

# **A survey on online monitoring approaches of computer-based systems**

*Prepared by:* Vladimir Stankovic, Lorenzo Strigini

*Date:* June 2009

*Version:* 2.6

This report<sup>1</sup> surveys forms of online data collection that are in current use (as well as being the subject of research to adapt them to changing technology and demands), and can be used as inputs to assessment of dependability and resilience, although they are not primarily meant for this use.

## 1. Introduction

Monitoring the components of a system can be used to make decisions about the management of the system and thus control its behaviour. Monitoring is also used for debugging and for evaluating in real time the performance or quality of service (QoS) of a system. Monitoring in IT systems can be defined as the process of dynamic collection, interpretation and presentation of information concerning objects or software processes under scrutiny [Mansouri-Samani and Sloman 1992]. This paper provides a general description of monitoring: “*the behaviour of the system is observed and monitoring information is gathered; this information is used to make management decisions and perform the appropriate control actions on the system*”. When setting up a monitoring system, technical problems have to be solved in relation to detecting the events of interest, labelling these events with sufficient information for use in measurement, transmitting the event notices to where they will be used, filtering and classifying the events according to the measures of interest, and acting upon the collected information. Online monitoring has long been recognised as a viable means for dependability improvement [Schroeder 1995]. This report surveys the use of monitoring in diverse areas of: dependability, performance enhancement, correctness checking, security, debugging and testing, performance evaluation and control.

In general, when we deal with the monitoring of computing systems we have one or more *monitored elements* (e.g., computers, operating systems, application packages) send out *signals* or *measurements* of their well-being or distress to a *monitoring site* (logically centralised, even when computing elements are distributed), which thus can extract (or help humans to extract) various forms of processed information and make (or help humans to make) decisions. Concrete examples include:

- A computer manufacturer receiving *health/failure data* at a centralised maintenance centre from customer installations of its machines.

---

<sup>1</sup> A version of this technical report is included in the State-of-the-art deliverable of the EU-funded Coordination Action AMBER (Assessing, Measuring and Benchmarking Resilience), <http://www.csr.city.ac.uk/projects/amber.html>

- An independent company that installs and maintains machines at client sites polling/receiving *health data* from them.
- A software vendor/distributor receiving reports of failures from its installed software.

Monitoring may serve either or both of the following two purposes:

- Supporting direct feedback control: the monitoring data trigger decisions about, for example, dispatching maintenance technicians; or switching in stand-by components; or switching to a higher-security configuration of a system; or asking developers to investigate a suspected software bug; etc. That is, monitoring helps to improve the monitored system so as to make it more resilient/dependable.
- Providing data for assessing the dependability or resilience of the monitored system (which is the focus of this report).

The rest of this chapter is organised as follows. First, we provide a brief timeline description of monitoring activities. Subsequently, the “Research in Distributed Systems Monitoring” section outlines important issues in *monitoring distributed systems*. In the following sections, we group *forms of online monitoring* according roughly to areas of use, which determine communities of users and researchers. For each such area, we describe pertinent industrial and academic work, focusing on its potential for dependability assessment. These “areas” are not intended as a partition of all related work, based on first principles, but just as a convenient clustering of existing practices, research and literature into (possibly overlapping) categories.

### **1.1. A concise historical perspective of monitoring**

The field of monitoring and related telemetry activities for dependability assurance can be regarded to be as old as the electronic engineering discipline.

In the early 20<sup>th</sup> century the design and implementation of dependable systems have not been rigorously constrained by cost and schedule limitations as is the case nowadays - “*In many cases, high levels of reliability were achieved as a result of over-design*” [Smith 2001]. Quantified dependability assessment was, therefore, largely non-existent.

The maintenance procedures for early days’ computers consisted of a technician inspecting the state of the vacuum tubes. “*Computers built in the late 1950s offered twelve-hour mean time to failure. The vacuum tube and relay components of these computers were the major source of failures; they had lifetimes of a few months.*” [Gray and Siewiorek 1991]. These tasks were daunting, and in the case of a large number of simultaneously faulty vacuum tubes

almost impossible, since the locations and causes of the errors were difficult to determine. The reliability and ease of maintenance have been significantly improved by using various tools, such as cathode-ray oscilloscope [Chandler 1983], as in the case of Colossus, the famous World War II codebreaker machine that had to provide a 24-hour service. Signal voltages were examined with an oscilloscope and it was used to assess the effect of relative timings of different parts of a circuit.

The post-war age hastened the advent of intricate mass-produced component parts, which brought with them greater complexity and parameter variability. This resulted in poor field reliability of the assembly-line products. As a result, for example, the focus of military equipment in the 1950s shifted to the collection of field failure data of safety-critical systems and the interpretation of the corresponding test data. Databases holding failure rate information were created by such organisations as UKAEA (United Kingdom Atomic Energy Association), RRE (Royal Radar Establishment), RADC (Rome Air Development Corporation, USA), [Smith 2001].

