An unusual Approach to Dependability for Space SW-Applications

Sergio Montenegro [(1)], **Raffaele Vitulli** [(2)],
[(1)] *FIRST, Kekuléstraße 7, 12489 Berlin, Germany, Email: sergio.montenegro@first.fraunhofer.de*
[(1)] *ESA/ESTEC, Keplerlaan 1, 2201AZ Noordwijk, The Netherlands, Email: Raffaele.Vitulli@esa.int*

0. Abstract

Much more than fault tolerance, space applications require dependability, which is the combination of availability, reliability and safety. Dependability is a major challenge when designing space computing systems, both at software and hardware level. After 30 years of contentious research on how to achieve high dependability, not an universal solution has been found. A huge effort has been invested to improve reliability, using reliable radiation hardened components. However, failures cannot be eliminated totally. The best solution one can achieve is just to postpone failures. Even when significantly postponed, failures still may occur.

Big efforts are made to improve reliability, for example by using reliable radiation hardened components. Many other efforts go into improved robustness by means of fault tolerance. In our approach we join efforts from reliability and robustness to reach very high dependability using limited resources. This is important because space missions have to be sparing with resources.

Another important aspect when designing board computers is that nowadays, it is not imaginable the use of the same (general purpose) computer in a cube sat (10x10x10 cm^3, 1 Kilogram) and in a mini satellite. For each satellite, a special purpose computer configuration need to be assembled, consisting of board computer, payload for cameras, star tracker, power control, reaction wheels, etc.

1. Introduction

1.1. common conceptual obstacles

From our experience we can identify at least four important and very common conceptual obstacles when designing a fault tolerant system.

1) Any digital control system is teamwork of software and hardware. Trying to solve any problem or challenge using only one of them means to lose a lot of possibilities, like a team (Software-Hardware) where only one player works. But still very few try to face the challenge of fault tolerance through teamwork of software and hardware.

2) Another wrong assumption is to think that you can provide fault tolerance by just replicating resources. For example let us try to control a water tank by opening and closing a valve, like in figure 1.

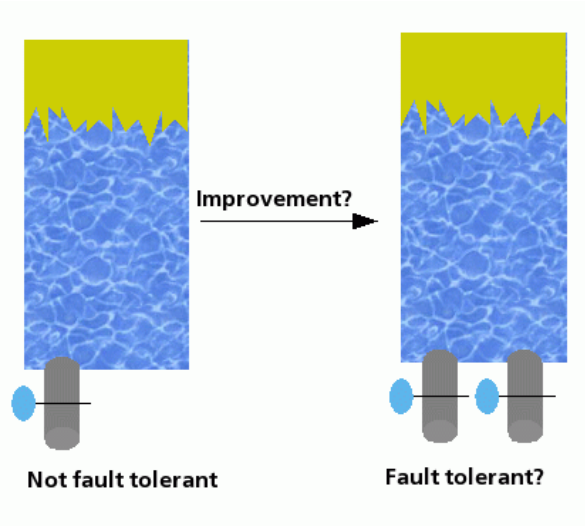

Figure 1 : Implementation of fault tolerance by replication?

The valve can fail. The first solution is to add another valve (figure 1). Now one of the redundant valves fails in its open position so that you can not control the tank any longer. What you got is a higher possibility of failure and not more dependability. This was not the right solution!

3) Another wrong assumption is to think one can reach an absolute failure free system. Then an enormous effort is invested to reach what one can not reach, the system will (in any way) crash some day and the worst is not to be ready to handle the crash. This will be the case if you think you have done a totally dependable system.

2. A non conventional approach to dependability

Let us consider (in this paper) four stages of fault tolerance:
0: no fault tolerance. One failure is enough to cause a system failure or system crash
1: Any single failure can collapse some critical functions, but the system remains in a safe operation. (Gracefully degradation)
2: Like 1, but after a short time period the system can reconfigure itself to go back to normal operation (resilience)
3: No single failure can disturb the normal operation of the system.

Stage 3 sounds very nice, but it requires at least a 3 fold replication of (almost) all resources, including volume, mass, power consumption, heat production, etc. For some space applications this can be a prohibitive condition. Furthermore after the first permanent failure, the system will go down to stage 2 or 1. For long operations where permanent failures are expected, stage 3 will be possible only at the beginning of the mission.

It is therefore advisable to create a resources economical system which provides stage 2 for the whole mission even after several failures. Our target is to get the highest possible dependability using very limited resources. The system has to be always operable in a safe way, but temporal loss of advanced features is allowed. Thereby it was very important not to

do again the common mistakes listed in chapter 1.

2.1. Step 1: Let it crash!

A realistic node computer operation can be represented like in figure 2.

