TECHNISCHE UNIVERSITÄT WIEN

TTTech

Dependable Computer Systems

Part 6: System aspects of dependable computers

Contents

- System design considerations
- Fault-tolerance: systematic vs. application-specific
- The problem of *Replica Determinism*
- Services for replicated fault-tolerant systems
  - Basic Services
  - Clock Synchronization Services
  - Communication Services
  - Replica Control Services

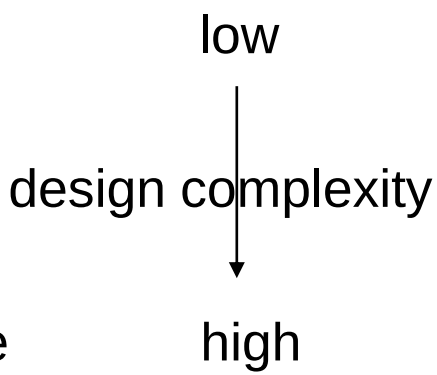
System design considerations

System design considerations

Fault-tolerance is not the only means for dependability

To achieve given dependability goals the following aspects need consideration
(usually in the order given here):

- perfection
- maintenance
- fault-tolerance
  - systematic fault-tolerance
  - application-specific fault-tolerance



Perfection vs. fault-tolerance

Perfection is easier than fault-tolerance:

- if it is possible to attain a given dependability goal by means of perfection then use perfection in favor of fault-tolerance
- perfection leads to conceptual simpler systems
- lower probability of design faults
- does not require error detection, damage confinement and assessment, error recovery and fault treatment to tolerate faults
- steady reliability improvement of hardware components supports perfection

But, perfection is limited:

- perfection is limited by the dependability of individual components
- very high dependability goals can only be reached by maintenance or by fault-tolerant systems

Maintenance vs. fault-tolerance

Maintenance is easier than fault-tolerance:

- if it is possible to attain a given dependability goal (availability) by means of maintenance then use maintenance in favor of fault-tolerance
- maintenance adds to system complexity, but is still considerable simpler than fault-tolerance
- lower probability of design faults
- requires error detection and damage confinement, but no error recovery and fault treatment at system level
- also trade off between maintenance and reliability (connector vs. solder joint)

But, maintenance is limited:

- maintenance is limited by the dependability of individual components
- applicability of maintenance is limited (cf. next slide)

Limitations of maintenance

- maintenance (without fault-tolerance) is only applicable if system down times are permitted
  - ail-stop or fail-safe systems allow down times:
    (train signalling, anti-lock braking system, …)
  - fail-operational systems do not allow down times:
    (fly-by-wire, reactor safety system, …)
- only restricted reliability and safety improvements by preventive maintenance
- preventive maintenance is only reasonable if:
  - replacement units have constant or increasing failure rate
  - nfant mortality is well controlled and failure rates are sufficiently low

The maintenance procedure

The maintenance procedure consists of the following phases:

- error detection
- call for maintenance
- maintenance personnel arrival
- diagnosis
- supply of spare parts
- replacement of defect components
- system test
- system re-initialization
- resynchronization with environment

Aspects of maintenance

- maintenance costs vs. system costs
- error latency period, error propagation and error diagnosis
- maintenance personnel
  (number, education, equipment, location, service hours, etc.)
- spare part supply, stock or shipment
- maintainability
  - documentation
  - test plans
  - system structure
  - error messages
  - size and interconnection of replacement units
  - accessibility of replacement units
  - mechanical stability of replacement units

Determining factors for SRU size

The size of the smallest replaceable unit (SRU) is determined by the following factors:

factor (increases)	*SRU size*
qualification of service personnel	decreases
effort for diagnosis	decreases
cost of SRU	increases
spare part costs[1]	increases
maintainability	increases
maintenance duration	decreases

[1]Cost for parts which are used to construct SRU's

Diagnosis support for maintenance

- diagnosis support is very important
- self diagnosis with meaningful messages
- complete coverage of error domain
- maintenance documentation:
  - symptom → cause and affected SRU
  - error symptom/cause matrix indicates for each symptom all possible SRU's that may cause the symptom
  - sparse matrices indicate good diagnosability
- expert system support for diagnosis
- duration of diagnosis is important for MTTR
- needs to be considered during system design

