perform periodic actions

A real-time broadcast



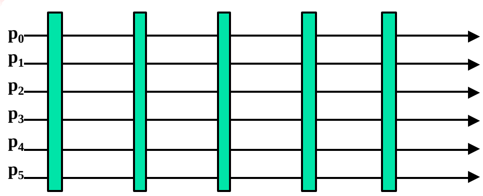
Message is sent at time *t* by p_0. Later both p_0 and p_1 fail. But message is still delivered atomically, after a bounded delay, and within a bounded interval of time (at non-faulty processes)

A real-time distributed shared memory



At time *t* p_0 updates a variable in a distributed shared memory. All correct processes observe the new value after a bounded delay, and within a bounded interval of time.

Periodic process group: Marzullo



Periodically, all members of a group take some action. Idea is to accomplish this with minimal communication

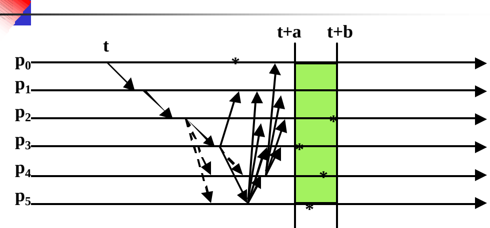
The CASD protocols

- Also known as the “Δ -T” protocols
- Developed by Cristian and others at IBM, was intended for use in the (ultimately, failed) FAA project
- Goal is to implement a timed atomic broadcast tolerant of Byzantine failures

Basic idea of the CASD protocols

- Assumes use of clock synchronization
- Sender timestamps message
- Recipients forward the message using a flooding technique (each echos the message to others)
- Wait until all correct processors have a copy, then deliver in unison (up to limits of the clock skew)

CASD picture



p_0, p_1 fail. Messages are lost when echoed by p_2, p_3

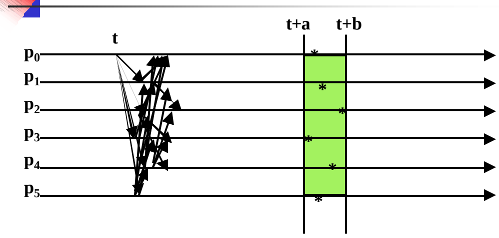
Idea of CASD

- Assume known limits on number of processes that fail during protocol, number of messages lost
- Using these and the temporal assumptions, deduce worst-case scenario
- Now now that if we wait long enough, all (or no) correct process will have the message
- Then schedule delivery using original time plus a delay computed from the worst-case assumptions

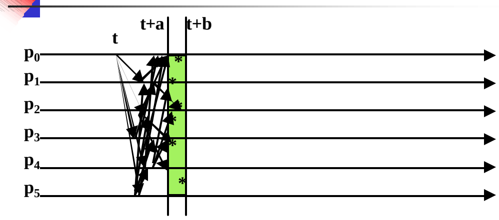
The problems with CASD

- In the usual case, nothing goes wrong, hence the delay can be very conservative
- Even if things do go wrong, is it right to assume that if a message needs between 0 and δms to make one hope, it needs [0,n* δ] to make n hops?
- How realistic is it to bound the number of failures expected during a run?

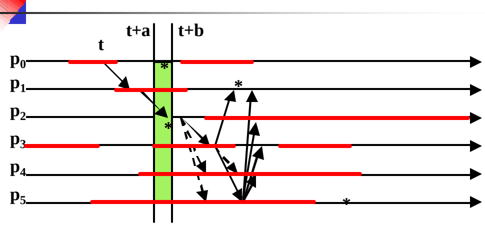
CASD in a more typical run



… leading developers to employ more aggressive parameter settings



CASD with over-aggressive paramter settings starts to “malfunction”



all processes look “incorrect” (red) from time to time

CASD “mile high”

- When run “slowly” protocol is like a real-time version of abcast
- When run “quickly” protocol starts to give probabilistic behavior:
  - If I am correct (and there is no way to know!) then I am guaranteed the properties of the protocol, but if not, I may deliver the wrong messages

How to repair CASD in this case?

- Gopal and Toueg developed an extension, but it slows the basic CASD protocol down, so it wouldn't be useful in the case where we want speed and also real-time guarantees
- Can argue that the best we can hope to do is to superimpose a process group mechanism over CASD (Verissimo and Almeida are looking at this).