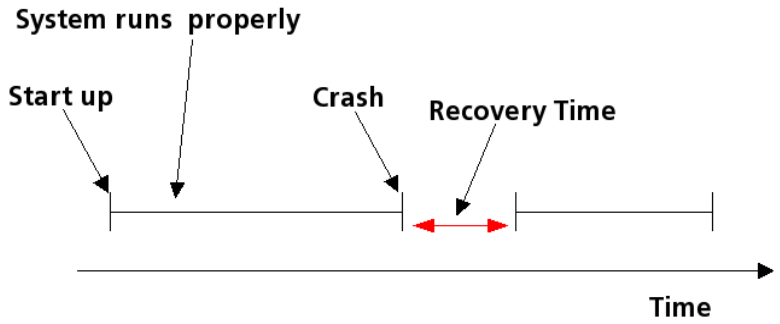

Figure 2 realistic node computer operation.

After start-up a node computer will work properly for a time period (may be years) but some day it will crash. Instead trying to create a (sub)system which shall never fail or where a failure is an extraordinary exception, we figure on failures. They are not exceptions, but expected events which will be handled smoothly. The paradox situation is: to be safe, we need to be ready to crash at any time. To increase the dependability of the system we try to increase the reliability of each node (e.g. mean time to failure - MTF) and reduce the recovery time after crashes (increasing availability). See figure 3.

Much more important than trying to avoid crashes (which we can not) is to provide an ultra fast recovery. Having a very short recovery time, we can tolerate very easily multiple node crashes. By using some kind of redundancy, the probability of two redundant nodes to be down at the same time becomes lower. See Figure 3.

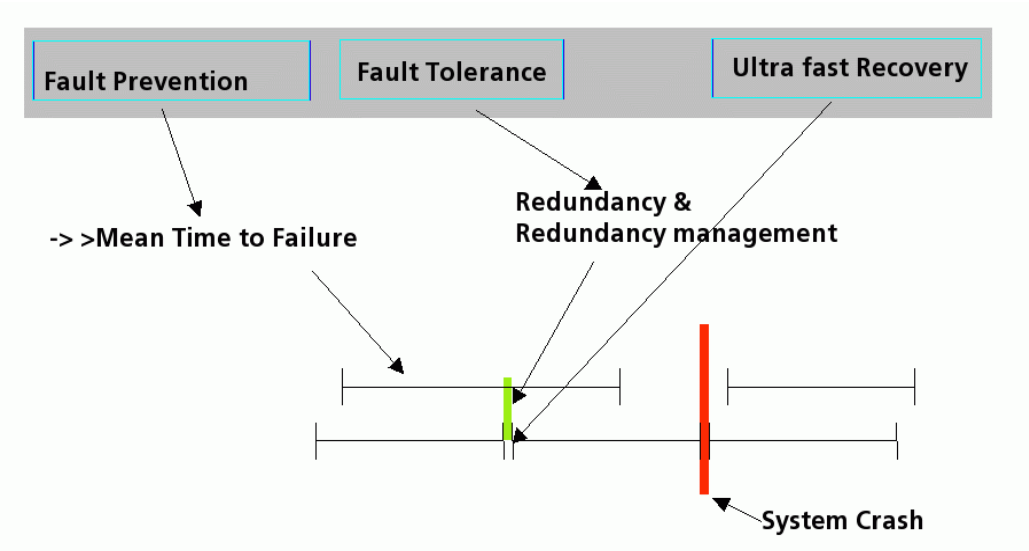

Figure 3: Increasing dependability

Today the recovery time is very long. Depending on the operating system it can range from

10 seconds to some minutes. This long period increase the probability of a total system failure if redundant nodes are down at the same time (see figure 5). Our target is to have a recovery time of a few milliseconds. If we can reach this target, we can tolerate several crashes per second without any problems.

3. Our approach

Contrary to most other approaches who try to minimise the possibility of failures, this approach considers that failures will occur and tries to handle them efficiently. Dependability is implemented using smart and efficient resource and redundancy management. The novel approach is to focus on efficient software solution instead on adding more hardware.

FhG FIRST has been working on the development of an operating system and its middleware (BOSS and TinyBOSS) specially designed to support dependability. Two major objectives are the ultra fast task recovery and ultra fast task migration mechanisms. Both functions are transparent to the applications.

An application with multiple tasks running on BOSS is distributed over a set of node computers. The communication between tasks is performed using named ports with a producer/subscriber protocol. The number of tasks, ports and the topology are transparent to the application. This allows redistribution of tasks without any programmer intervention. Consequently, any task can be replicated for redundancy purposes making the running system highly reconfigurable.

To achieve a fast recovery, each task saves its context in other nodes. When a node crashes, its tasks may be restarted (with communication links automatically rebuilt) at any other node with enough resources. To minimise the application's restart time, tasks are loaded preferable on nodes where the copies of their context already exists. Using the same mechanism, tasks can be migrated from node to node, for load balancing and power management purposes.

The major innovations of this approach can be summarized in the following: (1) Failures are seen as normal events that may occur anytime; (2) Dependability is achieved by cooperative assistance between software and hardware; (3) Complexity is maximally reduced by application distribution on parallel hardware and by transparent handling of tasks migration and restart by the operating system.