Consensus in Synchronous Systems

Implementing Fault-tolerant Services using the State Machine approach: A tutorial

Schneider, F.B. (1990).
ACM Computing Surveys,
Vol. 22, No. 4, pp. 299–319.

Problem

A component is considered faulty once its behavior is no longer consistent with its specification.

Faulty behavior

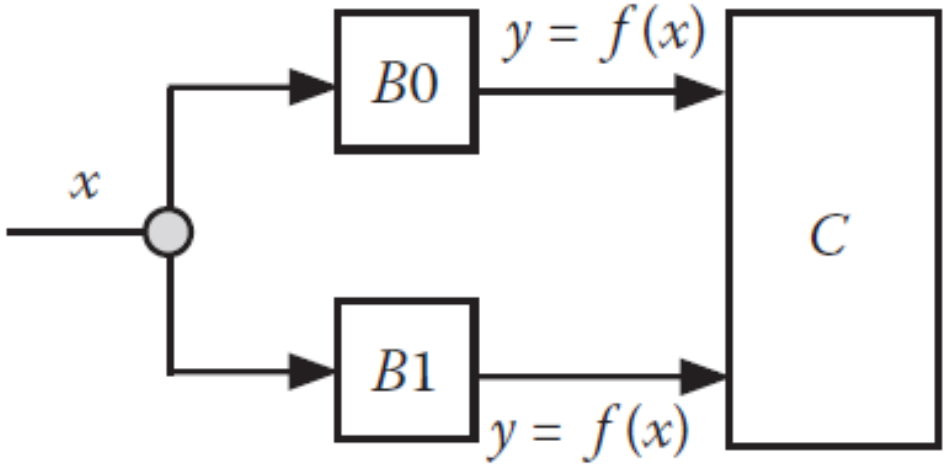
- Fail-stop
- Arbitrary

Fault tolerant systems

A system consisting of a set of n distinct components is ***t*** fault tolerant if it satisfies its specification provided that no more than ***t*** of those components become faulty during some interval of interest.

Fault tolerant systems

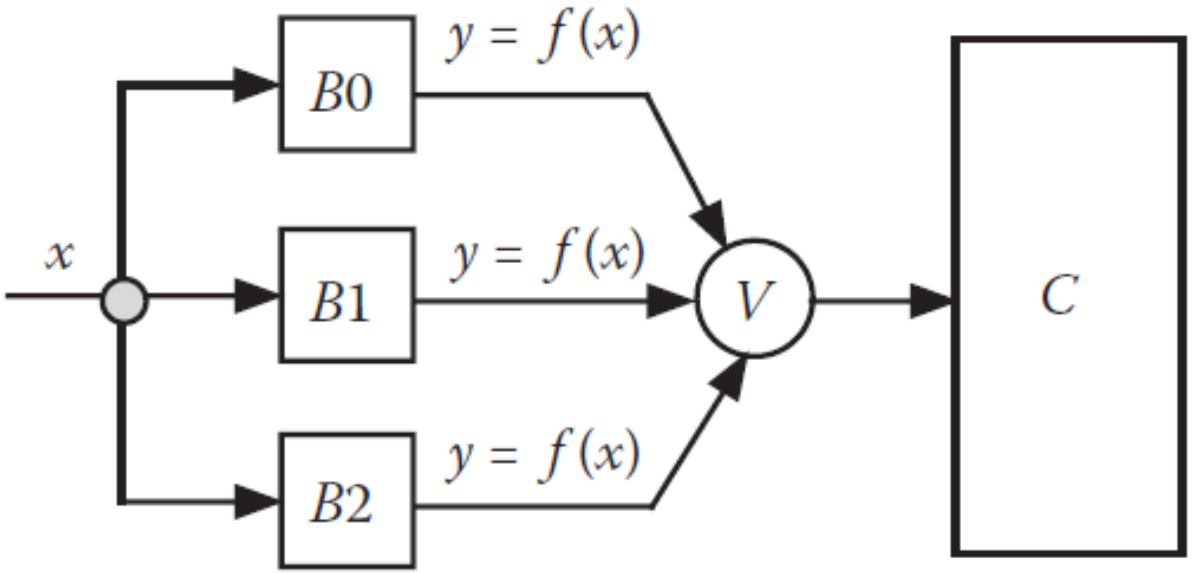
Double modular redundancy tolerates a single failure, when fail-stop components are being assumed.



$n \geq t+1$

Fault tolerant systems

Triple modular redundancy tolerates a single failure of arbitrary type.



$n \geq 2t+1$

Fault tolerant systems

What is the problem?

A failed component *x* may send conflicting information to different parts of the system.

x is a “Byzantine” general.

SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control

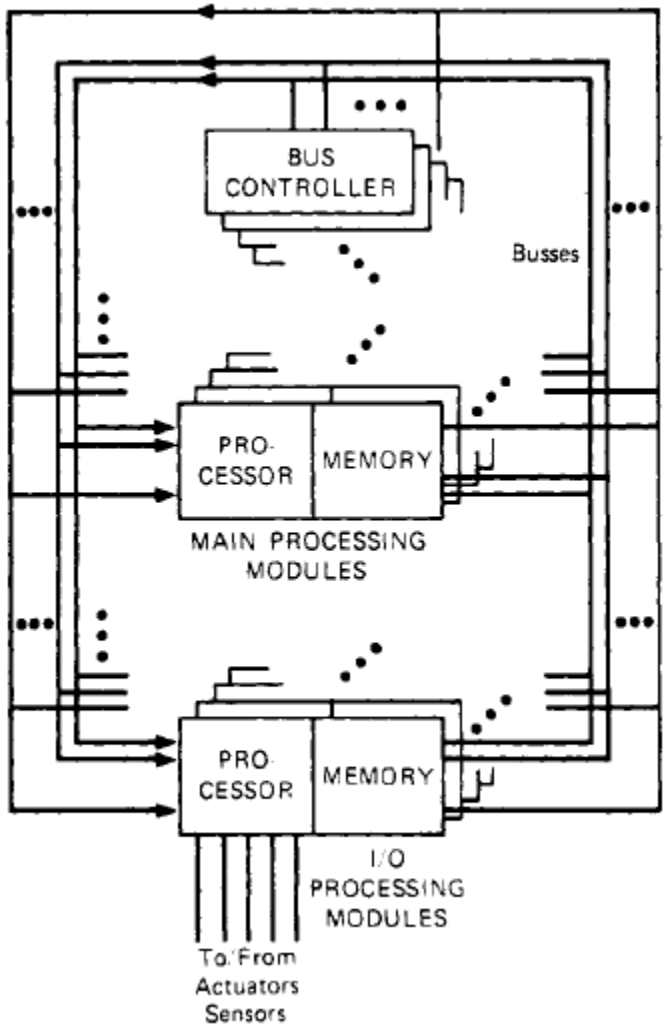
JOHN H. WENSLEY, LESLIE LAMPORT, JACK GOLDBERG, MILTON W. GREEN, KARL N. LEVITT, P. M. MELLIAR-SMITH, ROBERT E. SHOSTAK, AND CHARLES B. WEINSTOCK. (1978).

Proceedings of the IEEE, Vol. 66, No. 10, pp. 1240-1255.

Structure of the SIFT system

SIFT is an ultrareliable computer for critical control applications that achieves fault tolerance by the replication of tasks among processing units.

Software **I**mplemented **F**ault **T**olerance



System overview

SIFT executes a set of tasks, each of which consists of a sequence of iterations. The input data to each iteration of a task is the output data produced by the previous iteration of some collection of tasks.

Reliability is achieved by having each iteration of a task independently executed by a number of modules.

The value of the output data is chosen by a “two out of three” vote.

Tasks synchronization

The system must ensure that the different processors allocated to a task are executing the same iteration.

SIFT allows a degree of asynchronism between processors and avoids the lockstep traditional in ultrareliable systems. Up to 50 μs of skew between processors can readily be accommodated.

The processors must periodically resynchronize their clocks to ensure that no clock drifts too far from any other.

Clock synchronization

For reliability, the resynchronization procedure must be immune to the failure of any one clock or processor.

In order to guarantee high reliability, we cannot allow a system failure to be caused by any condition whose probability cannot be quantified.

This means that our synchronization procedure must be reliable in the face of the worst possible behavior of the failing component, even though that behavior may seem unrealistically malicious.

The median algorithm

Each clock observes every other clock and sets itself to the median of the values that it sees.

The justification for this algorithm is that, in the presence of only a single fault, the median value must either be the value of one of the valid clocks or else it must lie between a pair of valid clock values.

C_c is faulty and $C_B<C_A$ $C_B<C_A<C_C$ $C_B<C_C<C_A$ $C_C<C_B<C_A$

In either case, the median is an acceptable value for resynchronization.

The median algorithm

The weakness of this argument is that the worst possible failure mode of a clock may cause other clocks to observe different values for the failing clock.

Suppose that C_A sees a value $C_C > C_A$, while C_B sees a value $C_C < C_B$.

C_A sees $C_B < C_A < C_c$

C_B sees $C_C < C_B < C_A$

C_A and C_B will both see their own value as the median value, and therefore not change it. Both C_A and C_B are therefore resynchronizing onto themselves, and they will slowly drift apart until the system fails.

Impossibility result

Marshall Pease proved that, if the failure-mode behavior is permitted to be arbitrary, then there cannot exist any reliable clock resynchronization algorithm for three clocks.

Algorithm for 4 clocks tolerating 1 failure

The algorithm is carried out in two parts.

1. Each clock computes a vector of clock values, called the interactive consistency (IC) vector, having an entry for every clock.
2. Each clock uses the interactive consistency vector to compute its new value.

Computing the interactive consistency vector

The entry of the vector corresponding to clock p is set equal to p's own clock value. The value for the entry corresponding to another processor q is obtained by p as follows:

1. read q's value from q,
2. obtain from each other clock r the value of q that r read from q,
3. if a majority of these values agree, then the majority value is used, otherwise, the default value NIL (indicating that q is faulty) is used.

If at most one of the clocks is faulty, then:

1. each nonfaulty clock computes exactly the same interactive consistency vector (**agreement**),
2. the component of this vector corresponding to any nonfaulty clock q is q's actual value (**validity**).

Computing the new value

Let δ be the maximum amount by which the values of nonfaulty clocks may disagree. Any component that is not within δ of at least two other components is ignored, and any NIL component is ignored. The clock then takes the median value of the remaining components as its new value.

Since each nonfaulty clock computes exactly the same interactive consistency vector, each will compute exactly the same median value. Moreover, this value must be within δ of the original value of each nonfaulty clock.