

Fault Tolerance Scenarios in Control Engineering

Klemen ŽAGAR

Cosylab, Teslova ulica 30, Ljubljana, 1000, Slovenia

Tel: +386 41 341-846, Fax: +386 1 426-18-79, Email: klemen.zagar@cosylab.com

Abstract: This paper presents the results of requirements analysis regarding fault tolerance in control engineering, which was performed in the context of the *Dependable Distributed Systems (DeDiSys)* project that is a part of the 6th European Framework Programme. The requirements analysis focuses on non-functional requirements, in particular availability, in presence of faults. The nature of faults, their consequences and mitigation mechanisms are discussed using several example scenarios. Mitigation mechanisms where availability improvements can be gained at the expense of reduced system consistency are given special attention, as the consistency/availability trade-off is the core research topic of the DeDiSys project.

1. Introduction

The primary objective of controls engineering is to allow supervision and control of physical devices and other machinery, for example power supplies, actuators, pressure and thermal sensors, video cameras, etc. Control is typically performed through dedicated control hardware, such as *Programmable Logic Controllers (PLCs)*, *Field-Programmable Gate Arrays (FPGA)* and, recently, the more flexible and higher-performing *Programmable Automation Controllers (PACs)*. Such control hardware is connected to the entities under control using serial connections, digital/analog cabling or other links of similar simplicity.

Due to constraints in cable length and the space that the cables may take, there is an ongoing trend where the control hardware (in this context also termed *Remote Control Units*, or RCUs) is interconnected with a communication network whose purpose is to deliver monitoring signals and convey control commands to/from an operator located at the central control headquarters. Such deployment allows the operators to supervise and control complex facilities with several thousand control points from a centralized control room, saving costs and improving efficiency (see Figure 1 for an example of operator's view).

Initially, dedicated communication networks were used for surveillance and control. In the last decade, *commercial off-the-shelf (COTS)* products such as the Ethernet, GSM, Wi-Fi and power-line carriers are gaining dominance in the controls domain as well, thanks to their increased availability, robustness and economic efficiency. In particularly harsh environments, *Industrial Ethernet* can also be deployed, as it offers resilience of cabling and connectors to temperature, vibrations, electro-magnetic interference, dust and water.

The term *Distributed Control System (DCS)* designates a system as characterized above: remote control units deployed near devices under control interconnected with a network and controlled from a central location. Similarly, *Supervisory Control and Data Acquisition (SCADA)* systems serve the same purpose. In the past, the differences were subtle: whereas SCADA systems were deployed over a wide geographical area (e.g., power distribution network), DCS were contained within a plant. However, with unification of the underlying technologies the SCADA and DCS systems are converging as well.

1.1 Applications of Distributed Control Systems

Monitoring of remote equipment (*telemetry*) is crucial for industries such as water supply and distribution, power and gas distribution, oil pipelines, etc. Through telemetry, maintenance cost and time required for repair are radically reduced, as the point of failure can be quickly pinpointed, thus improving the availability of the facility. Also, operations such as powering-off of a part of electrical grid or closing a gas valve can be performed quickly, within seconds, by the operator without leaving the control headquarters.

A variant of a DCS is employed in modern aircraft where it transmits pilot's commands to steering mechanisms on wings and the tail (*fly-by-wire*). One of the DCS tasks is to assist the pilot in guiding the plane, e.g., by preventing actions that would put too much stress on the plane's construction, as well as help the aircraft maintain aerodynamic stability.

The DeDiSys project is also interested in control systems of large experimental physics facilities, such as particle accelerators and radio-astronomy telescopes. In these scenarios, the DCS is fairly complex due to the large number of controlled devices and equipment that is at the bleeding edge of technology (ultra-low temperature control, superconducting equipment, nanosecond response times, etc).