Fault-tolerance: systematic vs. application-specific

Application-specific fault-tolerance

- the computer system interacts with some physical process, the behavior of the process is constrained by the law of physics
- these laws are implemented by the computer system to check its state for reasonableness
- for example:
  - the acceleration/deceleration rate of an engine is constrained by the mass
    and the momentum that affects the axle
  - signal range checks for analog input signals
- reasonableness checks are based on application knowledge
- fail-stop behavior can be implemented based on reasonableness checks

Application-specific fault-tolerance

- the laws of physics constraining the process can be used to perform state estimations in case some component has failed
- for example:
  - if the engine temperature sensor fails a simple state estimation could assume a default value
  - a better state estimation can be based on the ambient temperature of the engine, engine load and thermostatical behavior of the engine
- the speed of a vehicle can be estimated if the engine speed and the transmission ratio is known
- state estimations are based on application knowledge
- fail-operational behavior can be implemented based on reasonableness checks and state estimations

Systematic fault-tolerance

- does not use application knowledge, makes no assumptions on the physical process or controlled object
- uses replicated components instead
- if among a set of replicated components, some—but not all—fail then there will be divergence among replicas
- information on divergence is used for fault detection
- replicas are therefore required to deliver corresponding results in the absence of faults
- The problem of replica determinism:
  due to the limited accuracy of any sensor that maps continuous quantities onto computer representable discrete numbers it is impossible to avoid nondeterministic behavior

Systematic fault-tolerance (cont.)

- systematic fault-tolerance requires agreement protocols due to replica nondeterminism
- the agreement protocol has to guarantee that correct replicas return corresponding results
  (the problem of replica determinism is discussed later)
- fail-stop behavior can be implemented by using the information of divergent results
- fail-operational behavior can be implemented by using redundant components

Comparison of fault-tolerance techniques

Systematic fault-tolerance	Application-specific fault-tolerance
• replication of components	• no replication necessary
• divergence among replicas in case of faults	• —
• no reasonableness checks necessary	• reasonableness checks for fault detection
• requires replica determinism	• —
• no application knowledge necessary	• depends on application knowledge
• exact distinction between correct and faulty behavior	• fault detection is limited by a *gray zone*

Comparison of fault-tolerance techniques (cont.)

Systematic fault-tolerance	Application-specific fault-tolerance
• no state estimations necessary • independence of application areas	• state estimations for continued service • missing or insufficient reasonableness checks for some application areas
• service quality is independent of whether replicated components are faulty or not • correct system function depends on the number of correct replicas and their failure semantics	• quality of state estimations is lower than quality delivered during normal operation • correct system function depends on the severity of faults and on the capability of reasonableness checks and state estimations
• only backward recovery	• forward and backward recovery

Comparison of fault-tolerance techniques (cont.)

Systematic fault-tolerance	Application-specific fault-tolerance
• additional costs for replicated components (if no system inherent replication is available)	• no additional costs for replicated components
• no increase in application complexity	• considerable increase in application complexity
• considerable increase of system level complexity	• no increase of system level complexity
• separation of fault-tolerance and application functionality	• application and fault-tolerance are closely intertwined
• fault-tolerance can be handled transparently to the application	• — // —

Systematic *and* application-specific fault-tolerance

- under practical conditions there will be a compromise between systematic and application-specific fault-tolerance
- usually cost, safety and reliability are the determining factors to choose a proper compromise
- software complexity plays an important role:
  - for complex systems software is almost unmanageable without adding fault-tolerance (fault containment regions and software robustness)
  - therefore systematic fault-tolerance should be applied in favor of application-specific fault-tolerance to reduce the software complexity
  - systematic fault-tolerance allows to test and to validate the mechanisms
    independently of the application software (divide and conquer)

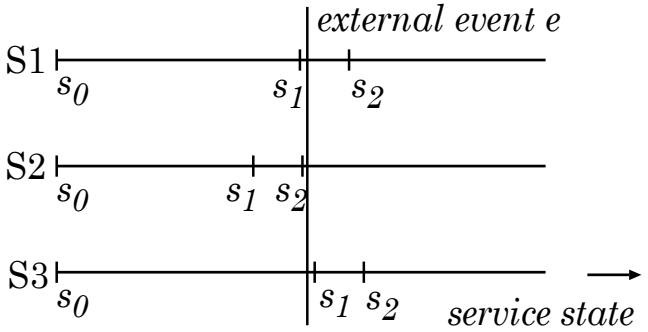
The problem of *Replica Determinism*

The problem of *Replica Determinism*

- For systematic fault-tolerance it is necessary that replicated components show consistent or deterministic behavior in the absence of faults
- If for example two active redundant components are working in parallel, both have to deliver corresponding results at corresponding points in time
- This requirement is fundamental to differentiate between correct and faulty behavior
- At a first glance it seems trivial to fulfill replica determinism since computer systems are assumed to be examples of deterministic behavior, but
- in the following it is shown that computer systems behave almost deterministically