An ability to detect, log and react to dependability-relevant events has been part of the design of, at least, high-end dependable computers since the 1970s and 1980s. During these decades, several computer vendors (IBM, Tandem, Digital Equipment Corporation (DEC), etc.) used electronic communication (usually a secure telephone link) to maintain and service their customers' computer devices. As a result, data for maintenance as well as for statistical studies of field dependability were made available. For instance, IBM offered customers of its 3090 high-end mainframe series (which appeared around 1985) an option for automatic remote support capability (an automatic call was initiated to IBM) or a field service engineer was sent to the customer's site after a maintenance job has been raised due to a *processor controller*<sup>2</sup> detecting errors (see [Siewiorek and Swarz 1998, Chap. "General-purpose Computing"]). All this information was logged in a central database called RETAIN, and, for example, a service engineer could communicate with it to obtain the latest maintenance information. Another option offered with the 3090 series was that a field technical support specialist could establish a data link to a processor controller to remotely monitor and/or control the mainframe. A similar option existed as a part of the well-known VAX system from DEC (VAX-11/780, the first member of the VAX computer family, was introduced in 1977). The user mode diagnostic software of the VAX system reported errors, which initiated

---

<sup>2</sup> Processor controller is a unit of, usually mainframe, computers, which handles the functions of maintenance, monitoring, and control.

calls to the DEC's Remote Diagnostic Centre (RDC). Subsequently, an RDC engineer would establish a data link to the customer's VAX system through a special remote port and execute diagnostic scripts, logging everything during the process (see [Siewiorek and Swarz 1998, Chap. "General-purpose computing"]).

Other examples of mature computer-based technology that employed forms of online monitoring for achieving high dependability include:

- Telephone switching systems from the pioneer in the field, AT&T Inc. For example, "*the 3B20D processor diagnostic capabilities detect faults efficiently, provide consistent test results, protect memory content, allow automatic trouble location and are easy to maintain*", see [Siewiorek and Swarz 1998, Chap. "High-Availability Systems"]. These systems employ dynamic redundancy (e.g., resuming a computation on another processor once an error is discovered) as the means for achieving high availability. Error detection techniques are numerous. The high availability processors of the 3B series (3B20D/3B20C/3B21D/3B21E) were first produced in the second half of the 1970s. The 4ESS switch, the first digital electronic toll switch for long distance switching (introduced in 1976), initially used the 3B20D processor. It is worth noting that 4ESS employed advanced error reporting techniques. The switch had automated procedures in place for analysing error data, with the purpose of correlating and identifying intermittent errors which evaded standard diagnostic mechanisms.
- Long-life unmanned spacecrafts, like deep-space planetary probes *Voyager* and *Galileo*, which use block redundancy for fault tolerance. They are capable of performing error detection, diagnosis and reconfiguration automatically (on the spacecraft) or remote control (from ground stations), see [Siewiorek and Swarz 1998, Chap. "Long-life systems"] for more details.

Following the use of, most commonly, secure telephone links for collecting customers' failure reports in the 1970s and 1980s, a natural step forward was to automate the collection of failure data: failure data of computers/applications were automatically collected and sent back to the manufacturer/vendor, after what the data were forwarded to the engineering department for analysis. Consequently, much research on dependability has been based on data from automated error logging ([Iyer 1995] lists many early examples), demonstrating how measurement can inform modelling to lead to assessment (prediction) of dependability in future operation.

The technological advancement for reliability improvement has been noticeable in the hard disk drives industry in the 1990s, when IBM introduced Self-Monitoring, Analysis, and Reporting Technology (S.M.A.R.T.) and the closely related Predictive Failure Analysis (PFA) technology. Dependability monitoring mechanisms are now often packaged in off-the-shelf components, e.g., hard disk drives. Nowadays, many hard disks are equipped with S.M.A.R.T., which detects and reports various reliability indicators and administrators can use it for maintenance and potentially for reliability assessment. As computer systems became increasingly distributed and integrated with networks, this kind of monitoring has taken on more varied forms.

## 2. Research in Distributed Systems Monitoring

In this section we provide a general introduction to the problem of *monitoring* and *on-line management* of distributed systems, with an emphasis on academic research.

Distributed systems are more difficult to design, build and monitor, than centralised systems because of i) parallelism among processors, ii) random and non-negligible communication delays, iii) partial failures (failures of components of the system) and iv) lack of global time base and strict synchronisation between distributed nodes. The main issues of monitoring distributed systems are as follows:

- *No central point of decision making*: the process of making decisions in a distributed system may itself be distributed (leading to, for example, the “agreement problem”).
- *Incomplete observability*: in some cases it is not possible to observe certain parts of the system, resulting in incomplete (and possibly inconsistent) information about events in the system. Thus usually, locally observed events are collected and some entity, using information about these events, has to build a global view of the system (an example of data fusion problem).
- *Non-determinism*: distributed, asynchronous systems are inherently non-deterministic. Thus, two executions of the same program may produce different, but correct, orderings of events. This makes the reproduction of errors and the re-creation of test conditions difficult.
- *Vast data volumes*: The huge amounts of data produced by monitoring have to be processed (possibly in real-time, so as to perform short-term management, or to filter the logs to avoid the difficulty of storing the large amounts of data).

Somewhat old, but still useful overviews of research problems in monitoring distributed systems can be found in [Joyce, Lomow et al. 1987] and [Mansouri-Samani and Sloman 1992]. Important research results in the field of distributed systems, which affect the problem of monitoring them, include:

- Seminal research related to the identification of the global state of a distributed system: work on logical clocks and vector clocks [Lamport 1978]; clock synchronization [Lamport 1987], [Cristian 1989]; global snapshots [Mani-Chandy and Lamport 1985] and a comparison between the concepts of group membership and system diagnosis [Hiltunen 1995]. These results are important in the field since the authors identified problems and solutions needed for the correct monitoring of distributed systems.
- Results about “Unreliable failure detectors” [Chandra and Toueg 1996], dealing with the problem of monitoring crash failures of components in distributed systems.
- Quality of Service (QoS) monitoring and management: challenges and approaches in providing QoS monitoring are described in detail in [Jiang, Tham et al. 2000]; the idea of a “dependability manager” (a special type of QoS monitor) was defined in [Porcarelli, Bondavalli et al. 2002] and specialized in [Porcarelli, Castaldi et al. 2004].
- The whole field of extracting information from an event log (possibly presented as a real-time data stream) is relevant to the needs of monitoring for dependability assessment. Among recent developments, the work on adaptable parsing of real-time data stream [Campanile, Cilaro et al. 2007] proposes methods and tools to support monitoring of a e.g., data stream under real-time constraints.
- Relationships between monitoring and metrology (see [van Moorsel 2009, Chap. "Metrology"]): recent work about monitoring distributed systems [Falai 2007] defines a conceptual framework for experimental evaluation and monitoring activities that supports rigorous (from a metrological point of view) observation of distributed system.

