Fault Tolerant Real-Time Distributed Systems

István Majzik

Technical University of Budapest
Department of Measurement and Information Systems

September 14, 2000

Overview

- Basic definitions
- General structure
  - Why using distributed RT systems
  - Structural elements
  - Software requirements
- Fault tolerant real-time systems
  - Structural elements
  - System design
- Communication
  - Error detection
  - Implicit flow control
  - Explicit flow control
- Case study: The MARS system
  - TTP/C protocol layers
  - TTP/A protocol

Basic definitions

Real-time (RT) computer system: Correctness of the system depends not only on the logical (functional) results, but also on the timeliness, i.e. the physical time at which the results are presented.
RT system: changes its state as a function of physical time

General structure:
operator $\leftrightarrow$ RT computer system $\leftrightarrow$ controlled RT object
man-machine and *instrumentation* interface

General task: react to stimuli from the controlled object/operator request

Deadline: At which the results must be produced

- soft: result has utility after the deadline
- firm: no utility after the deadline
- hard: catastrophic consequences if the deadline is missed

Basic definitions (continued)

Hard RT system: at least one hard deadline exists

- required: GUARANTEED temporal behavior under all specified LOAD and FAULT conditions

Comparison of hard and soft RT systems:

Hard RT	**Soft RT**
hard deadline	soft deadline
predictable performance	degraded performance in peak load
synchronous with env.	computer control
often safety-critical	usually non-critical
active redundancy	checkpoint-recovery
short-term data integrity	long-term data integrity
autonomous error detection	user-assisted error detection

Basic definitions (continued)

Fail-safe vs. fail-operational systems:

- fail-safe: the controlled object has a safe state
  - (e.g. all traffic lights are red)
  - error $\rightarrow$ transition to safe state
  - computer systems: high error detection coverage required
- fail-operational: no safe state in the object
  - (e.g. airplane)
  - computer systems: minimal level; of service required to avoid catastrophe

Guaranteed response vs. best effort approach:

- guaranteed response: peek load and fault scenario – to be specified! (rigorous)
- best effort: hard to predict rare event scenarios (+ the design is often resource-inadequate)

Basic definitions (continued)

Event-triggered vs. time-triggered systems:

- event: any occurrence that happens in time
  - state change in the object/controller
- trigger: event that causes the start of an action
- event-triggered (ET) system:
  - all activities are initiated by events
  - interrupt-like mechanism
  - dynamic scheduling of activities (tasks)
- time-triggered (TT) systems:
  - activities are initiated by the progress of time
  - interrupt: clock only
  - (synchronized distributed clock)

System architecture

Nodes: functional+temporal properties

- mapping between nodes and functions
- node error $\rightarrow$ it is clear, which function is affected

Communication network:

- interface of nodes (CNI)
- event queue: often FIFO
- state information: overwriting old values

Composability

System properties follow from subsystem properties

- all subsystem combinations work properly

ET systems: not composable

- message overload to receivers, conflicts

TT systems: composable

- temporal control resides in the communication subsystem
- message scheduling tables are used
- transfer happens at predefined time points
- computer and communication subsystem’s properties are isolated

Scalability

No limits on the extensibility of the system

- nodes can be added (up to communication capacity)
- clusters, gateways can be established

Controlling complexity:

- partitioning into subsystems
- preservation of abstractions (hierarchy) in the case of faults
- strict control over interactions (interfaces)

Dependability

Responsive systems:

- RT performance + fault tolerance + distribution of functions

In distributed system:

- error containment (EC) regions:
  - fault detected + corrected/masked before corrupting the rest of the system
  - error is detected at the service interface
  - nodes: often EC regions
- node failure modes:
  - fail-stop
  - fail-silent
  - crash
- active replication: deterministic behavior (replica determinism)
- subsystems of different criticality (critical and non-critical parts) in different EC regions

Modeling RT systems

Assumptions used:

- load hypothesis
- fault hypothesis

Timing properties:

- actual, minimal duration of actions
- worst-case execution time (WCET)
- jitter

Structural elements

- task: sequential program execution
  - simple: no synchronization
  - complex: synchronization (blocking may occur)
- node: self-contained unit with well-defined function
  - abstraction: hardware+software into a single unit
  - SRU: smallest replaceable unit
- FTU: fault tolerant unit
  - set of replicated nodes + adjudicator
- computational cluster:
  - set of FTU-s (+ gateways)
- interfaces:
  - control+data+temporal properties
  - functional intent

RT software

ET systems: interrupts

- as CPU interrupt frequency increases, also the time
- with housekeeping (wasted, overhead) increases

TT systems:

- ”sampling” the input
- predictable overhead

Analysis of the system behavior

- Worst-case execution time (WCET should) be known a priori
  - simple task:
    - source code analysis → critical path – dynamic code? (recursion, loops,...)
    - compiler analysis (timing tree: execution time of high-level constructs)
    - microarchitecture: pipeline, cache?
  - complex task:
    - global problems in the system
    - preemption+blocking → full system model is required
    - solution: annotated source code + instrumentation
- H-state (history state) analysis:
  - data that undergoes changes as computation progresses
  - fault → error (changes in state)
  - ideal: stateless system (easy to recover)
  - cyclic computation: no state transition among cycles

Special properties of fault-tolerant RT systems

- permanence:
  - a message is permanent if there are no predecessors which may arrive
- idempotency:
  - effect of receiving more copies (of the same message) is the same as receiving a single copy
  - replica management is easier
- replica determinism:
  - all members have the same visible h-state in time points that are at most an interval of d units apart (d unit: replace a missing message or erroneous message)
  - required to:
    * system test
    * FT by replication. E.g. no replica determinism:
      node1: commit, go; node2: abort, stop; node3: abort, go (erroneous)
      decision: abort, go (inconsistent)

Special properties of fault-tolerant RT systems (continued)

- causes of replica nondeterminism:
  - different inputs (digitalization, sensor characteristics)
  - different computational progress relative to physical time (CPU clock drift, FT instruction retry mechanism)
  - preemptive scheduling
  - race conditions
- solutions:
  - sparse time base (no local clock, event is assigned to the same clock tick)
  - agreement on inputs
  - static control structure (no non-deterministic language)
  - deterministic algorithms (no preemption, deterministic race)

Architectural elements: Node

A node should display simple failure modes (fail-silent, fail-stop)
Re-integration of a node:

- minimal h-state is required to speed up re-integration
  - backward recovery: checkpoint may be invalid due to elapsed time (e.g. sensor data age invalidation)
  - checkpointing mechanism is not enough
- ideal re-integration points:
  - after the completion of component cycle
  - after the commit of atomic operations
- h-state restoration:
  - retrieve input data from environment (sensors, semaphores etc.)
  - restart vector: control output of the node to synchronize the environment (e.g. yellow traffic lights)
    restart vector is defined at development time
  - request data from operator or neighbors

Architectural elements: FT unit

- fail-silent nodes: duplication
- value errors: replication (TMR)
- Byzantine failures: 4 nodes required for a FT unit
- Service required: membership (with short latency)
  - ET systems: silence of a node: failure or there is no event?
    heartbeat protocol is needed
  - TT systems: periodic message sending defined as membership points

Software issues

What to do to increase dependability:

- clean structure: simple paradigm (structured programming)
- formal methods: specification and verification
- FT schemes: diverse versions of software

Approaches:

- independent monitoring
  Case study 1: VOTRICS tram signaling system
- minimal safe service
  Case study 2: Airbus fly by wire

Case studies

- Case study 1: VOTRICS tram signaling system
  - subsystem1:
    - collecting track data + operator data
    - calculating switch (actuator) positions
    - TMR architecture
  - subsystem2: safety bag
    - monitors the safe state of the system
    - evaluates safety predicates (rule book is given)
    - $\rightarrow$ blocking output of unsafe signals
    - TMR architecture
  - Advantages:
    - independent specifications
    - independent implementations (standard program + expert system)
- Case study 2: Airbus fly by wire
  - higher level subsystem: full functionality + error detection
  - lower level subsystem: reduced but safe functionality

Real-time operating systems

To do: task management + scheduling + communication + time management

Error detection:

- monitoring task execution times
  - does not end in WCET $\rightarrow$ error
- monitoring interrupts:
  - minimal inter-arrival time must be enforced
- replica management:
  - double execution of tasks $\leftarrow$ specified in design time
- watchdog functions:
  - heartbeat of the node (in the case of fail-silent nodes)
- challenge-response protocol
  - calculation of response patterns

Common problems

- flexibility $\leftrightarrow$ error detection
  - error detection requires a priori knowledge of the error-free behavior
  - ”partial” restriction is needed: e.g. heartbeat
  - or replication (deterministic!)
- sporadic data $\leftrightarrow$ periodic data
  - sporadic: dynamic schedule fits to it
  - periodic: conflict-free (static) schedule
- single locus of control $\leftrightarrow$ fault tolerance, robustness
  - single locus: e.g. token in a token ring
  - FT: additional mechanism is required (e.g. token recovery)
- probabilistic access $\leftrightarrow$ replica determinism
  - probabilistic: e.g. Ethernet collision resolution
  - replica: identical behavior is required

System design: Requirement analysis

Acceptance test to each requirement:

- performance, deadlines
- dependability
- cost

System design: Decomposition

Horizontal structuring: layering (centralized systems)

- stepwise abstraction
  $\rightarrow$ faults: exception handling

Vertical structuring: partitioning (distributed systems)

- nearly independent subsystems
- interfaces among the components (low external connectivity)
  $\rightarrow$ faults: error-containment regions
- (partitioning to be kept even in the case of faults!)