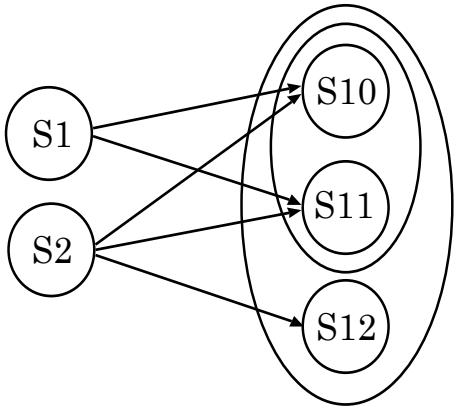
Nondeterministic behavior

- **Inconsistent inputs:**
  If inconsistent input values are presented to the replicas then the results may be inconsistent too.
  - a typical example is the reading of replicated analogue sensor read(S1) = 99.99 °C, read(S2) = 100.00 °C
- **Inconsistent order:**
  If service requests are presented to replicas in different order then the results will be inconsistent.



Nondeterministic behavior (cont.)

- Inconsistent membership information:
  Replicas may fail or leave groups voluntarily or new replicas may join a group.
  If replicas have inconsistent views about group membership it may happen that the results of individual replicas will differ.



Nondeterministic behavior (cont.)

- **Nondeterministic program constructs:**
  Besides intentional nondeterminism, like random number generators, some programming languages have nondeterministic program constructs for communication and synchronization (Ada, OCCAM, and FTCC).

```
task server is
   entry service_1();
   ...
   entry service_n();
end server;
```

```
task body server is
begin
   select
      accept service_1() do
         action_1();
      end;
   ...
   or
      accept service_n() do
         action_n();
      end;
   end select;
end server;
```

Nondeterministic behavior (cont.)

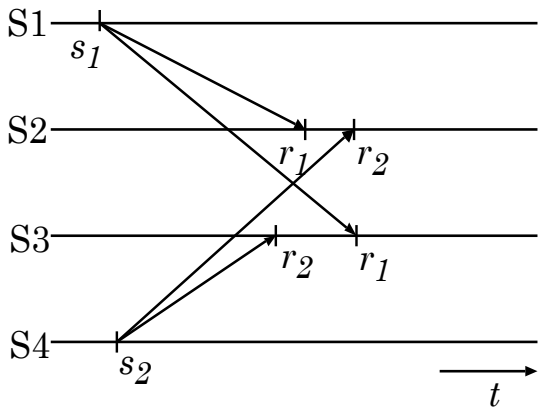
- **Local information:**
  If decisions with a replica are based on local knowledge (information which is not available to other replicas) then the replicas will return different results.
  - system or CPU load
  - local time
- **Timeouts:**
  Due to minimal processing speed differences or due to slight clock drifts it may happen that some replicas locally decide to timeout while others do not.

Nondeterministic behavior (cont.)

- **Dynamic scheduling decisions:**
  Dynamic scheduling decides in which order a series of service requests are executed on one or more processors. This may cause inconsistent order due to:
  - non-identical sets of service requests
  - minimal processing speed differences

Nondeterministic behavior (cont.)

- **Message transmission delays:**
  Variabilities in the message transmission delays can lead to different message arrival orders at different servers (for point-to-point communication topologies or topologies with routing).



The consistent comparison problem:

- computers can only represent finite sets of numbers
- it is therefore impossible to represent the real numbers exactly, they are rather approximated by equivalency classes
- if the results of arithmetic calculations are very close to the border of equivalency classes, different implementations can return diverging results
- different implementations are caused by: N-versio programming, different hardware, different floating point libraries, different compilers
- for example the calculation of (a – b)2 with floating point representation with a mantissa of 4 decimal digits and rounding where a = 100 and b = 0.005 gives different result for mathematical equivalent formulas.