### **3. Automatic Failure Reporting for Software**

Commercial and open-source software increasingly employs automatic failure reporting features.

A discussion of these monitoring mechanisms [Murphy 2004] lists benefits for both the vendors and the customers; vendors are enabled to extend their quality assurance processes, while customers are offered improved user experience (the user’s perception of the product’s

quality, often seen as the most important quality measurement, can be continuously improved). This paper however warns that certain pre-requisites are necessary during development of a failure reporting system: i) an understanding of the customer base and usage profile, ii) an understanding of the failure profile, and iii) the users' privacy, frequently regarded as the most sensitive requirement, has to be respected.

For software development organisations, the main role of automatic failure reporting is usually detecting the bugs that manifest themselves most frequently (e.g., in [Murphy 2004] it is claimed that Windows XP exhibited very skewed failure set immediately following its release – “a very small number of bugs were responsible for the majority of customer failures”), thus triggering and prioritising the bug fixes. However, these data could also be used for assessing reliability of the software during use, if only data about the amount of use were also collected. Reliability will of course vary between users, depending on their “operational profiles” (frequencies of different kinds of demands on the software). So, collecting simple usage data, like total time in operation, or number or volume of demands on an installation, would be enough for retrospectively assessing the reliability of that installation, but extrapolation to reliability in future use requires linking failure reports to frequencies of demand types, or other characterisations of operational profiles. Extending failure reporting with automated collection of accurate usage data and operational profiles was proposed by Voas and colleagues [Voas 2000] to help vendors make better long-term decisions with respect to their customer requirements (benefits to vendors would include, for example, detecting misused and unused features or discovering the most typical machine configurations used to execute the vendor's software). A scheme for collecting operational profile data, related to the work in [Voas 2000], is one of the claims in the US patent 6862696 [Voas 2008].

The other important limitation of automated failure reporting systems as usually deployed is that they typically focus on detecting crash failures, as this is an easy requirement to satisfy, but ignore “value domain” failures (production of wrong results), which however should be considered at least as important. The costs of the two kinds of failures vary between applications, but the importance of non-crash failures should not be underestimated: for instance, bug reports for SQL servers show a surprisingly high prevalence of defects that cause non-crash failures [Gashi, Popov et al. 2007], [Vandiver, Balakrishnan et al. 2007].

[Garzia, Khambatt et al. 2007] present an approach for assessing end-user reliability. In essence, the approach is based on the following:

- Grouping users' computers in terms of the number of disruptions they experience per unit of time. Three groups are defined: *excellent*, *good* and *poor* reliability group. The grouping is performed in regard to the anticipated improvement of the Windows Vista over Windows XP SP2 operating system.
- Placing each user's machine in one of the three groups based on the frequency of failures on a particular build, over a period of two weeks (for this the data from the installations of beta users were used).
- Identifying bugs and assessing reliability were then employed for every major Windows Vista milestone.

The results of the assessment approach are used for ensuring the reliability objectives at major software release stages have been met. The authors focus on end-users of desktop and laptop computers; these users seek disruption-free operation of the software – both a software failure and a planned activity, e.g., installation of a software update, is regarded as detrimental. The case study was performed with Microsoft Windows Vista operating system and, according to the paper authors, the outcomes of the proposed approach have been used in deciding on the product's shipment strategy.

The insightful work of [Li, Ni et al. 2008] provides a classification of current approaches used in the IT industry for collection of reliability feedback data – four classes are identified: interactive user reporting, online user reporting, automated per-incident reporting and automated reliability monitoring. The classification, though possibly incomplete, together with seven proposed evaluation criteria, helps comparing different approaches. The authors recommend that automated reliability monitoring is used as the approach for reliability assessment of mass-market software, and they give details of Windows Reliability Analysis Component (RAC), an implementation of the recommended approach. RAC is available on Windows Vista operating system; it runs as a low-priority process; it collects the data using operating system event logs and system calls and periodically sends the information to Microsoft. The information includes details of all detected failures during an observation period, and it provides the basis for data normalisation by reporting successful executions and execution durations.

The limitations of the automated reliability monitoring approach can be identified reflecting on, for instance, the above cited article [Li, Ni et al. 2008]:

- The authors acknowledge that software producers could be interested in reliability feedback data for two reasons: evaluation of the reliability status of the product, and identification and reaction to the failures. Their viewpoint, however, focuses on the latter, whereby the collection of reliability feedback data is primarily undertaken for software improvement. This viewpoint might be inadequate in some cases, e.g., during the software procurement phase when only assessment results are needed. The stance of the software producers is, however, expected, for rarely is a software vendor willing to share assessment results (if they exist at all) with the party in charge of procurement.
- Crash failures are assumed (as is the case with most automatic failure reporting systems, see above) - the focus of the study is on application crashes, application hangs and operating systems crashes.
- Even if the fault-model is to include the faults leading to non-crash failures, the *correctness*, one of the seven evaluation criteria proposed in the paper, is defined with regard to a “perfect” oracle – it is based on the assumption that the software specifications are correct. Chances are, however, that due to developers’ misunderstandings of users’ needs, the users perceive the software to malfunction (observing a non-crash failure for example) even if it is running correctly according to the developers’ specification. This may lead to non-diagnosable failures.