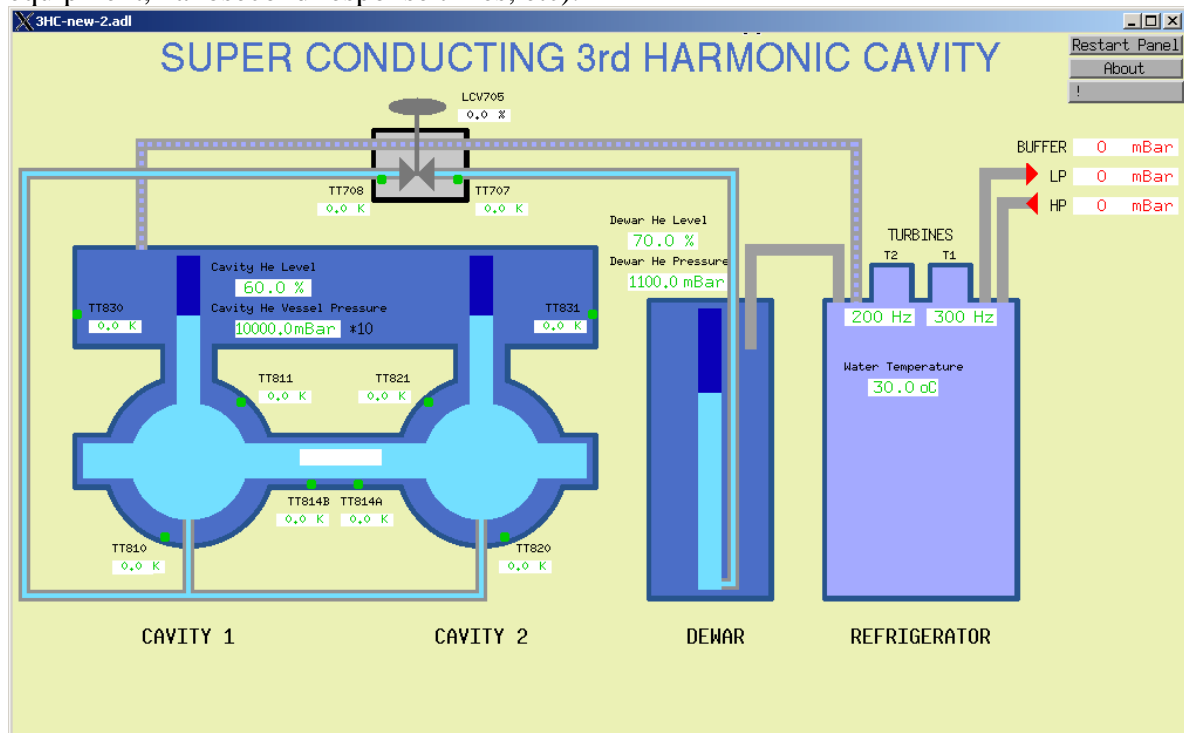


Figure 1: Screen-shot of a human-machine interface (HMI) for supervision and control of a complex device in a synchrotron light source. Such screens are designed in such a way to mimic the processes and physical layout of controlled machinery, whereas presenting the measured values alongside. The quality (e.g., stale data) and state (e.g., alarm threshold exceeded) of the measured values are colour-coded.

1.2 Fault Tolerance in Distributed Systems

In distributed systems data and services are located on more than one node (computer), and nodes are interconnected using a network to provide access to each other's resources.

If no additional measures are taken, this implies that many or all of the computers in a distributed system must be fault-free in order for the system to be available. For example, if a business application uses a client computer to present the user interface to the user, a server to handle the business logic, and a database to manage persistent store of the data, all three computers must be functional in order for a typical operation to be able to commence.

The probability of at least one computer out of three failing is approximately three times greater as the probability of a single computer's failure. Consequently, such a distributed system would be approximately three times less reliable (and thus also less available) than a single-computer solution. All three components in this case – the client, the server and the database – are *single-points-of-failure*.

