Distributed Systems
Fault Tolerance: Principles and Paradigms

Vinny Cahill & Brendan Tangney

Distributed Systems Group
Department of Computer Science
The University of Dublin
Trinity College

vinny.cahill@cs.tcd.ie
brendan.tangney@cs.tcd.ie
www.dsg.cs.tcd.ie

Motivation 1/2

"A distributed system is one that stops you from getting any work done when a machine you've never even heard of crashes."

Attributed to Lamport [Mullender 1989]

Motivation 2/2

See [Laprie 1992]

- *Dependability* is defined as the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers

- Subsumes reliability, safety, maintainability, availability, security

Contents

Why do computer systems fail? 1/2

See [Gray and Reuter 1993]

- Hardware
  - processor, memory, connectors, network, disks
- Software
  - program and specification bugs
- Maintenance
  - including preventive maintenance, upgrades, and repairs
- Environment
  - power, air conditioning, fire, flood, sabotage, war
- Operations
  - system management and configuration
- Process
  - strikes, administrative decisions

Why do computer systems fail? 2/2

- Failures are rare
- Hardware is reliable
  - possibly less than 10% of failures due to hardware
- Software is a problem
  - possibly 50% of failures due to software
- Maintenance, environment, and operations are significant
  - 30%

Faults, errors, and failures 1/3

See [Laprie 1992]

- The *service* delivered by a computer system is its behaviour as it is perceived by its users
  - user may be a human or another system
  - external view

- A *specification* is an agreed description of the service that a system is intended to deliver

Faults, errors, and failures 2/3

- A *failure* occurs when the delivered service no longer complies with its specification

- An *error* is an unintended state of the system that is liable to lead to a subsequent failure

- The adjudged or hypothesised cause of an error is a *fault*

Faults, errors, and failures 3/3

- A *fault* such as electromagnetic interference may cause corruption of data stored on magnetic disk
- The presence of corrupted data represents an *error*
- The disk system *fails* when the user is subsequently unable to retrieve stored data as advertised

Means of dependability

See [Laprie 1992]

- *Fault prevention* is concerned with how to prevent fault occurrence or introduction
- *Fault tolerance* is concerned with how to provide services complying with their specifications in spite of faults
- *Fault removal* is concerned with how to reduce the number or seriousness of faults present
  - verification (including testing), diagnosis, correction
- *Fault forecasting* is concerned with how to estimate the present number, the future incidence, and the consequences of faults
  - evaluation of system behaviour wrt to fault occurrence or activation

Tolerating faults 1/5

- Processor faults
  - → machine or operating system crash
- Communications faults
  - → lost, corrupted, duplicated messages or network partition
- Media faults
  - → disk head crash, decay
- Process faults
  - → resource shortage, program bug
- User aborts
  - → ^C

Tolerating faults 2/5

- Many systems need to be able to *continue* or *recover* even in the event of such faults occurring
  - → reliability and availability
- More important for many systems is that critical *data* must not be lost or corrupted in the event of a fault
  - → consistency

Tolerating faults 3/5

Techniques for handling various types of faults are well-known:

- Processor faults
  - reboot using checkpoint/log or use replicated processes
- Communications faults
  - time-out and retransmit
- Media faults
  - keep multiple copies of the data (backups)
- Process faults
  - anticipate likely problems and write exception handlers or write multiple copies of application
- User aborts
  - anticipate likely problems and write exception handlers

Tolerating faults 4/5

- In general, all of these techniques use *redundancy/replication*
- May use redundancy in *space* or *time*
- Redundant processing, data, transmission

Tolerating faults 5/5

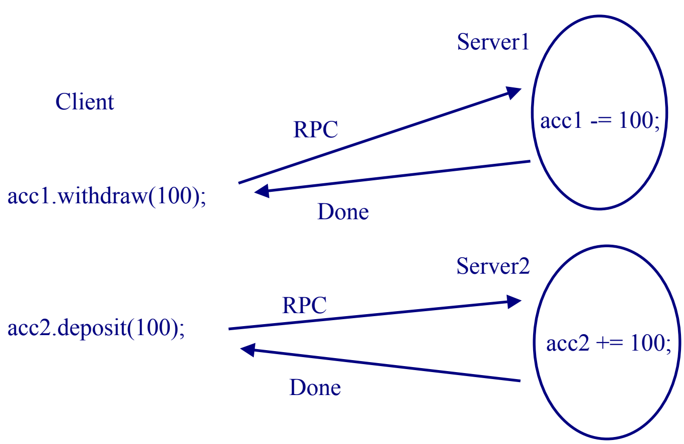
- Distributed systems obviously provide a good basis for implementing fault-tolerant systems because of their inherent redundancy

