Robustness through replication

(Slides inspired by Ken Friis Larsen and Marcos Vaz Salles)

Joachim Tilsted Kristensen

University of Oslo

IN5570, April 13, 2023

Agenda

- Terminology.
- Metrics of reliability.
- Means of robustness.
- Examples (from the wild).
- Replication in general.

Terminology

Adverse conditions

- **Fault** : A defect that causes problems (software bugs, network problems, hardware defects, etc).

Terminology

Adverse conditions

- **Fault** : A defect that causes problems (software bugs, network problems, hardware defects, etc).
- **Error** : The manifestation of a fault, which causes the system to perform unintended actions or produce incorrect results.

Terminology

Adverse conditions

- **Fault** : A defect that causes problems (software bugs, network problems, hardware defects, etc).
- **Error** : The manifestation of a fault, which causes the system to perform unintended actions or produce incorrect results.
- **Failure** : The system is unable to perform as intended. For instance, an error that causes the interface to disconnect.

Terminology

Adverse conditions

- **Fault** : A defect that causes problems (software bugs, network problems, hardware defects, etc).
- **Error** : The manifestation of a fault, which causes the system to perform unintended actions or produce incorrect results.
- **Failure** : The system is unable to perform as intended. For instance, an error that causes the interface to disconnect.

Robustness

The ability to handle failures gracefully: A robust system can recover from faults and errors. (Often achieved through redundancy, fault tolerance, and error handling)

Terminology

Adverse conditions

- **Fault** : A defect that causes problems (software bugs, network problems, hardware defects, etc).
- **Error** : The manifestation of a fault, which causes the system to perform unintended actions or produce incorrect results.
- **Failure** : The system is unable to perform as intended. For instance, an error that causes the interface to disconnect.

Robustness

The ability to handle failures gracefully: A robust system can recover from faults and errors. (Often achieved through redundancy, fault tolerance, and error handling)

Reliability

The ability to consistently provide the expected results or output, over a period of time and under normal operating conditions: A reliable system can be trusted to function correctly, without producing errors.

Reliability

Metrics

- Failure rate ($\frac{\text{number of failures}}{\text{operation time}}$)
- Mean time to failure (MTTF = reciprocal failure rate)
- Mean time to repair (MTTR)
- Mean time between failures (MTBF = MTTF + MTTR)
- Availability (MTTF / MTBF)
- Downtime (1 - Avaliability)

Reliability

Metrics

- Failure rate ($\frac{\text{number of failures}}{\text{operation time}}$)
- Mean time to failure (MTTF = reciprocal failure rate)
- Mean time to repair (MTTR)
- Mean time between failures (MTBF = MTTF + MTTR)
- Availability (MTTF / MTBF)
- Downtime (1 - Avaliability)

Exercise

Consider a distributed system with 1000 machines, and the failure rate of any particular machine is once per 15 *years*. What is your approximation of mean time between failures?

Reliability

Metrics

- Failure rate ($\frac{\text{number of failures}}{\text{operation time}}$)
- Mean time to failure (MTTF = reciprocal failure rate)
- Mean time to repair (MTTR)
- Mean time between failures (MTBF = MTTF + MTTR)
- Availability (MTTF / MTBF)
- Downtime (1 - Avaliability)

Exercise

Consider a distributed system with 1000 machines, and the failure rate of any particular machine is once per 15 *years*. What is your approximation of mean time between failures?

- What happens if we increase the number of machines?

Robustness

Failure is inevitable!

With 10000 machines and MTBF of 30 *years*, we expect one failure per day. So, how can we deal with it?

Robustness

Failure is inevitable!

With 10000 machines and MTBF of 30 *years*, we expect one failure per day. So, how can we deal with it?

Error detection

- Program instrumentation.
- Use redundancy to verify correctness.
- Example : Continuous integration.

Robustness

Failure is inevitable!

With 10000 machines and MTBF of 30 *years*, we expect one failure per day. So, how can we deal with it?

Error detection

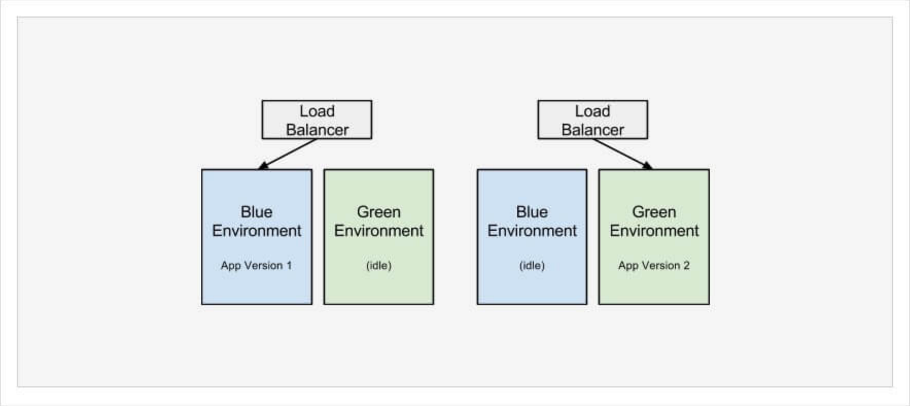
- Program instrumentation.
- Use redundancy to verify correctness.
- Example : Continuous integration.

