ROBUSTNESS TESTING OF HIGH AVAILABILITY MIDDLEWARE SOLUTIONS

Zoltán MICSKEI
Advisor: István MAJZIK

I. Introduction

Lately dependability became a key factor even in common off-the shelf computing platforms. High availability (HA) can be achieved by introducing *manageable redundancy* in the system. The common techniques to achieve minimal system outage can be implemented independently from the application, and can be exposed as a *HA middleware*. Recently the standardization of the functionality of such middleware systems has begun (for example the AIS [1] specification, and its open source implementation, OpenAIS). The benefit of an open specification would be for example the easier integration of different off-the shelf components.

With multiple products developed from the same specification the demand to compare the various implementations naturally arises. The most frequently examined properties are performance and functionality, but especially in case of a HA solution the *dependability* is also, or even more important. This paper outlines an approach to *compare the robustness* of HA middleware systems.

II. Related work

Robustness is defined as the degree to which a system operates correctly in the presence of *exceptional inputs* or *stressful environmental conditions*. In the past ten years several research projects examined the robustness of different kinds of applications.

Earliest works used software implemented fault injector tools to simulate hardware faults. Console applications were tested using randomly generated streams searching for input combinations that can crash the system under test. In *Ballista* [2] the POSIX API was examined, and the robustness of fifteen POSIX operating systems was compared. The method used in Ballista was to develop for each type in the API a test generator that can produce valid and exceptional values. The API functions were called with the combinations of the values returned by the generators. The goal of the European Commission project *DBench* [3] was to define a general framework for dependability benchmarking. The procedure of the benchmarking is to characterize a workload representing normal operation and a faultload with injected faults.

III. Robustness testing of HA middleware systems

One of the earliest phases of developing a test strategy is to identify potential places where faults can occur in the system, i.e. develop the fault model of the system. Figure 1 illustrates a typical node in a HA distributed system and the identified specific fault types.

1. *External errors*: These errors are application-specific, thus they are not included in our tests.
2. *Operator errors*: In general, operator errors mainly appear as erroneous configuration of the middleware and erroneous calls using the specific management interface.
3. *API calls*: The calls of the components using the public interfaces of the middleware can lead to failures if they use exceptional values, e.g. NULL or improperly initialized structures.
4. *OS calls*: the robustness of a system is also characterized by how it handles the exceptions returned by the services it uses, in this case the operating system errors.
5. *Hardware failures*: The most significant HW failures in a HA middleware-based systems are

host and communication failures (that has to be tolerated in the normal operating mode) and lack of system resources.

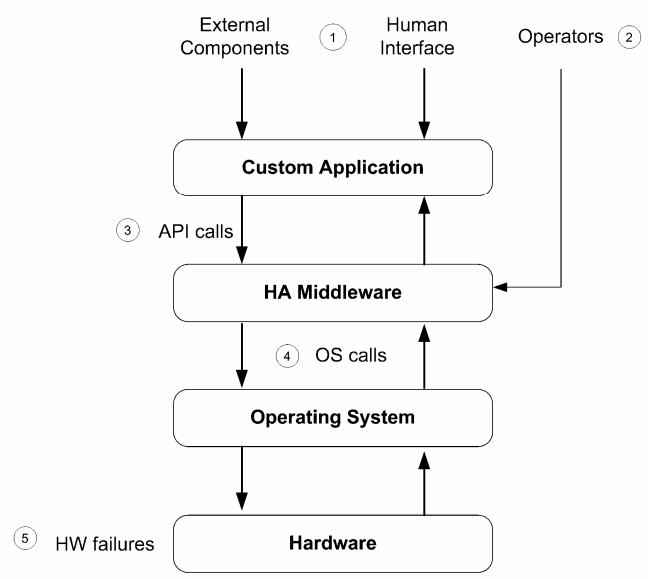

Figure 1: HA middleware fault model

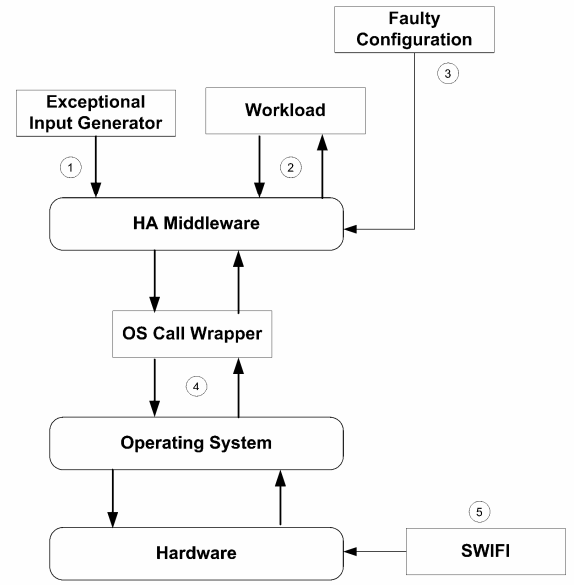

Figure 2: Architecture of the test environment

According to the fault model described above we developed the method illustrated in Figure 2 to test the robustness of a HA middleware. The method can be implemented in two separate phases.

- Phase I: testing exceptional inputs in API calls with the following techniques (1):
  - Generic input generators using fixed domain of input for all types.
  - Type-specific exceptional input generators using the knowledge of the specific types.
  - Scenario-based testing using the sequence diagrams in the specification to reach such system states in which exceptional inputs can be provided.
- Phase II: testing stressful environment conditions. The steps needed for this are the following.
  - Defining a workload according to the normal operation (2).
  - Constructing a faultload representing various faults from the environment (3, 4, 5).

To demonstrate the feasibility and efficiency of this method we developed tests for the OpenAIS middleware. Code examples were created with generic and type-specific input generators, and robustness test cases were generated from the available functional test cases and sequence diagram specifications using *mutation techniques*. The tests found several robustness failures, mostly related to improper pointer handling (e.g. null pointers). The generation of test cases with generic testing was partially automated using templates.

IV. Conclusion

In this paper we examined methods used for robustness testing of different implementations of the same specification of a HA middleware. We presented the fault model and proposed a process to test the robustness of the middleware at different layers. The implementation of the test programs has been started based on OpenAIS with promising preliminary results.

References

[1] Service Availability Forum, URL: http://www.saforum.org/

[2] P. Koopman et al, "Automated Robustness Testing of Off-the-Shelf Software Components," in *Proc. of Fault Tolerant Computing Symposium*, pp. 230-239, Munich, Germany, June 23-25, 1998.

[3] K. Kanoun et al, "Benchmarking Operating System Dependability: Windows 2000 as a Case Study," in *Proc. of 10th Pacific Rim Int. Symposium on Dependable Computing*, Papeete, French Polynesia, 2004.