But, must cater for:

- Complex dependencies between components
  - number of components, parallelism, timing
- Partial failure
  - must maintain consistency
- Lack of global knowledge
  - message passing is slow and is itself unreliable

A classic example 1/3

Funds transfer between two bank accounts managed by different servers



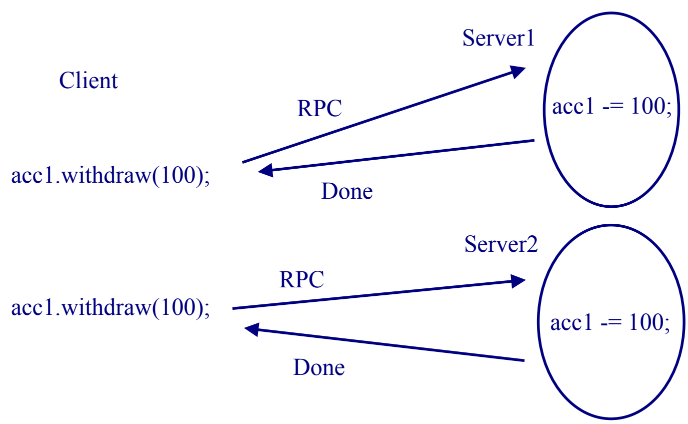
A classic example 2/3

What happens if:

- No reply to first request is received?
  - request lost? reply lost? server crashed?
  - has the withdrawal been made or not?
  - retry? but don’t want to withdraw money twice!
- Server1 crashes after replying to client?
  - will the client know?
  - will the update to acc1 persist?
- No reply to second request received?
  - as above but what to do about Server1?
  - inconsistency may arise due to partial failure!

A classic example 3/3

Suppose our servers manage replicas of a bank account from which the client wants to withdraw money?



Classification of failure 1/9

See [Cristian 1991]

- A given *service* provides a set of operations that may be executed in response to requests from clients
- Execution of such an operation may result in *state transitions* in the service and/or in *output* to the user/environment
- The operation of a server is described by its *specification*
- A description of the ways in which a server may fail is called its *failure semantics*

Classification of failure 2/9

- In general any service *depends* on a number of other services
  - a file service might depend on a disk block service and a processor service
- A *fault-tolerant service* is one that behaves according to its specification even in the presence of failures in the other services on which it depends
  - a fault-tolerant service may exhibit any desired failure semantics
  - in this view a fault is simply a failure of a service on which some service depends
    - recursive
- A service *masks* a failure in a service on which it depends either by hiding it altogether or by converting it into one of the failures it is allowed to exhibit

Classification of failure 3/9

Class of failure	Subclass	Description
Omission failure		a server omits to respond to a request
Response failure		a server responds incorrectly to a request
	value failure	server returns wrong value
	state transition failure	*server makes incorrect state transition*

Note that under our previous definitions state transition failures are not meaningful
- not visible to users and hence not part of the specification

Classification of failure 4/9

- Omission failure: UDP may lose messages
- Response failure: IP may corrupt messages (as well as lose them)
- UDP has omission failure semantics while IP has response/omission failure semantics
- UDP masks response failures in the IP service on which it depends by using a checksum to detect corrupted messages and discarding them
  - → convert IP response failure to UDP omission failure
- UDP tolerates IP response failures

Classification of failure 5/9

Class of failure	Subclass	Description
Timing failure		server does not respond in the specified real-time interval
	late/ performance	server responds too late
	early	server responds too early

Classification of failure 6/9

Class of failure	Subclass	Description
Crash failure		server repeatedly fails to respond to requests until restarted
	amnesia crash	*server restarts in initial state*
	partial amnesia crash	*some part of state is as before the crash while remainder is reset to initial state*
	pause crash	*server restarts in state before crash*
	halting crash	server never restarts

A crash failure can be characterised as a persistent omission failure

A system whose failures are to an acceptable extent only crash failures is called a *fail-silent* system

Classification of failure 7/9

- Amnesia crash: machine/operating system crash and reboot
  - of diskless system
- Partial amnesia crash: file server crash before all updates flushed to disk
  - contents of primary memory lost but secondary storage survives
- Partial amnesia crash: database crash and recovery with effects of committed transactions reflected in database state
  - Cristian refers to this as transaction-failure semantics
- Pause crash: operating system hangs due to over-load
- Halting crash: machine room burns down

Classification of failure 8/9

- A service can exhibit behaviours in the union of two failure classes
- Such a server has *weaker* failure semantics than one which exhibits behaviours in only one class
- The *stronger* the failure semantics specified the more expensive it is to build the service
- The weakest failure semantic is the union of all the failures classes introduced earlier – known as *arbitrary* failure semantics
  - Murphy's law: consider that anything that can go wrong will and at the worst possible time!