Detailed design and implementation:

- scheduling,
- determining I/O tasks etc.

System design: Test of the design

- functional coherence:
  - node = self-contained function
  - minimum h-state
  - error recovery of nodes
  - data sharing interfaces (no control signals)
  - timing
- testability
  - message interface: all properties should be defined (worst-case scenario)
  - h-state observation: modification possibility
  - replica determinism (input$\rightarrow$output determinism)
  - how to test FT properties?
  - built-in self-test

Test of the design (continued)

- dependability
  - node failure $\rightarrow$ cluster computation effects (performance + timeliness)
  - maintaining a safe state in the node
  - if the communication subsystem fails
  - detection of a node failure externally?
  - internal node error detection $\rightarrow$ fail-silence?
  - node recovery: time; single failure only
  - safety critical functions: in different ECR (err. containment. region)
- physical characteristics
  - mechanical interfaces = SRU boundaries = diagnostic boundaries
  - SRUs of a FTU are mounted at different locations, avoiding common mode external failures (e.g. mechanical damage)
  - SRUs of an FTU should have different power sources, grounding (common mode failures)
  - EMI effects via the cabling
  - environmental conditions (temperature, shock)

Communication: Requirements

- low protocol latency (standard: multicast network)
- minimal jitter (e.g. time redundancy)
- composability (independent evaluation)
- fast error detection
  - blackout: correlated mutilation of all messages (e.g. lightning)
  - babbling idiot: sending messages at wrong moments (TT systems: message exchange at predefined points)
  - lost channel: safe state of a node required
  - node error: membership service required (e.g. heartbeat)
  - trashing: too much messages causing breakdown (if the number of messages increases then the throughput will drastically decrease after a given point)
  - end-to-end acknowledgements
    - e.g. message from node A to an actuator, acknowledge from a sensor to node B, acknowledge from node B to node A
    - e.g. Three Mile Island nuclear reactor: valve was not closed, but the monitoring light was green, ”never trust an actuator”

Communication: Requirements (continued)

- Physical structure: multicast is required; bus vs. ring
  - bus: simultaneous arrival of messages
    resilience to fail-silent node failures
  - ring: optical fibers

Flow control:
Controlling the speed of information exchange (receiver can keep up with the sender)

- Explicit flow control
- Implicit flow control

Communication: Explicit flow control

- Receiver sends acknowledgements: previous message arrived, ready to get the new one
- Receiver decides the rate of transmission
- Example: PAR (Positive Ack or Retransmission) protocol
- Properties:
  - timeout $\rightarrow$ delay may be long!
  - error detection by sender

Explicit flow control: The PAR protocol

1. sender (is asked) to send a message
   retry count=0
   timeout reset
   sending the message
2. receiver gets a message:
   was it already sent?
   not $\rightarrow$ send ack to sender
   yes $\rightarrow$ send ack + skip message
3. sender receives ack $\rightarrow$ terminates
   no ack in timeout period: check retry count
   exceeded $\rightarrow$ abort
   not exceeded $\rightarrow$ increment retry count
   reset timeout
   re-send message

Communication: Implicit flow control

- sender and receiver agree a priori on the message send times
- global time base is required
- sender sends at the predefined point
- receiver accepts all messages
  - no acknowledgements
  - missing messages are detected by the receiver
- FT: active redundancy (multiple messages, multicast)
- no trashing (no dynamic scheduling, re-send)

Comparison of explicit and implicit flow control

Property	**Explicit flow control**	**Implicit flow control**	**Hard RT requirement**
control	receiver controls	time controls	receiver may not control
error detection	at the send	at the receiver	at the receiver
trashing	yes	no	to avoid
multicast	difficult	yes	required

Hard RT: Implicit flow control improves predictability

Problem in explicit flow control:
event shower $\rightarrow$ overload $\rightarrow$ catastrophic

Communication architecture

- backbone network connecting nodes to non-critical clients (reports etc.)
- RT network: connecting the nodes
  - predictable message transmission required
  - support for FT: replicated nodes and channels
  - membership service

    $\rightarrow$ duplicated channels without SPOF (single point of failure)

