Using Software Health and Quality Indicators

Janusz Sosnowski, Jakub Machnicki, Marcin Król
Institute of Computer Sscience, Warsaw University of Technology, Warsaw, Poland
jss@ii.pw.edu.pl

I. INTRODUCTION

Developing complex software applications we face various reliability and quality problems. They are taken into account during all phases of the design and implementation processes, starting from the specification of requirements. These issues are widely discussed in the literature and have resulted in some recommendations of good practices in software production steps. They can be considered as the necessary conditions to improve software reliability and quality. In complex systems we are not able to preview all aspects, which may impact their operation e.g. static and dynamic properties of hardware/software platforms; environment, workload and user profiles; upcoming modifications and updates. These characteristics are preliminary specified for the designers but in the running system may change, moreover their primary specifications may be not sufficiently accurate (e.g. usage profile, system load). So the design process should take into account some uncertainty level and assure future adaptation to real conditions. Hence, an important issue is to provide self-control during the system operation. The goal of this self-control is assessing the reliability and quality of delivered services. This should be combined with built in mechanisms detecting (or predicting) problems and further used in recovery procedures (or new software versions).

We have gained some experience with collecting operational logs and monitoring performance of many computer systems [1]. We have found that this technology can be adapted to resolve the above mentioned problems. An important issue is to define appropriate indicators related to reliability and quality of service, specify schemes of their monitoring and analysis (section 2). This approach we have used in the developed transaction oriented applications implemented in a distributed environment (section 3).

II. THE SPACE OF INDICATORS

To provide automatic indication of system well-being (within the reliability and quality of service aspects) we can embed various monitoring mechanisms which collect and report parameters or events related to the observation goals. We will call them indicators. In general we distinguish direct and indirect indicators. The direct indicators present specified reliability or quality properties e.g. number of crashes, MTTF, distribution of service times, statistics of time-outs. Indirect indicators present some properties which can be correlated with the goal of observation e.g. percentage of processor or memory usage, statistics of registered system events.

Taking into account the implementation aspects we can have indicators handled by hardware or software. Hardware mechanisms mostly relate to hardware faults but some software faults can also be detected e.g. certain system exceptions such as overflow, invalid opcode, memory access violation. Software mechanisms can be related to system or application level.

Monitoring system operation we can concentrate on application properties at the user, system or specified resource levels. The first approach is application dependent and may relate to such parameters as transaction response time, frequency of access denials, frequency of data losses, scope of these losses, etc. They give some general assessment, however many erroneous situations may be hidden at this level and revealed with significant delay. Embedding into application some assertions we can identify improper control flow, violation of invariants, etc. Many problems can be visible at the level of the application program interaction with the operating system and other resources (CPU usage, available RAM, I/O utilization). These issues are much more difficult to analyze and need long term observations as well as many indicators.

Most operating systems provide various logs on the system operation, and this is a source of data on system hardware, software and configuration problems. Typically we have the following logs [2]: *security log* (events related to security and auditing processes), *system log* (diagnostic messages, abnormal conditions, events generated by system components), *application event log* (errors that occur during the application execution e.g. failing to allocate memory, aborting the transfer of a file). In addition, it is possible to collect data characterizing various system operation facets e.g. component activities, their state and performance changes, resource utilization, the network traffic statistics, overall performance information, etc. [1,3]. Available auditing/logging programs provide information on events generated by system programs such as login, logout for various services (e.g. network), type and time of the event, process requesting the event, object accessed or user generating the event, etc. Performance counters count various events representing the state of specified system entity (e.g. main memory, disc). Moreover, it is possible to create own counters attributed to various application programs and embed software sensors that collect specific data from the program.

In the case of distributed applications we have an additional problem related to collecting indicators within many computers and correlate them. Moreover, long term application operation may be influenced by configuration or

environment changes, updates of application or other software modules. Some of these aspects are not directly reported in standard logs, but it is reasonable to add logs filled by administrators and system users. They can be correlated with automatically generated data for logs.

