

Integration of adaptable fault management techniques into a dependable middleware architecture

Rodica Tirtea

*K. U. Leuven, ESAT/ELECTA, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
E-mail: rirttea@esat.kuleuven.ac.be*

Abstract

Distributed automation systems such as the one used for the control of electric power infrastructure needs to fulfil requirements such as availability, reliability, safety, fault tolerance, real-timeliness, etc. Those systems have to be able to tolerate changes in the environment and faults. They rely on monitoring mechanisms to identify changes in the resources or the environment to make the appropriate trade-offs (e.g. from cost point of view) for a fast reaction or reconfiguration.

This paper presents the integration of a resource monitoring mechanism into a dependable middleware architecture. A mathematical model generates a composite indicator for system performance based on the different monitored parameters. The composite indicator allows dynamic switching of the recovery strategies based on the current state of the distributed system environment. The monitoring mechanism can also be used for proactive fault management techniques.

1. Introduction

Through verification, validation and testing not all faults can be eliminated at design stage. This is due to the dynamic nature of the distributed architecture (nodes can be removed or added, new connections established, or lost connections occurring) and also due to the interactions and the complex interleave of the messages exchanged in distributed systems [1]. Occasionally processes crash or hang. Software errors are also triggered by peak conditions in workload, etc. [1].

In this context, for the system to be able to tolerate changes in the environment and faults, monitoring mechanisms are required. Also the fault management actions should be adapted at run time. Based on

quantitative analysis of the monitored systems reactive fault management actions (e.g. fault containment, recovery strategies) or proactive fault management actions (e.g. software rejuvenation [2]) can be triggered.

Recovery strategies in distributed systems include actions such as migration/restart of processes/tasks on other nodes. The appropriate recovery strategy is usually selected based upon the type of failure and the circumstances of the failure event. The available resources of the future host (for the migrating processes) are usually not taken into account, neither is the performance of the network. In [3, 4] we show that those run-time conditions, if neglected, could negatively impact the reliability of the fault-tolerant systems.

2. The target architecture

Before going further, a short introduction is given for the target distributed software architecture used in the control of electrical power infrastructure. The distributed system is composed of a collection of sites with distributed location. Each site is composed of nodes (hosts) and they are interconnected by an intra-site dedicated communication network (see figure 1)[5]. The sites are interconnected via gateways by an inter-site communication network, which can be through e.g. Internet or dedicated channels (e. g. dial-up, satellite, others).

In case of our target architecture, the fault-tolerant operation assumes that the system will deliver its service even in presence of faults. In case of severe situation, according to the fault management and safety requirements, a graceful degradation of service is required.

A predictive response interval is required to assure the real-time behavior of the architecture. In this context, all fault management actions (e.g. fault detection, fault containment, recovery action, etc.) should have bounded time limits.

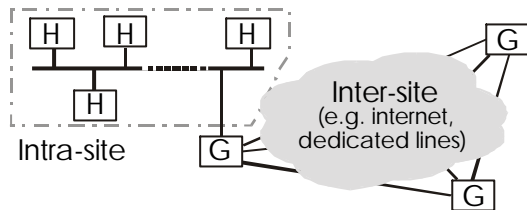


Figure 1. Distributed system: intra-site and inter-site levels (G =gateway, H =host).

3. The middleware option

A middleware architecture is able to address different requirements depending on the location in the distributed system. At this level it is possible to develop an architecture with detection, recovery and adaptation mechanisms for the control systems to be able to fulfil their tasks in a timely manner, even in the presence of faults and changes in the environment.

Depending on the criticality of each system layer, the corresponding middleware component realizes a different trade-off between the target properties. An example of trade-off for real-time would be mapping hard real-time requirements to the local site level of the architecture (e.g. Ethernet), but not to the inter-site level (e.g. Internet)[1].

In [6, 7] different available middleware architectures are surveyed and their support for fault tolerance, real-time and quality-of-service (QoS) in context of target application requirements is presented.

Analysis of the underlying distributed automation system of the electrical power infrastructure shows that different levels of QoS are required for the inter-task communication. From the highest level dedicated to signaling and recovery traffic - regarding commands and alerts among sites and to orchestrate recovery steps - to the lowest one used to configure or manage processes. Similarly, not all tasks within an application are equally critical, e.g., for survivability, specific tasks should always run while others, less critical ones, may be rescheduled [5].

In order to meet the QoS requirements, there is a need for a mechanism to monitor the performance characteristics and availability for the communication network (e.g. bandwidth) and the nodes (e.g. available resources). Based on this monitoring mechanism, different strategies can be deployed to adapt or negotiate the communication and the priority of the tasks depending on the available resources.

Those data supplied by the monitoring mechanism can be used further into fault management techniques,

improving this way the fault-tolerance of the system. How this can be used for recovery adaptation (i.e. selection of recovery actions and switching of recovery strategies [4]) is presented in Section 4.

The monitoring mechanism increases the differentiation between node crash and network problems for failure suspected nodes. Another advantage of using this mechanism is the dynamic adaptation of resource allocation for an overall increase in application availability.

4. Recovery adaptation

As mentioned in the introduction, a requirement of the target architecture is adaptability to changes in the environment. This requires a monitoring mechanism for the most important factors that can influence the normal functionality of the distributed system.

In [4] is shown how fault management, especially the recovery strategies and actions can be adapted at run time based also on the state and load of the components involved, not only on the circumstances of the fault. Also in [4] a mathematical model for generating a Composite Indicator based on sampled parameters is presented. The mechanism for monitoring resources at the node level is described and it is presented how this can be used in the selection of a recovery action (e.g. restart/migration of processes on/to overloaded nodes should be avoided). Based on the communication characteristics between distributed sites (e.g. depending on availability or on cost), different recovery strategies can be switched.