To counter this effect, data and services are replicated. By replicating a resource on two computers, both computers must fail in order for the resource to be rendered unavailable. The probability of two computers failing, however, is significantly smaller than the probability of a single computer's failure. For example, if a single computer has a 10% chance of failing in, say, a week, two computers would both fail only with probability of 1% (10% of 10%), which is an order-of-magnitude improvement.

But replicas bring about complexities of their own. One of the most important issues is how to implement a mechanism that handles replication. If a replica on one host is modified, this update should propagate to all the other replicas. The propagation could be either *synchronous* (all other replicas are updated before accepting the modification of the local replica), or *asynchronous* (the modification is accepted whereas the other replicas are updated at some later time). The synchronous case ensures that all the replicas are consistent, whereas in the asynchronous case the replicas may become temporarily inconsistent. On the other hand, the synchronous case does not solve the single-point-of-failure problem described above: as soon as a host of one of the replicas is unavailable, the synchronous propagation of replica's state can not commence.

2. Objectives

The objective of this paper is to present the results of a user requirements collection and analysis related to distributed control systems. The goal of the analysis was to understand the specifics of distributed controls with regard to dependability and investigate the failure modes, mechanisms available for recovery from failures, opportunities for using traditional fault-tolerance techniques such as replication, and applicability of novel approaches to increasing availability, e.g., by compromising constraint consistency (Figure 2).

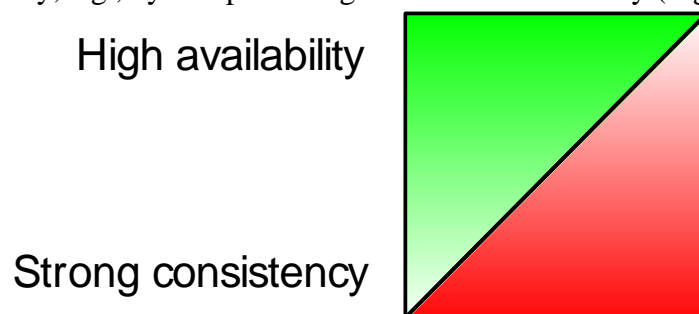


Figure 2: Graphical depiction of the availability/consistency trade-off. The greater the consistency requirements, the more likely it is that availability will decrease. In some cases, applications do not need the strongest consistency, but it does make sense to trade it against improved availability.

Based on this investigation, a prototype system will be built that will strive to demonstrate practical benefits of improving availability at the cost of consistency. If the prototype is successful, existing industrial control systems such as ACS would be adjusted to take advantage of the concept. Ultimately, an industrial control system that features configurable availability/consistency trade-off would allow for finding a point close to a cost optimum of a plant or facility where such a control system will be deployed.

3. Technology Description

The particular technology that served as the basis of our investigation is ACS. ACS (*Advanced Control System/Atacama Large Millimeter Array Common Software*) is an open-source component-oriented infrastructure for building distributed control systems.

Components representing controlled devices or control logic can be deployed across host computers throughout the network. A central entity called the Manager is responsible for determining on which host a given component will reside. This centralized approach to deployment allows dynamic reconfiguration of the system, e.g., due to changes in requirements or as an automated response to failures within the system.

Apart from managing the lifecycle of components, ACS also provides other infrastructural services, such as distributed logging, fault-detection, centralized configuration database, load-balancing and asynchronous publisher-subscriber communication mechanisms. ACS is built on CORBA middleware. ACS components and clients can be written in several programming languages (C++, Java or Python) and can be hosted by a variety of operating systems (many flavours of Linux, Solaris and Windows).