We outline below two of the established software failure reporting systems.

The Mozilla Foundation has been providing automatic bug tracking through a proprietary system called *Talkback*, or *Quality Feedback Agent* (QFA), since the 1990s. The strategy was inherited from its parent company Netscape and had been successful in that, in the 2000–2004 period, the QFA program helped to resolve around 778 bugs [McLaughlin 2004]. Mozilla report [McLaughlin 2004] that in their experience a user base of 20,000 or more provide the best information for the purpose of uncovering software faults, building reproducible test cases and producing appropriate fixes; we may note that confidence in dependability estimates also increases with a larger number of monitored installations. Beginning with Firefox version 3 alpha 5, the Mozilla platform is equipped with an open-source crash reporting system, which is a combination of the following products:

- Google Breakpad [Google 2008a] client and server libraries.
- Mozilla-specific crash reporting user interface and bootstrap code.
- Socorro Collection and reporting server [Google 2008b].

Before the introduction of the abovementioned RAC [Li, Ni et al. 2008] feature in the Vista operating system, Microsoft™ offered their customers automated per-incident reporting through *Windows Error Reporting (WER)* [Microsoft 2008]. This framework provides for extraction of crash dump data, which are forwarded to Microsoft for further analysis. Whereas crash, hang, and kernel fault reporting is provided by default in the Windows Vista operating system, application-specific issues have to be communicated through WER – the client-side WER service forwards the memory dump information from a desktop application, which had crashed or stopped responding, to Microsoft. This action is initiated only after the user has given his/her consent. Reported problems trigger a mix of automated and human-mediated reactions like:

- A solution is sent to the user.
- More information is requested from the user so to aid developers in resolving the problem.
- A new “case” for the problem is opened, and the user is notified by a status message.

There are other software products that provide automated, opt-in systems for failure reporting, such as The Apple Crash Reporter [Apple 2006], the GNOME [GNOME 2008] and KDE [KDE 2008] desktop environments. Apart from these integrated solutions, there exist more comprehensive, stand-alone products which enable one to collect, quantify, and control large amounts of test and field data, such as FRACAS by Relex™ [Relex™ 2008].

Academic research has followed the pace of advance in automatic failure reporting. To take an example, *Cooperative Bug Isolation* [Liblit 2004] is a low-overhead instrumentation strategy for collecting information from a multitude of software end-users. The focus of the work is a *statistical debugging* approach. It provides a suite of algorithms for finding and fixing software errors based on statistical analysis of feedback data collected from the end-users.

Many interesting ideas about (automated) failure data analysis were presented at the “Reliability Analysis of System Failure Data” workshop held at Microsoft Cambridge, UK on 1-2 March 2007. For instance, one suggestion [Fetzer 2007] was that execution traces to be replayed at the developer’s site following a failure, be collected through a “hybrid” approach: instead of executing automatic tracing constantly and incurring (unnecessary) overhead during normal operations, one could combine it with the RX approach [Qin, Tucek et al. 2005] for software fault tolerance. The RX approach provides a light-weight checkpointing mechanism, which ensures that a program is rolled back to the most recent checkpoint after an

error is discovered. During the retry of the program, the environment is changed to maximize the probability that bugs are masked. A possibility, then, is to initiate collection of execution traces only during the retries, which would reduce the overhead during normal operation.

Online monitoring plays an integral role in the field of *online failure prediction*. This field's main concern is the assessment of the risk that "misbehaviors" result in failures in a near future. For this purpose, online failure prediction employs runtime monitoring and measurement of the current system state. This kind of monitoring enables detection of errors through observation of side-effects on the system, e.g., a memory leak can be discovered through exceptional circumstances like high CPU load or disk activity, or an atypical function call. A comprehensive survey of online failure prediction techniques is given in [Salfner, Lenk et al. In print].

## 4. Network Monitoring

*Network monitoring* is a part of network management functions. It includes the following: means for identification of problems caused by network failures and overloaded and/or failed devices, continuous measurement of QoS attributes to enable corrective management actions or to produce longer term dependability measurements, as well as means of alerting network administrators to virus or malware infections, questionable user activity and power outages. Network monitoring systems differ from intrusion detection systems (IDSs) or intrusion prevention systems (IPSs)<sup>3</sup> in that their focus is not exclusively on security, but also network functioning during ordinary operation. In addition to being used for availability assurance and performance improvement, network monitoring helps to accumulate information that could be used for planning future network growth.

Network monitoring systems typically use "network monitoring interface cards" (NMIC), pieces of hardware similar to standard NICs, but with a difference in design so that they passively listen on a network. NMICs have no MAC (Media Access Control) layer addresses and are invisible to other devices on the network. One of the most prominent manufactures is Endace Ltd. (<http://www.endace.com/dag-network-monitoring-cards.html>).

Network monitoring systems may listen to different application level protocols like FTP, SMTP, POP3, IMAP, DNS, SSH, TELNET. As for communicating data, many protocols can be used, but the standard is Simple Network Management Protocol (SNMP), defined by the

---

<sup>3</sup> Both IDSs and IPSs are discussed in the section "Intrusion Detection and Intrusion Prevention Systems" of this report.