The identification of problem sources in general is quite complex, the more that it is not clear if the problem relates to hardware or software or wrong usage, lack of sufficient resources, bad configuration, etc. This can be resolved by correlating different indicators. Moreover, we have to initiate monitoring in different time perspectives: detailed monitoring for short periods, medium and long term observations. In higher time perspectives we are targeted at checking trends of the measured parameters, here we concentrate on selected aggregated properties e.g. average, minimal and maximal parameter values for longer time periods, frequency and distribution of parameter spike values, time segments corresponding to anomalous behavior. For an illustration we give statistics of system restarts in 4 didactic laboratories and their qualification based on event log analysis.

TABLE I. RESTART DIAGNOSIS BASED ON EVENT LOG.

	CASE	NET	PR1	PR2
days	144	144	135	90
restarts	500	431	443	276
updates	20.2%	16.5%	22.6%	24.3%
applic.	19.2%	29.7%	12.0%	29.6%
hardware	1.2%	0.5%	2.0%	0.4%
ambig	1.4%	5.6%	5.0%	5.4%
unknown	59.4%	53.4%	63.4%	45.7%

About 20% of restarts related to program updates (e.g. antivirus). Hardware problems occurred scarcely. A relatively large percentage of restarts was unknown or ambiguous. It was not possible to identify them using only event logs (mostly used in the literature [2]). They were diagnosed with additional data provided by monitoring selected performance parameters. Moreover, we found useful capturing messages reported to the users (normally not recorded) and notes describing activities of administrators. Increased accuracy of problem identification needs combined analysis integrating different levels of monitoring. Here still we need further studies to identify correct and anomalous behavior. Some known abnormal situations are relatively easy to define in terms of observed indicators. The most difficult is specifying conditions to detect unknown anomalies and develop early warning (predictive) conditions.

III. APPLICATION

We have implemented the presented ideas in the distributed system handling different transactions (from over 300 000 users). The transactions are handled by up to 16 servers. Transaction requests are directed to the servers so as to assure equal load balancing and eliminate faulty or inoperative nodes. The system operation is monitored by an additional server via internet using GNU GPL v2 Zabbix program (www.Zabbix.org). The Zabbix server collects data from agents installed in other nodes. Each production node (Unix platform) comprises Zabbix agent which collects preconfigured performance data, traces specified warning levels, critical events and sends them on request (or autonomously) to the server. The monitoring server comprises a database for storing the collected indicators from all the production computers. We have integrated the Zabbix server with a powerful analyzer which systematically traces short and long term profiles, finds correlations and generates warning signals, causes switching off faulty nodes, etc. The collected performance data is also used by the system manager to balance the processing load. Having found typical behavior of selected parameters we can identify also external attacks on the system (e.g. DoS attacks [3]). For some known attacks it is possible to identify statistical anomalies of the most sensitive performance variables. Similar solution we have developed for another transactional system based on Windows platform. The integrated analysis of various operational facets collected within 6 months allowed us to identify some critical situations, program and supplementary script errors, etc.

To get better knowledge of the sensitivity of monitored indicators we have performed many experiments with faults (hardware and software) injected into applications. Hardware faults are injected by disturbing the states of registers and memory areas (bit flips). We also check the impact of available resources by artificially consuming some percentage of CPU power or available memory. Typical software errors are simulated according to program mutation techniques. This process allowed us to select the most sensitive indicators.

The process of failure detection and identification can be enhanced with failure prediction and preemption. Many practical reports proved that quite often a system goes through a series of state deviations (or errors) before getting crashed. So an important goal of the log analysis is to identify accompanying failure symptoms in the log reports or other operation report files. This can help to predict upcoming failures. Such task is quite complex due to large volume of reports and their informal textual nature. Moreover, the messages or reports can be lost or corrupted under failure or heavy load, this results in some data incorrectness or vagueness, imprecision, etc.

REFERENCES

[1] M. Król, J. Sosnowski, Multidimensionl monitoring of computer systems, Symposium and Workshops on Ubiquitos and Trusted Computing, IEEE Comp. Soc. 2009, pp.68-74.

[2] W. Peng, Ch. Peng, T. Li,, H. Wang, "Event summarization for system management", *Proc. of ACM KDD'07*, 2007, pp. 1028-1032.

[3] N. Ye, *Secure Computer and Network Systems*, John Wiley& Sons, Ltd., 2008.

This work was supported by MNiSzW granr 4297/B/T02/2007/33