  (a – b)2 = 1.000 104

  (a – b)2 = a2 –2ab + b2 = 9.999 103

Fundamental limitations to replication

The real world abstraction limitation:

- dependable computer systems usually interface with continuous real-world quantities:

quantity	SI-unit
distance	meter [m]
mass	kilogram [kg]
time	second [s]
electrical current	ampere [A]

- these continuous quantities have to be abstracted (or represented) by finite sets of discrete numbers
- due to the finite accuracy of any interface device, different discrete representations will be selected by different replicas

Fundamental limitations to replication (cont.)

The impossibility of exact agreement:

- due to the real world abstraction limitation it is impossible to avoid the introduction of replica non-determinism at the interface level
- but it is also impossible to avoid the once introduced replica nondeterminism by agreement protocols completely
- exact agreement would require ideal simultaneous actions, but in the best case actions can be only simultaneous within a time interval ***d***

Fundamental limitations to replication (cont.)

Intention and missing coordination:

- replica nondeterminism can be introduced intentionally
- or unintentionally by omitting some necessary coordinating actions

Replica control

- Due to these fundamental limitations to replication it is **necessary** to enforce replica determinism which is called replica control.

Internal vs. external replica control

Internal replica control:

- avoid nondeterministic program constructs, uncoordinated timeouts, dynamic scheduling decisions, diverse program implementations, local information, and uncoordinated time services
- can only be enforced partially due to the fundamental limitations to replication

External replica control:

- control nondeterminism of sensor inputs
- avoid nondeterminism introduced by the communication service
- control nondeterminism introduced by the program execution on the replicated processors by exchanging information

Def.: Replica Determinism

Correct replicas show correspondence of service outputs and/or service states under the assumption that all servers within a group start in the same initial state, executing corresponding service requests within a given time interval.

- this generic definition covers a broad range of systems
- correspondence and within a given time interval needs to be defined according to the application semantics

Services for replicated fault-tolerant systems

- Basic Services
- Clock Synchronization Services
- Communication Services
- Replica Control Services

Groups, resiliency and replication level

- Replicated entities such as processors are called groups.
- The number of replicas in a group is called replication level
- A group is said to be n-resilient if up to n processor failures can be tolerated

Services for replicated fault-tolerant systems

- Basic Services
- Clock Synchronization Services
- Communication Services
- Replica Control Services

Basic services for replicated fault-tolerant systems

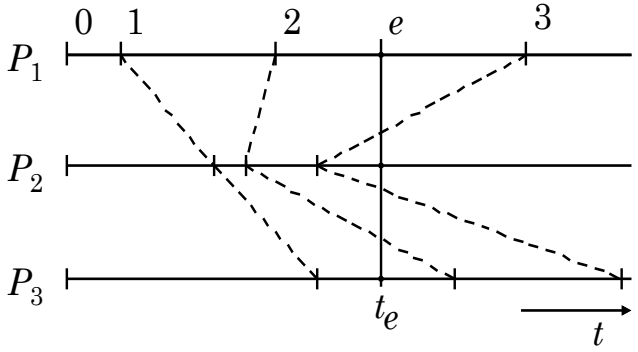
- **Membership:**
  Every non-faulty processor within a group has timely and consistent information on the set of functioning processors which constitute the group.
- **Agreement:**
  Every non-faulty processor in a group receives the same service requests within a given time interval.
- **Order:**
  Explicit service requests as well as implicit service requests, which are intro-duced by the passage of time, are processed by non-faulty processors of a group in the same order.

Services for replicated fault-tolerant systems

- Basic Services
- Clock Synchronization Services
- Communication Services
- Replica Control Services

Logical Clocks

- all members in a group observe the same events in the same order
- this applies to process internal events and external events such as service requests and faults
- for all events e_i and e_j, if $e_i \rightarrow e_j$ then $LC(e_i) < LC(e_j)$ where $\rightarrow$ is the happened before relation as defined by Lamport
- external events need to be reordered according to the internal precedence relation and individual processing speeds



Logical Clocks (cont.)

- The happened before relation (→) defines a partial order of events and captures potential causality, but excludes external clandestine channels

- Def.: The relation → on a set of events in a distributed system is the smallest relation satisfying the following three relations:
  1. If a and b are events performed by the same process, and a is performed before b then a → b.
  2. If a is the event of sending of a message by one process and b the receiving of the same message by another process, then a → b.
  3. Transitivity: if a → b and b → c, then a → c.

Logical Clocks (cont.)

Two distinctive events are said to be concurrent if neither a → b nor b → a.