Internet Engineering Task Force (IETF) in their Requests for Comments, RFC 1157 [IETF 1990] and RFC 3413 [IETF 2002]. The protocol prescribes an architecture in which a software component (denoted as *agent*) executes on managed systems (network elements such as hosts, gateways, terminal servers, and the like), collecting the desired network information, that is in turn communicated to one or more managing systems. It exists in three versions; SNMPv3 has been the standard version since 2004. The IETF has designated SNMPv3 a full Internet Standard, the highest maturity level for an RFC. An interesting extension of SNMP with accounting features is given in [Stancev and Gavrilovska 2001].

Closely related to SNMP is the RMON standard for remote network monitoring, which prescribes the ways of monitoring the internet traffic. It has been standardized by RFC 1271 in November 1991, but it is updated by RFC 1757 [IETF 1995] in February 1995. RMON2 [IETF 1997] is an extension of RMON that focuses on higher layers of traffic above the medium access-control (MAC) layer.

Network monitoring software products are either embedded in larger software suites or are offered as stand-alone products. A comprehensive repository of network monitoring tools can be found at [StanfordLinearAccelerationCentre 2008].

Numerous QoS monitoring tools have been built by industry. QoS monitoring is made more difficult by the fact that various pieces of network equipment route traffic through numerous firewalls and address translators. Cisco™, for instance, offers several tools for QoS monitoring [Cisco 2008]. Another example of QoS monitoring tool is XenMon [Gupta, Gardner et al. 2005], a performance monitoring tool based on Xen virtual environment, [Xen 2008]. XenMon provides support for resource allocation and management functions through accurate monitoring and performance profiling infrastructure.

## 5. Intrusion Detection and Intrusion Prevention Systems

Whereas network monitoring is primarily concerned with checking and reporting from “inside” a network, e.g., detailing measurements of link bandwidth, latency and response from routers and switches, and CPU utilisation time, *intrusion detection systems* (IDS) monitor a network with the aim of discovering threats coming from *outside*, to trigger appropriate defensive reactions.

IDSs are an important mechanism in the assessment of network security. They are used to collect data about attacks (see, for example, [Cukier, Berthier et al. 2006], [Leurre.com 2008], [HoneynetProject 2008]), an important part of security research. Detecting both attempted and

successful intrusions would of course allow measurements of security (measurements of both absolute frequency of successful intrusions and of the fraction of intrusions that are successful), but this potential is limited by the fact that the IDSs themselves are inevitably imperfect, and exhibit *false negatives* (failing to raise an alarm following a real intrusion), *false positives* (raising an alarm due to legitimate traffic) or *event misclassification* (raising an alarm following a real intrusion, but failing to classify it correctly).

IDSs are classified into different categories, which bring different imperfections in their suitability as measurement instruments, on the basis of various criteria, such as:

- The *location* of the monitoring sensors:
  - Network intrusion detection systems (NIDS) identify intrusions by examining network traffic, e.g., Snort [Snort.org 2008].
  - Host-based intrusion detection system (HIDS) identify intrusions by analyzing system calls, application logs, file-system modifications (binaries, password files, etc.) and other host activities and state, e.g., OSSEC [OSSEC 2008].
- The *monitoring focus*:
  - Protocol-based intrusion detection systems (PIDS) monitor and analyze the traffic of a protocol.
  - Application protocol-based intrusion detection system (APIDS) works similarly to PIDS, but the focus is an application layer protocol.
- The *detection method*:
  - Anomaly-based, e.g., Cfengine [Cfengine 2008], which can be configured to perform anomaly detection, i.e. to detect data patterns that differ from the normal, established behavioural patterns.
  - Signature-based, network traffic or host activity is analysed so that predetermined attack patterns, known as signatures, are discovered.

Expectedly, there exist hybrid intrusion detection systems, which combine two or more approaches, e.g., *Prelude* [Prelude 2008] or *Emerald* [SRI 2008].

An alternative classification groups IDSs into *passive* and *reactive* [Tipton and Krause 2003]. The latter are also known as Intrusion Prevention Systems (IPS). In addition to standard

detection, logging and signalling of intrusions, they disable the connections initiated by the suspected malicious sources.

A list of intrusion detection systems is given in [TimberlineTech 2008], and a classification in a tree-like structure can be found at [IPSec 2008]. Comprehensive guidance on intrusion detection is given in the NIST's standard [Scarfone and Mell 2007]. A solution for online flow-level intrusion detection for high-speed network, denoted as HiFIND, is given in [Gao, Li et al. 2006]. HiFIND's major characteristics are: it is scalable with respect to flow-level detection on high-speed networks; it is resilient to DoS (Denial of Service) attacks; it can distinguish SYN flooding<sup>4</sup> and various port scans for effective mitigation; it enables aggregate detection over multiple routers/gateways; and separates anomalies to limit false positives. The approach for resilience improvement of intrusion detection systems in [Sousa, Bessani et al. 2007] combines proactive and reactive recovery. The complementary approach maintains system's correct operation by ensuring the availability of minimum amount of replicas.

Yet another way of detecting intrusions is given in [Strunk, Goodson et al. 2002]. The authors propose a new technology for storage devices to safeguard data, even when the host operating system is compromised. The technology enhances the intrusion survivability through:

- Faster intrusion detection through monitoring of (suspicious) storage activity.
- Facilitating intrusion diagnosis by providing a wealth of data to administrators.
- Simplifying and speeding up post-intrusion recovery by: preserving all pre-intrusion states, providing low granularity restoration (any number of files can be restored) and enabling users to continue utilising the system (non-destructive recovery).