Classification of failure 9/9

- UDP has only omission failures
- IP exhibits both omission and response failure semantics
  - omission/response
- IP has weaker failure semantics than UDP
- UDP uses extra space for checksum and additional processing to calculate it

- Off the shelf CPUs have arbitrary failure semantics
- Can build a CPU with crash failure semantics by using two off the shelf CPUs which execute in parallel and comparing their results
  - crash silently if the results ever differ

Probability of failure

- When we say that a service S has X failure semantics, we mean that there is a *high probability* that S will *only* exhibit failures of class X
  - or low probability of any other failure being observed

- If S exhibits any other class of failure, this is a *disaster*
  - users of S are normally unprepared for disasters and will fail

Specify:

- Minimum probability s_r that standard behaviour is observed
- Maximum probability c_r that a failure different from specified failure behaviour will occur
- Whether c_r can be attained depends on probability of failure of services on which S depends

Perception of failure 1/2

Where a system has multiple users

- *Consistent* failures occur when all users see the same (incorrect) response
  - → including omission
- *Inconsistent* failures occur when different users may see different responses
- Inconsistent arbitrary failures are often called *Byzantine* failures
  - → aka two-faced, malicious

Perception of failure 2/2

- Question: what effect does failure perception have on redundancy required to tolerate failure?

Severity of failure

- *Benign* failures have costs that are of the same order of magnitude as the benefit provided by correct service delivery
- *Catastrophic* failures have costs that are orders of magnitude greater than the benefit provided by correct service delivery
  - → genuine catastrophe, human death
- Systems whose failures might be catastrophic are called *safety-critical systems*

Attributes of dependability 1/4

- *Reliability* is a measure of continuity of service delivery
  - → time to failure
- The reliability R(t) of a system is the probability that the system will provide the specified service until time t
- Reliability depends on the expected *failure rate* (λ) in failures/hour
- The *mean time to failure* (MTTF) in hours is the inverse of the failure rate, i.e., $1/\lambda$
- If λ is required to be of the order of 10^{-9} failures/hour or less, the system is said to have an *ultrahigh reliability* requirement

Attributes of dependability 2/4

- *Safety* is reliability with respect to catastrophic failures
  - → time to catastrophic failure
- Safety-critical systems must have reliability wrt to catastrophic failures that satisfies the ultrahigh reliability requirement

Attributes of dependability 3/4

- *Maintainability* is a measure of the time to restoration from the last failure
  - → probably only interesting for benign failures
- The maintainability M(t) of a system is the probability that the system is restored within a time interval t after the failure
- *Repair rate* (μ) repairs/hour; *mean time to repair* (MTTR) in hours is 1/μ
- Possible tradeoff between reliability and maintainability
  - → c.f. smallest replaceable unit

Attributes of dependability 4/4

- *Availability* is a measure of the delivery of correct service wrt to the alternation of correct and incorrect service
- Availability is measured by the fraction of time that the system is ready to provide service
- With constant λ and μ

$$A = MTTF/(MTTF+MTTR)$$

Classification of dependable systems

From [Gray and Reuter 1993]

Class	System type	Availability	Unavailability
1	Unmanaged	90%	52,560 min/year
2	Managed	99%	5,256
3	Well-managed	99.9%	526
4	Fault-tolerant	99.99%	53
5	High-availability	99.999%	5
6	Very-high-availability	99.9999%	.5
7	Ultra-high-availability	99.99999%	.05

Examples

- Nuclear reactor monitoring - class 5
- Telephone switches - class 6
- In-flight computers - class 9

References 1/2

- [Cristian 1991] Flaviu Cristian, *Understanding Fault-Tolerant Distributed Systems*, CACM, 34(2):56-78, February 1991.
- [Gray and Reuter 1993] Jim Gray and Andreas Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1993, ISBN 1-55860-190-2, Chapter 3.
- [Laprie 1992] Jean-Claude Laprie (Ed.), *Dependability: Basic Concepts and Terminology*, Springer-Verlag, 1992, ISBN 3-211-82296-8.
- [Mullender 1989] Sape Mullender (Ed.), *Distributed Systems*, ACM Press Frontier Series, 1989, ISBN 0-2-1-41660-3, Chapter 1.

References 2/2

See also:

- [Birman 1996] Ken Birman, *Building Secure and Reliable Network Applications*, Manning/Prentice Hall, 1996, ISBN 1-884777-29-5, Chapter 12.
- [Kopetz 1997] Hermann Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997, Chapters 1 and 6.