An algorithm implementing logical clocks:

1. Each process P_i increments LC_i between two successive events
2. Upon receiving a message m, a process P_j sets $LC_j = \max(LC_j, T_m)$, where T_m is message m's timestamp.

Logical Clocks (cont.)

- the algorithm defines no total order since independent processes may use the same timestamp for different events
- a possible solution is to break ties by using a lexicographical process order
- logical clocks have no gap-detection property

- **Gap-detection**:
  Given two events e and e' with clock values LC(e) < LC(e'), determine whether some other event e'' exists such that LC(e) < LC(e'') < LC(e').

Logical Clocks (cont.)

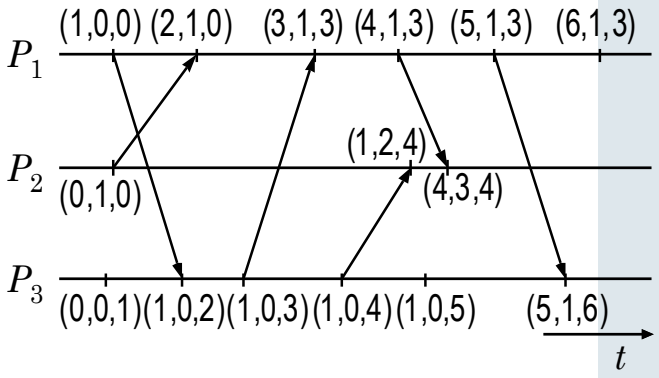
- the gap-detection property is necessary for stability and a bounded action delay, i.e., before an action is taken it has to be guaranteed that no earlier messages are delivered
- stability and action delay are based on potential causality, two events e and e' are potential causal related if e → e'.

Vector Clocks

- vector clocks are an extension of logical clocks which have gap-detection property
- An algorithm implementing vector clocks:

  (1) Each process P_i increments VC_i between two successive events

  (2) Upon receiving a message m, a process P_j sets all vector elements VC_j to the maximum of VC_j and T_m, where T_m is message m's vector clock timestamp. Afterwards the element $VC_j[j]$ is incremented.



- Potential causality for vector clocks $e \rightarrow e' \equiv VC(e) < VC(e')$
  - $VC < VC' \equiv (VC \circ VC') \wedge (\forall i{:}1 \leq i \leq n{:}\ VC[i] \leq VC'[i])$

Real-Time Clocks

- all processors have access to a central real-time clock or
- all processors have local real-time clocks which are approximately synchronized
- the synchronized clocks define a global time grid where individual clocks are off at most by one tick at any time instant t
- the maximum deviation among clocks is called precision
- t-precedent events (events that are at least t real-time steps apart) can be causally related regardless of clandestine channels

Comparing Real-Time and Logical Clocks

real-time clocks	logical clocks
synchronous system model	asynchronous system model
higher synchronization overhead	little delays and synchronization overhead if only system internal events are considered
needs to achieve consensus on the systematic clock error of one tick	external events need to be reordered in accordance to logical time
stability within one clock tick	unbounded duration for stability, requires consistent cut or vector clock
potential causality for *t*-precedent external events	potential causality only for closed systems
bounded action delay (total order)	unbounded action delay (no total order)

Services for replicated fault-tolerant systems

- Basic Services
- Clock Synchronization Services
- Communication Services
- Replica Control Services

Communication Services

The following arguments motivate the close interdependence of fault-tolerant computer systems, communication and replica control:

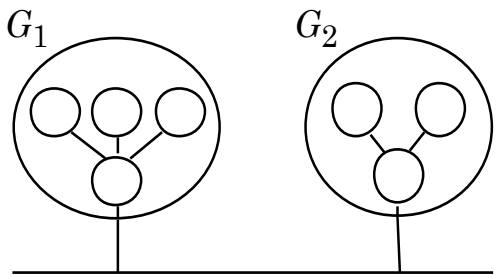
- fault-tolerant systems are built on the assumption that individual components fail independently
- this assumptions requires the physical and electrical isolation of components at the hardware level
- these properties are best fulfilled by a distributed computer system where nodes are communicating through message passing but have no shared resources except for the communication media
- furthermore it has to be guaranteed that faulty nodes are not able to disturb the communication of correct nodes and that faulty nodes are not allowed to contaminate the system

Services for replicated fault-tolerant systems

- Basic Services
- Clock Synchronization Services
- Communication Services
- Replica Control Services

Central Replica Control

- **Strictly central replica control:**
  - there is one distinguished processor within a group called leader or central processor
  - the leader takes all nondeterministic decisions
  - the remaining processors in the group, called followers, take over the leaders decisions

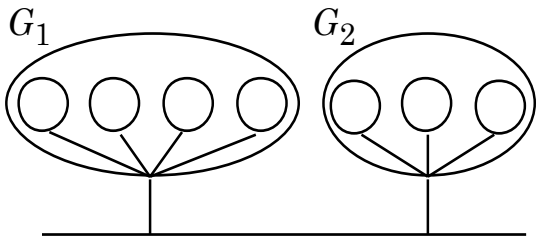


Central Replica Control (cont.)