## 6. Web Services Monitoring

Web services technology represents a significant support element for the growing integration of ICT systems across organisational boundaries. The World Wide Web Consortium (W3C) adopts the following broad definition: "*A Web service is a software system designed to support interoperable machine-to-machine interaction over a network*", [W3C 2008].

Web services have generated increased interest in QoS monitoring. Web Service Level Agreements (WSLA) are contractual agreements between two parties, either customers and

---

<sup>4</sup> A SYN flooding is a form of DoS attack in which an attacker keeps sending SYN (synchronise) messages to a target's system, until the target's system resources are exhausted.

service provider(s) or service providers only, which formally specify the Quality of Service (QoS) level to be provided, as well as the consequences resulting from its violation. Contractual obligations are commonly stipulated between provider and consumer of a service, as well as the financial consequences in case the obligations are violated. A language specification for WSLAs is given in [Ludwig, Keller et al. 2003]. WSLA monitoring and enforcement is increasingly important; this is particularly true for environments where dynamic or on-demand subscriptions to web services are possible. The monitoring is important for customers, who need the reliability and performance agreements to be maintained at runtime, and indeed to service providers who want to control the agreed level of usage. The unpredictable Internet environment implies that reliability guarantees of the applications using web services become difficult to ensure: this increases the importance of web service reliability assessment techniques. Research has been active in the web service monitoring field. Monitoring web services should include measuring at multiple sites; both client- and server-side measurements should be taken. To this end, the authors of [Sahai, Machiraju et al. 2002] propose an automated and distributed SLA monitoring engine, the Web Service Management Agent (WSMN), which provides for joining and correlating web service and business process data. WSMN is based on the proposed specification language that allows for definition of accurate and flexible WSLAs. A framework for specifying and monitoring WSLAs is given in [Alexander and Heiko 2003]<sup>5</sup>. It integrates flexible and extensible formal language (based on XML) for expressing service level agreements and a runtime architecture comprising several SLA monitoring services. The framework is flexible in that it can be applied to not only SLAs of web services, but to other inter-domain management scenarios, such as management of networks, systems or applications. In [Raimondi, Skene et al. 2008] WSLA contractual conditions are represented as timed automata and a methodology, with a prototype implementation using an Eclipse<sup>6</sup> plug-in, for WSLA online monitoring is proposed.

A possible architecture, and a corresponding implementation, for performance management of web services is given in [Levy, Nagarajarao et al. 2003]. The feedback controller of the management system, which tunes the performance parameters (e.g., load balancing or request queuing), is based on a continuous monitoring approach, wherein system components (servers, gateways, global resource manager, and console) share monitoring information

---

<sup>5</sup> Further information can be found on the webpage <http://www.research.ibm.com/wsla/>.

<sup>6</sup> Eclipse, <http://www.eclipse.org/>, is at the time of writing recognised as one of the most popular software development platforms.

through a publish/subscribe network (e.g., the global resource manager computes the maximum number of concurrent requests that a server should execute, based on the performance information *published* by each gateway).

Besides the contractual aspect of QoS monitoring, there is a need for users to assess or rank the dependability of web services that they might use. Chen et al. propose WSsDAT tool for dependability assessment of Web Services from the client perspective by collecting metadata representing a number of dependability metrics [Chen, Li et al. 2006]. One of the purposes is to identify good replication strategies, which employ the same or similar web services to enhance reliability. WS-DREAM [Zheng and Lyu 2008] addresses these issues by allowing users, potentially geographically dispersed, to carry out web service reliability assessment in a collaborative manner. This worldwide testing, and sharing of test case results, is coordinated by a centralised server; reliability assessment becomes affordable to many participants through data sharing. What both WS-DREAM and WSsDAT seem to be missing in assessing dependability levels is the account of the impact of operational profile from each individual client.

## 7. Embedded Systems Telemetry

The use of *embedded systems* has been continuously growing for decades. They have been integrated in portable devices such as mobile phones as well as in automotive safety systems, e.g., anti-lock braking system (ABS), but also in nuclear power plant control and safety systems.

The trend has required advances in monitoring of these systems for supporting hardware and software maintenance, as well as dependability and performance assessment. As manufacturers tend to embed “telemetry” capabilities to monitor the performance and health of the hardware which the computers control (from engines in aircraft, trucks and cars to factory equipment and house appliances) and support maintenance and repair, so they can add similar capabilities to monitor the performance and health of the embedded computers. Like other monitoring practices, these can be used to evaluate the dependability and resilience of these systems.

A sort of online monitoring is frequently employed for error detection, e.g., as in “watchdog” technique, where a small coprocessor is used to monitor the behaviour of the main processor. An insightful survey on this technique is given in [Mahmood and McCluskey 1988]. This

technique is not exclusive to embedded systems but is also used for dependability assessment of general-purpose computers.

Monitoring embedded systems as a part of runtime verification process requires non-intrusive arrangements, e.g., with hardware resources dedicated to the monitored system. These concerns are addressed in [Watterson and Heffernan 2007]. A number of existing runtime verification tools are referenced, highlighting their requirements for monitoring solutions (see the section “Runtime Verification” of this report for more details on online monitoring for runtime verification). The authors present a review of established and emerging approaches for the monitoring of embedded software execution.