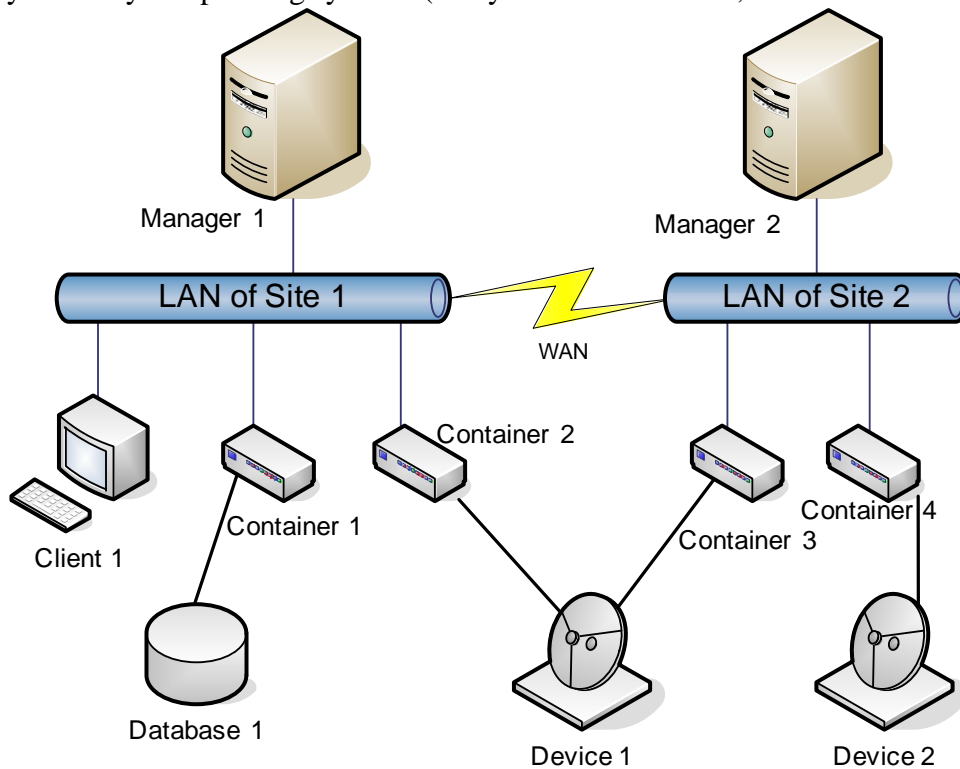


Figure 3: Example ACS Deployment. Two Sites Connected via WAN Link. Containers Capable of Hosting Components that Provide Application-Specific Logic (Drivers for Hardware Devices and Database Access). Client Access Any System Component Through Manager, Which Supervises and Controls the Deployment.

In ACS, like in any other distributed system, the individual nodes and their links are expected to fail. Depending on the criticality of the failed node, the ACS may be unable to continue with normal operations. In some cases, temporary disruption of a node has caused other parts of the system to malfunction as well. This avalanche effect is mostly due to improper development practices of the application developers, who fail to account for all the possible conditions that may occur in a distributed system. However, these issues should not be of application developer's concern – the ACS should provide reasonable fault-tolerant behaviour on the application's behalf.

4. Applications

The primary use of ACS today is in large experimental physics controls. In Karlsruhe (Germany), ACS is used to control an ANKA synchrotron light source [1], and will also be the underlying infrastructure for the *Atacama Large Millimeter Array* (ALMA, being built in cooperation between the *European Southern Observatory* and *North American Radio Observatory*), the largest array of radio-telescopes now under construction in the Atacama Desert [2][3]. Further applications of ACS include the controls for the 1.5 meter *Hexapod Telescope* (*Ruhr University Bochum*), the *Atacama Pathfinder Experiment* radio-astronomy antenna (APEX, *Max Planck Institute for Radio Astronomy*), the 40 meter radio-antenna of the *Observatorio astronomico nacional* [4] and the 64 meter *Sardinia Radio Telescope* [5].

The usability of ACS transcends the boundaries of distributed control systems. In essence, ACS is an infrastructure for component-oriented distributed systems, making it also suitable for use in classical business scenarios. Cosylab has successfully applied load-balancing features of ACS to build a scalable Geographical Information System.