- Strictly central replica control requires a communication service assuring reliable broad- or multicast.
- **Reliable broadcast**: A distinguished processor, called the transmitter, sends its local service request to all other processors in the group, fulfilling the following properties:
  - Consistency: All correct processors agree on the same value and all decisions are final.
  - Non-triviality: If the transmitter is non faulty, all correct processors agree on the input value sent by the transmitter.
  - Termination: Each correct processor decides on a value within a finite time interval .

Distributed Replica Control

- **Strictly distributed replica control:**
  - there is no leader role, each processor in the group performs exactly the same way
  - to guarantee replica determinism the group members have to carry out a consensus protocol on nondeterministic decisions



Distributed Replica Control (cont.)

- Any (partially) distributed replica control strategy requires a communication service assuring consensus.
- Consensus: Each processor starts a protocol with its local input value, which is sent to all other processors in the group, fulfilling the following properties:
  - Consistency: All correct processors agree on the same value and all decisions are final.
  - Non-triviality: The agreed-upon input value must have been some processors input (or is a function of the individual input values).
  - Termination: Each correct processor decides on a value within a finite time interval.

Replica Control Strategies

Lock-step execution:

- processors are executing in synchronous
- the outputs of processors are compared after each single operation
- typically implemented at the hardware level with identical processors

- **Advantages:**
  - arbitrary software can be used without modifications for fault-tolerance (important for commercial systems)
- **Disadvantages:**
  - common clock is single point of failure
  - transient faults can affect all processors at the same point in the computation
  - high clock speed limits number and distance of processors
  - restricted failure semantics

Replica Control Strategies (cont.)

Active replication:

- all processors in the group are carrying out the same service requests in parallel
- strictly distributed approach, nondeterministic decisions need to be resolved by means of an agreement protocol
- the communication media is the only shared resource

- **Advantages:**
  - unrestricted failure semantics
  - no single point of failure
- **Disadvantages:**
  - requires the highest degree of replica control
  - high communication effort for consensus protocols
  - problems with dynamic scheduling decisions and timeouts

Replica Control Strategies (cont.)

Semi-active replication:

- intermediate approach between distributed and centralized
- the leader takes all nondeterministic decisions
- the followers are executing in parallel until a potential nondeterministic decision point is reached

- **Advantages:**
  - no need to carry out a consensus protocol
  - lower complexity of the communication protocol (compared to active replication)
- **Disadvantages:**
  - restricted failure semantics, the leaders decisions are single points of failures
  - problems with dynamic scheduling decisions and timeouts

Replica Control Strategies (cont.)

Passive replication:

- only one processor in the group – called primary – is active
- the other processors in the group are in standby
- checkpointing to store last correct service state and pending service requests

- **Advantages:**
  - requires the least processing resources
  - standby processors can perform additional tasks
  - highest reliability of all strategies (if assumption coverage = 1)
- **Disadvantages:**
  - restricted failure semantics (crash or fail-stop)
  - long resynchronization delay

Failures and Replication

- Centralized replication:
  - semi-active and passive replication
  - the leading processor is required to be fail restrained
  - byzantine or performance failures of the leader cannot be detected by other processors in the group (“heartbeat” or “I am alive” messages)
  - to tolerate t failures with crash or omission semantics t + 1 processors are necessary
  - the result of any processor (e.g. the fastest) can be used
  - if no reliable broadcast service is available 2t + 1 processors are necessary

Failures and Replication (cont.)

Distributed replication:

- active replication
- no restricted failure semantics of processors
- to tolerate t crash or omission failures t + 1 processors are necessary
- to tolerate t performance failures 2t + 1 processors are necessary
- to tolerate t byzantine failures 3t + 1 processors are necessary
- for crash or omission failures it is sufficient to take 1 processor result
- for performance or byzantine failure t + 1 identical results are required