A few examples may illustrate current practice in telemetry of embedded systems. Micro Digital Inc™ (founded in 1975), one of the first companies to offer embedded-systems software, recommends the use of Web Network Management Protocol (WNMP)<sup>7</sup>. They claim the following features:

*“WNMP (Web Network Management Protocol) enables simple and flexible administration and monitoring of embedded systems using ordinary web browsers. In contrast to SNMP, no special program running on the client system is needed. Nor is a MIB (Management Information Base)<sup>8</sup> or MIB tool required. This makes it easy to administer or monitor an embedded system from any computer, even a modern cell phone”* [MicroDigital 2008].

The software tool  $\mu C/Probe$ , from *Micrium* [Micrium 2008], enables developers to monitor embedded systems in a live environment. Execution traces are obtained without the need for halting the application, unlike conventional debuggers, and thus the benefits extend beyond the development process. This type of monitoring application is particularly valuable for embedded systems that cannot be suspended to examine variable and program flow, e.g. compressors used to pump natural gas through a pipeline, or similar reciprocating engines (like car pistons). Bringing an engine online is a very complex task and the variables have to be examined at runtime to validate the working of the control system. On the other hand, such an approach carries an overhead (with respect to performance and resource usage) due to additional computation needed for communication between the  $\mu C/Probe$  and the target

---

<sup>7</sup> <http://barracudaserver.com/brochures/WNMP.html>

<sup>8</sup> “A management information base (MIB) stems from the OSI/ISO Network management model and is a type of database used to manage the devices in a communications network. It comprises a collection of objects in a (virtual) database used to manage entities (such as routers and switches) in a network.”,  
[http://en.wikipedia.org/wiki/Management\\_information\\_base](http://en.wikipedia.org/wiki/Management_information_base)

embedded application. Such an overhead is a characteristic of other monitoring-related features, which generally can be provided at additional cost if spare resources are available.

*Advanced Telemetry Linked Acquisition System (ATLAS)* [McLarenElectronicSystems 2008] is a software package from McLaren Electronics used for obtaining, displaying and analysing data from automotive control systems. It has capabilities of comparing live telemetry data with uploaded logs. Also, it provides fast data handling so that the large quantities of data could be processed in real-time.

The research led by professor Brian Williams from Computer Science and Artificial Intelligence Laboratory (CSAIL), at Massachusetts Institute of Technology, has led to a significant progress in the field of *model-based autonomy* [Williams and Nayak 1996], [Williams, Ingham et al. 2003]. This research advocates the use of the model-based programming paradigm for addressing intrinsically difficult task of programming complex embedded systems, which necessitates reasoning about complex interactions between sensors, actuators, and control processors. Statistical models, such as hidden Markov model, are constituent parts of model-based programming. This type of programming has led to the creation of *Livingstone*, a core component in the flight software of Deep Space One, the first spacecraft in NASA's New Millennium programme. Livingstone is a reactive, model-based and self-configuring kernel for autonomous systems. One of the main active research areas in the field is concerned with intelligent embedded systems, which are able to explore, diagnose and repair themselves using online assessment combined with adaptation. The online assessment approach relies on the advanced monitoring capabilities.

## 8. Monitoring Large-Scale Enterprise Software

Business integration across enterprise domains requires connection of data and applications throughout the enterprise. In this section, we discuss monitoring techniques for large, enterprise-wide, systems of networks and computers. Nowadays, enterprise software systems represent the main support for many businesses to provide services to the end users. The software is required to provide continuous service despite failures of individual hardware components or software processes.

Satisfying stringent dependability requirements is, however, hard – enterprise software is growing in size and complexity, and the necessary updates are becoming more frequent [Munawar and Ward 2006]. Each component in a system requires monitoring of its resources, performance and state variables. This frequently leads to generating an overwhelming amount

of data, which are hard to collect and store, and are complicated to analyse. State-of-the-art monitoring methods, therefore, focus on continuously examining only partial data, intensifying monitoring level only when a particular problem is suspected [Munawar and Ward 2006]. This improves performance by decreasing measurement effects and at the same time reduces storage, transmission, analysis and reporting overhead.

There is a variety of commercial solutions for enterprise software monitoring, such as:

- The IBM's MQ WebSphere [Stanford-Clark 2002] family of products (formerly know as MQ Series) offers a solution. The products transform field data into whatever form is required by different applications and a publish/subscribe model enables the receipt of the exact subset of data required. MQIsdp (MQ Integrator, SCADA device protocol) offers a publish/subscribe model over TCP/IP with different levels of delivery assurance: at most once, at least once and once-and-once-only.
- Windows Management Instrumentation (WMI) [Microsoft 2000] defines a non-proprietary set of environment-independent specifications which allow management information to be shared between management applications. WMI prescribes enterprise management standards and related technologies that work with existing management standards, such as Desktop Management Interface (DMI) and SNMP (see section "Network Monitoring" of this report). Distributed Management Task Force (DMTF 2008) (DMTF, formerly "Desktop Management Task Force") is a standards organization that develops and maintains standards for management of enterprise IT and Internet environments. DMTF standards, among others, include the following:
  - *Common Information Model (CIM)* - The CIM schema is a conceptual schema that defines how the managed elements in an IT environment (for instance computers or storage area networks) are represented as a common set of objects and relationships between them. CIM is extensible to allow product specific extensions to the common definition of these managed elements. CIM is the basis for most of the other DMTF standards.
  - *Web-Based Enterprise Management (WBEM)*, a set of systems management technologies, which specifies the following: protocols for the interaction between systems management infrastructure components implementing CIM, a concept for defining the behaviour of the elements in the CIM schema, the

CIM Query Language (CQL) and other specifications needed for the interoperability of CIM infrastructure.