5. Results

The following failure scenarios have been identified for the case of an ACS-based distributed control system:

1. **WAN link failure.** The link between two sites fails, essentially resulting in a **network split**. None of the subnets becomes entirely unavailable, as all the crucial services (e.g., the manager) are available in all of them. However, the subnets may evolve independently of each other, resulting in possible difficulties when they are re-joined (e.g., each of the subnets bringing up a service whose state must then be reconciled).
2. **LAN failure.** LAN failure may come in two forms: either link to a single host is lost, or the entire LAN infrastructure (e.g., a router or a hub) fails. The isolated node (or nodes) must decide what actions to take so as not to interfere with those of the rest of the system. For example, the node could decide to continue execution of its services (e.g., closed-loop control), or terminate. Similarly, the node should assume a reasonable line of action if it boots in an environment without network connectivity.
3. **Node crash.** A node crashes unexpectedly (e.g., a RAM or CPU failure). Unfortunately, to the rest of the system this kind of failure is difficult to differentiate from connectivity failure to the crashed node.
4. **Self-discovered fault of a node.** The node discovers a fault and terminates gracefully. During termination, it lets the rest of the system know of its condition.
5. **Manager unavailability.** Currently, the manager is a single-point-of-failure, as it can not be replicated yet. This shortcoming is not extremely critical, because the manager is capable of recovering its state upon restart, and its availability is a necessity only during start-up or reconfiguration of the distributed control system. However, to achieve higher availability rates for the entire system, the manager itself should also be replicated.

Apart from addressing these failure scenarios, a list of 44 requirements was compiled. The requirements call for a system that is:

- **Configurable** (e.g., the distributed resource locking strategy should be either optimistic or pessimistic).
- **Scalable** (at least 100 nodes should be supported, but the architecture itself should not impose an upper bound).
- **Maintainable** (advanced diagnostics utilities that allow quick pin-pointing of failures).
- **Manageable** (automatic collection of metrics such as *availability* for purposes of statistics and continuous improvement).

All of these features should be brought about by the middleware or a library, whose application programming interface (API) would be easy to use, but still allow flexibility (e.g. vetoing and application-level assistance during reconciliation of network splits).