    $\rightarrow$ babbling detection
- field bus: connecting individual nodes to their sensors/actuators
  - periodic transfer
  - strict RT
  - FT is not included in the bus
    - (dependability bottleneck is the sensor/actuator;
    - it is duplicated, connected to different nodes)

Case study: The MARS system

Maintainable Real-Time System (MARS, TU Wien, 1979-)

Project goals:

- distributed FT architecture
- hard RT
- nodes: single-chip communication interface, fail silence
- FT properties: replication (FTU)
- TT (time-triggered) architecture
- Global time base: FT, distributed clock synchronization (VLSI chip)

MARS: Distributed RT system

- Cluster - FTU - node - task
- Communication:
  - field bus: TTP/A protocol
  - RT bus: TTP/C protocol: membership, redundancy management
  - backbone: TCP/IP
- Node: host computer + communication controller
  - active: produces images, has bus slot, membership
  - passive: reads only, no bus slot, no membership
- Fail silence of nodes:
  - HEDC (High Error Detection Coverage) mode, transparent in OS
  - duplicate execution of application tasks
  - end-to-end CRC of messages
  - end-to-end CRC of task execution (similar to WP)
  - difference: messages are not sent; replicated node is switched

Hardware building blocks: The TTP controller

- components: dual-port RAM + controllers + EPROM (MEDL)
- connected to replicated buses
- commercial elements are used (COTS)
- TTP/A: four buses
- each node: two communication controllers
  - TTP/A + TTP/C
  - TTP/C + TTP/C
  - TTP/C + TCP/IP

Software support: Distributed and local services

- distributed services:
  - clock synchronization
  - membership
  - redundancy management
- local services:
  - static schedule (WCET, WCAO [administration overhead])
  - information transfer: communication controllers $\leftrightarrow$ tasks
  - High Error Detection Coverage (HEDC) mode
- cluster compiler:
  - static, deterministic schedule
  - generating message schedule (MEDL: message descriptor list)
  - inputs:
    data elements (length)
    update period + temporal accuracy
    sender and receiver ID
    redundancy strategy (replication: 2,3,...)

Fault tolerance

- fail-silent nodes
- FTU by replication
  - deterministic message transfer
  - deterministic node operation: static schedule
- redundant sensors: master-checker configuration
  - node1: master of field_bus1, listen to field_bus2
  - node2: master of field_bus2, listen to field_bus1
  - agreement is required

TTP/C Protocol layers

- data link/physical:
  - bit encoding
  - bit synchronization
  - media access
- SRU layer:
  - implicit acknowledgement
  - clock synchronization
  - SRU membership
- RM (redundancy management) layer
  - redundancy management
  - start-up

TTP/C Protocol layers (continued)

Layers above the Basic Communication Interface:

- FTU layer:
  - permanence of messages
  - FTU membership
- Host layer:
  - application membership
  - application software

TTP/C Data link layer

- media access: TDMA (time division multiple access)
- controlled by: MEDL (message descriptor list)
  - what point in time send a message
  - what point in time receive a message
  - contains:
    SRU slot (time), address, direction (I/O),
    length of message, type of message, parameters
- cluster cycle:
  - given number of slots
  - every node is assigned a slot
  - no data $\rightarrow$ empty slot
  - cluster cycle = sequence of all TDMA rounds
  - TDMA round: k nodes using k frames

TTP/C SRU layer

- Data frames to communication interface RAM
- membership:
  - every slot is a membership point
  - ”I am alive” without additional overhead
  - delay: 1 TDMA round
  - membership info: bit vector
- clock synchronization:
  - based on a priori known arrival times of messages
  - deviation between expected and real times: clock to be synchronized
  - no overhead of checking synchronization
- acknowledgement:
  - implicit flow control (based on membership)

TTP/C RM layer

Redundancy management and mode changes:

- shadow node $\rightarrow$ active node
  if the current active node fails
- active node $\rightarrow$ shadow node
  if the active node fails $\rightarrow$ fail silence!
- ”repaired node” $\rightarrow$ shadow node

TTP/C FTU layer

Permanence of messages

FTU membership: configurations

- replicated fail-silent nodes
- TMR

TTP/A protocol

Properties:

- standard serial communication (UART)
- master: node connected to the field bus
- slaves: sensors/actuators

Rounds:

- firework message from the master: one-byte synchronization event + round type determinator
- (synchronization) gap
- message bytes: determined by MEDL

TTP/A Error detection

- time-out for firework message:
  backup (shadow) node takes the role of the master (active)
- time-out of a data byte: local time-out
  error is reported to the host
- data byte outside the specified time window:
  - termination of the round
  - waiting for a new firework message
  - no firework $\rightarrow$ backup node will be active
- parity checks (UART)