Error containment strategies (fail fast)

- Halt on failure : Immediately stop error propagation.
- Fail safe : Limited operation during failure recovery.
- Soft failure : Continue with a subset of the functionality.

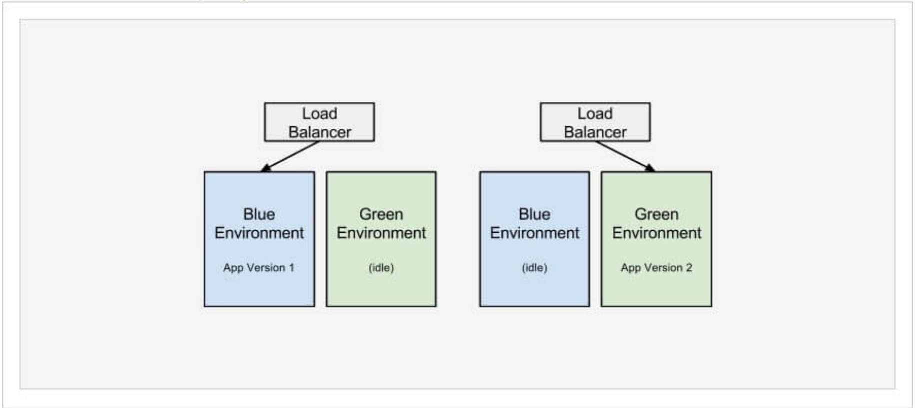
Fail safe

Blue/Green deployment scheme



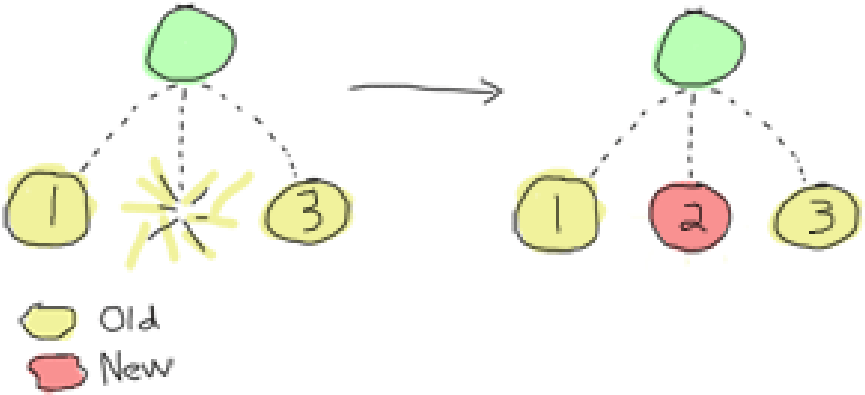
Fail safe

Blue/Green deployment scheme



Services (k8s)

Halt on failure (Supervisor pattern)



Mnesia (distributed telecommunications DMBS)

Capabilities

- Persistence. Tables can be coherently kept on disc and in the main memory.
- Replication. Tables can be replicated at several nodes.
- Atomic transactions. A series of table manipulation operations can be grouped into a single atomic transaction.
- Location transparency. Programs can be written without knowledge of the actual data location.
- Extremely fast real-time data searches.
- Schema manipulation routines. The DBMS can be reconfigured at runtime without stopping the system.

Replication (in general)

Desirable properties of distributed systems

- **Consistency** : Every read receives the most recent write or an error.
- **Availability** : Every request receives a response (without guarantee that it contains the most recent version of the information).
- **Partition Tolerance** : The system continues to operate despite arbitrary message loss or failure of part of the system.

Replication (in general)

Desirable properties of distributed systems

- **Consistency** : Every read receives the most recent write or an error.
- **Availability** : Every request receives a response (without guarantee that it contains the most recent version of the information).
- **Partition Tolerance** : The system continues to operate despite arbitrary message loss or failure of part of the system.

CAP Theorem

In any distributed system, you can only have two of these properties at the expense of the third. For example, if you prioritise consistency and partition tolerance, then availability may suffer in the event of network partitions or failures.

Final slide

What have we learned

- Large computer systems fail fairly often.
- Precisely how often, is called **Reliability**.
- **Robustness** : is about designing computer systems to account for failure.
- **Replication** : is a fundamental technique for ensuring partition tolerance and availability, by means of mitigating single point of failure.

What is next?

- **Consensus algorithms** : are used to ensure that nodes in a distributed system agree on a particular value or decision.
- **Vector clocks** : can be used to track causal relationships between events in a distributed system. By using vector clocks, a system can determine whether two events occurred concurrently or in a particular order.