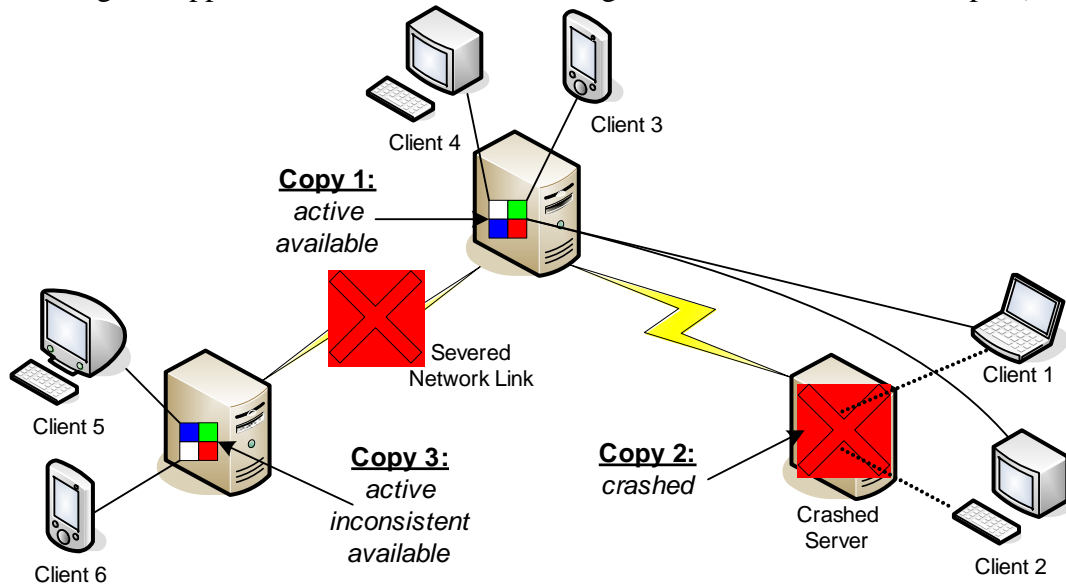


Figure 4: Presence of Network and Site Faults in Distributed System. Clients 1 and 2 can Switch to Copy 1 if Dedicated Server Crashes. If Network Link is Down, Copy 3 Still Available But Could Become Inconsistent.

6. Business Benefits

Proper understanding of fault-tolerance scenarios and requirements for mitigating them is important for improvement of distributed control system middleware. Fault-tolerant middleware would simplify development of control applications, making such development not only faster, but also provide a well-tested infrastructure that would resolve majority of fault-tolerance issues at the middleware level, reducing the risk of an inappropriate implementation at the application level.

The actual benefits to the end users depend on the amount of down-time they are subjected to due to faults in the distributed control system. In theory, a system that was 99% available (3.6 days of downtime per year) before the addition of fault-tolerance capabilities via replication would be 99.99% available (50 minutes per year) afterwards.

Being able to tune the consistency/availability trade-off would allow users to determine what level of consistency can be sacrificed so that the cost of inconsistencies does not exceed the benefits of improved availability. Thus, economic optimum of operations could be obtained during deployment without requiring system or configuration modifications.

7. Conclusions

The number of control points and the complexity of their interactions in control systems are increasing. To maintain a fair level of control, the resulting distributed control systems must remain dependable in spite of this increased complexity.

To improve dependability, effects of faults must be prevented from destabilizing the entire system, and possibly mitigated automatically to amend minor issues. Before attempting to resolve this problem, the nature of faults must first be thoroughly understood.

Therefore, the description of various fault scenarios, their classification, and recommendations for their mitigation are one major subject of this article. Apart from this, the article discusses the potential for improving availability by risking constraint consistency of the system's constituents. Finally, it proposes amendments to existing

middleware that would make the benefits of configurable availability/consistency trade-off available to controls developers without requiring much of their effort.

Within the context of the *DeDiSys* project, the scenarios discussed in this paper have been used to formalize functional and non-functional requirements that a dependable distributed system infrastructure should provide to applications. Currently, system architecture is being prepared that takes these requirements into account. In the implementation phase starting in October 2005, a prototype implementation of the proposed system will be prepared for various middle-wares, including CORBA-based ACS, J2EE and Microsoft .NET. In the final phase of the *DeDiSys* project ending in mid 2007, the prototype implementations will be validated against the formal requirements and subjected to the scenarios presented in this article. The impacts of trading constraint consistency against availability will be quantitatively analyzed. We are expecting to see an improvement of availability, ranging from none when no constraints are relaxed to an order-of-magnitude improvement when all constraints are relaxed.

References

- [1] I. Križnar, W. Mexner et al: *The Upgrade of the ANKA Control System to ACS (Advanced Control System)*, Proceedings of the *International Conference on Accelerator and Large Experimental Physics Control Systems 2003*, Gyeongju, Korea
- [2] G. Chiozzi, K. Žagar et al: *The ALMA Common Software (ACS): Status and Developments*, Proceedings of the *International Conference on Accelerator and Large Experimental Physics Control Systems 2003*, Gyeongju, Korea
- [3] Atacama Large Millimeter Array (ALMA): <http://alma.nrao.edu>
- [4] de Vicente, P. & Bolaño: *Software tools and preliminary design of a control system for the 40m OAN radiotelescope*, Proceedings of the *Astronomical Data Analysis Software and Systems 2003 Conference*, Strasbourg, France
- [5] Sardinia Radio Telescope: www.ca.astro.it/srt