Why worry?

- CASD can be used to implement a distributed shared memory ("delta-common storage")
- But when this is done, the memory consistency properties will be those of the CASD protocol itself
- If CASD protocol delivers different sets of messages to different processes, memory will become inconsistent

Why worry?

- In fact, we have seen that CASD can do just this, if the parameters are set aggressively
- Moreover, the problem is not detectable either by "technically faulty" processes or "correct" ones
- Thus, DSM can become inconsistent and we lack any obvious way to get it back into a consistent state

Using CASD in real environments

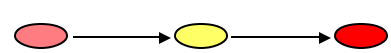
- Would probably need to set the parameters close to the range where CASD can malfunction, but rarely
- Hence would need to add a self-stabilization algorithm to restore consistent state of memory after it becomes inconsistent
- Problem has not been treated in papers on CASD
- pbcast protocol does this

Using CASD in real environments

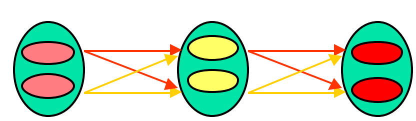
- Once we build the CASD mechanism how would we use it?
  - Could implement a shared memory
  - Or could use it to implement a real-time state machine replication scheme for processes
- US air traffic project adopted latter approach
  - But stumbled on many complexities…

Using CASD in real environments

- Pipelined computation



- Transformed computation



Issues?

- Could be quite slow if we use conservative parameter settings
- But with aggressive settings, either process could be deemed "faulty" by the protocol
  - If so, it might become inconsistent
    - Protocol guarantees don't apply
  - No obvious mechanism to reconcile states within the pair
- Method was used by IBM in a failed effort to build a new US Air Traffic Control system

Similar to MARS

- Research system done in Austria by Hermann Kopetz
  - Basic idea is that everything happens twice
  - Receiver can suppress duplicates but is guaranteed of at least one copy of each message
  - Used to overcome faults without loss of real-time guarantees
- MARS is used in the BMW but gets close to a hardware f.tol. scheme

Many more issues….

- What if a process starts to lag?
- What if applications aren't strictly deterministic?
- How should such a system be managed?
- How can a process be restarted?
  - If not, the system eventually shuts down!
- How to measure the timing behavior of components, including the network

FAA experience?

- It became too hard to work all of this out
- Then they tried a transactional approach, also had limited success
- Finally, they gave up!
  - $6B was lost…
  - A major fiasco, ATC is still a mess

Totem approach

- Start with extended virtual synchrony model
- Analysis used to prove real-time delivery properties
- Enables them to guarantee delivery within about 100-200ms on a standard broadcast LAN
- Contrast with our 85us latency for Horus!

Tradeoffs between consistency, time

- Notice that as we push CASD to run faster we lose consistency
- Contrast with our virtual synchrony protocols: they run as fast as they can (often, much faster than CASD when it is not malfunctioning) but don't guarantee real-time delivery

A puzzle

- Suppose that experiments show that 99.99% of Horus or Ensemble messages are delivered in 85us +/- 10us for some known maximum load
- Also have a theory that shows that 100% of Totem messages are delivered in about 150ms for reasonable assumptions
- And have the CASD protocols which work well with Δ around 250ms for similar LAN's

A puzzle

- Question: is there really a difference between these forms of guarantees?
- We saw that CASD is ultimately probabilistic. Since Totem makes assumptions, it is also, ultimately, probabilistic
- But the experimentally observed behavior of Horus is also probabilistic
- … so why isn't Horus a "real-time" system?

What does real-time mean?

- To the real-time community?
  - A system that provably achieves its deadlines under stated assumptions
  - Often achieved using delays!
- To the pragmatic community?
  - The system is fast enough to accomplish our goals
  - Experimentally, it never seems to lag behind or screw up

Some real-time issues

- Scheduling
  - Given goals, how should tasks be scheduled?
  - Periodic, a-periodic and completely ad-hoc tasks
- What should we do if a system misses its goals?
- How can we make components highly predictable in terms of their real-time performance profile?

Real-time today

- Slow transition
  - Older, special purpose operating systems and components, carefully hand-crafted for predictability
  - Newer systems are simply so fast (and can be dedicated to task) that what used to be hard is now easy
  - In effect, we no longer need to worry about real-time, in many cases, because our goals are so easily satisfied!