Different strategies can be designed to handle the changes in the environment, which have impact on communication. For testing purposes are used two strategies, a low cost one, for which the communication via internet is sufficient and a second one using a high cost connection via a dedicated line in case in which the Internet connection is lost or overloaded (e.g. due to denial-of-service attack).

Those recovery strategies are switched at run-time [8] based on the environmental conditions. The actions within the recovery strategies are executed in case of error detection (e.g. timeout of the watchdog) or event notification (e.g. just a dummy request for checking the conditions). The recovery actions are selected based on the context of the detected errors and on the available resources of the nodes in case of tasks/processes migration/restart [3].

Choosing the best strategy for recovery helps in different ways: avoiding the overload of an another node, minimizing in this manner the probability of a possible subsequent crash or delay, which could jeopardize the execution of critical tasks [4].

5. Using monitoring tools for proactive fault management

Fault management techniques can be divided from the perspective of the time of the fault occurred in proactive respectively reactive techniques.

The category of proactive techniques includes software rejuvenation, and predictive failure signaling. Recovery oriented computing (ROC) can be used to assure a fast restart/reboot of task and nodes and to have a fault free state of the system again.

Even if those techniques are called 'proactive', they are still triggered by faults, but the faults and their number did not yield a failure of the service yet.

In context of distributed systems, software rejuvenation can be done with lower costs due to the possibility of reallocation of resources/tasks/processes so that certain nodes can be rebooted. In this way, if back-up nodes are available for all type of applications running on all the nodes, the cost of a node downtime can be minimised, the service being supplied also during rejuvenation of certain nodes. In this case, the cost formula from [9] can be extended for redundant applications. From user perspective, this can be unnoticed, especially if the action is done during low workload.

ROC proposes detecting and recovering from failures, rather than preventing them [10]. The focus of this technique is to reduce Mean Time to Repair (MTTR) instead on only increasing Mean Time to Failure (MTTF). Four techniques are proposed in [10] as basis for ROC: redundancy and isolation, online self-testing and verification, support for problem diagnosis and concern for human/system interaction.

Other aspects that can be benefit from the proactive fault monitoring applications are the dynamic resource provisioning for the power management of e.g. handheld devices.

Also security can benefit from network monitoring. Enhanced intrusion detection can be achieved in case of attacks that lead to resource exhaustion (e.g. Denial-of-service attack).

6. Future work

Next step will be to identify which types of faults can be detected or tolerated based on resource monitoring approaches in case of our application.

The coverage of those faults is important in order to make the trade-off cost/benefit analysis. In context in which, for normal operation/functioning the monitoring mechanism is already required (e.g. in Grid systems), then the cost of monitoring implementation

and of the overhead is not only for fault management. In this way the fault management techniques developed based on monitoring can be seen as a gain in the system dependability.

7. References

- [1] Y. Huang and C. Kintala, "Software Fault Tolerance in the Application Layer", M. R. Lyu, (Ed.), "Software Fault Tolerance", John Wiley & Sons, 1995.
- [2] Y. Bao, X. Sun, K. Trivedi, "Adaptive Software Rejuvenation: Degradation Model and Rejuvenation Scheme", Proc. of Dependable Systems and Networks (DSN), San Francisco, California, June, 2003; pp. 241-248.
- [3] R. Tirtea, G. Deconinck, V. De Florio, R. Belmans, "QoS monitoring at middleware level for dependable distributed automation systems", 13th Intl. Symp. on Software Reliability Engineering (ISSRE 2002), Annapolis, MD, November 12-15, 2002; pp. 217-218.
- [4] R. Tirtea, G. Deconinck, V. De Florio, R. Belmans, "Using resource monitoring to select recovery strategies," 2004 Annual Reliability & Maintainability Symposium (RAMS2004), Los Angeles, California, USA, January 26-29, 2004; pp. 266-271.
- [5] G. Deconinck, V. De Florio, R. Belmans., G. Dondossola, J. Szanto, "Integrating recovery strategies into a primary substation automation system", Proc. Of Int. Conf. on Dependable Systems and Networks (DSN), San Francisco, California, June , 2003; pp. 80-85.
- [6] R. Tirtea, G. Deconinck, "Reusing middleware for a dependable power quality assessment system", IEEE Young researchers symposium in electrical power engineering - Distributed generation (CD Rom), Leuven, Belgium, February 7-8, 2002; 5 pages.
- [7] R. Tirtea, G. Deconinck, V. De Florio, "Le logiciel d'intermédiation: Un panorama critique et réutilisation dans les systèmes d'automatisation enfouis à sûreté de fonctionnement", *Genie logiciel*, Paris, No.61, June, 2002; pp. 2-8.
- [8] G. Deconinck, V. De Florio, O. Botti, " Software-implemented fault-tolerance and separate recovery strategies enhance maintainability", *IEEE Trans. On Reliability*, Vol.51, No.2, March, 2002; pp. 158-165.
- [9] Y. Huang, C. Kintala, N. Kolettis and N. Fulton, "Software Rejuvenation: Analysis, Module and Applications", IEEE Intl. Symp. on Fault Tolerant Computing, FTCS 25, Pasadena, California, June 27-30, 1995; pp.381-390.
- [10] D. Oppenheimer, A. Brown, J. Beck, D. Hettena, J. Kuroda, N. Treuhaf, D.A. Patterson, K. Yelick, "ROC-1: Hardware Support for Recovery-Oriented Computing", *IEEE Trans. on Computers*, Vol. 51, No. 2, February 2002; pp. 100-107.