Many products from leading software vendors are WMI enabled, such as:

- HP OpenView, a former Hewlett Packard product range consisting of network and systems management products. In 2007, HP OpenView was renamed to HP Software [HewlettPackard 2008]. HP OpenView is most commonly described as a suite of software applications which allow large-scale system and network management of an organization's IT infrastructure.
- IBM Tivoli Management Framework (TMF) [IBM 2008b]. TMF is a systems management platform from IBM – the framework is a CORBA-based architecture that provides management of large numbers of remote locations or devices.

IBM's *autonomic computing* initiative [IBM 2008a], of which IBM Tivoli is one technology, has addressed the growing complexity of computing systems management through self-managing autonomic system paradigm. The approach, whose concept originates from social animal collective behaviour, defines four functional areas: self-configuration, self-healing, self-optimisation, and self-protection. Online monitoring is pertinent to self-healing and self-protection, but also to self-optimisation, since the area is concerned with control of resources through automatic monitoring to ensure the optimal functioning with respect to the defined requirements. Academic research has provided significant contributions to the autonomic computing programme, and at the same time to monitoring of enterprise level software. Some of the notable examples are:

- Astrolabe [van Renesse, Birman et al. 2003], a DNS-like distributed management service for scalable management and self-organization of large-scale, highly dynamic, distributed applications.
- A mechanism for dynamically monitoring and reconfiguring a system-of-systems built out of a heterogeneous mix of components [Kaiser, Gross et al. 2002].

Research in the field of dependability of enterprise systems still suffers from scarcity of empirical data, with few published empirical studies [Sahoo, Squillante et al. 2004] [Hsueh 2008]. The earlier paper [Sahoo, Squillante et al. 2004] emphasised the need for a deeper understanding about the occurrence of failures and their statistical properties in real (enterprise) environments. The authors presented a detailed empirical and statistical analysis of system errors and failures from a production environment of 400 servers, pointing out that

the collection of empirical data is more difficult with the increased complexity of today's software. The latter paper [Hsueh 2008] presented a characterisation of failure data collected by Boeing's *internal incident and problem recording system*. The characterisation focuses on only service availability related records obtained from this online monitoring system.

## 9. Runtime Verification

*Runtime verification* techniques are means for runtime failure reporting and they typically rely on “formal methods” for designing the functions that detect failures (and are thus not limited to crash failures). These techniques are motivated more with verifying and improving software than by measuring its dependability attributes. Well-established, heavyweight formal methods, e.g., theorem proving or model-checking, are being increasingly accompanied by online (runtime) monitoring with the aim of helping in verifying program correctness. This is, at least partly, due to the constant increase of software complexity and size (frequently resulting in state space explosion), which limits the use of traditional verification techniques. Moreover, runtime error detection solutions are explicitly required by some development standards of safety critical industries: e.g., EN 50128 [BSI-BritishStandards 2001] for computerized railway control systems prescribes several methods and techniques that are mandatory or highly recommended for achieving specific *safety integrity levels* (SIL).

Monitoring for software failures during software runtime confirms (or disproves) whether an actual run of the software conforms to the requirements specification, by checking whether it preserves certain formally specified properties. Note that runtime error detection is closer to testing than to exhaustive model checking since it cannot prove the *lack of defects*, but is only able to indicate the *presence of defects* in the execution. Runtime error detection techniques are used for detecting defects that have remained in the software even after rigorous testing. Error detection signals emitted by these solutions can trigger error confinement and recovery activities, i.e., runtime verification can be seen as an entry point of fault tolerance mechanism. The actual implementation of runtime verification techniques can be built on such increasingly popular approaches as *aspect-oriented* (AOP [Kiczales, Lamping et al. 1997]) or *monitoring-oriented* programming (MOP [Formal-Systems-Laboratory 2008]).

The seminal paper [Diaz, Juanole et al. 1994], which extends the ideas from [Ayache, Azema et al. 1979], proposes the *Observer* concept for validating runtime behaviours of self-checking distributed systems – “*systems that detect erroneous behaviours as soon as errors act at some observable output level*”. The concept is based on the principle of an observer-worker system.

A system of this type consists of two distinct components, a *worker* and an *observer*: the former is a standard implementation of the system behaviour; while the latter is a redundant implementation which outputs are comparable with the outputs of the worker. Using the concept, it is possible that a system's runtime behaviour is (continuously) checked against a formal specification.

StateRover tool [TimeRover 2009] provides, among other features, support for validation, automatic test generation and verification of system requirements. The tool is based on an automatic model checking technique for UML (Unified Modeling Language) statecharts. The self-acting attribute of the technique stems from the use of an automatic white box test-generation combined with automatic run-time monitoring of statechart assertions. The white-box test generator automatically produces test sequences (events, conditions, and input data); chooses one of the test sequences and submits it to the System Under Test (SUT), which subsequently starts up an embedded assertion for run-time monitoring.

A taxonomy of runtime software-fault monitoring tools is described in [Delgado, Quiroz Gates et al. 2004]. It presents common building blocks of a monitoring system: specification language, monitor and event-handler (the results of monitoring are captured and communicated to the user by an event-handler, and in some cases responses to violations are initiated). Operational issues, such as program types targeted by the monitoring system, platform dependencies and tool maturity level, are also included in the taxonomy.

MOP [Formal-Systems-Laboratory 2008] is a tool-supported formal software development framework in which runtime monitoring presents a basic design principle. MOP enables a developer to specify required properties by using a formalism of choice. Many formalisms are allowed, due to the observation that different types of logic are best suited to different types of specifications. Depending on the success of the properties' validation (a property can be violated or successfully validated), a specific code, stated by the programmer, will be executed.

Another interesting work, which builds on the extensively studied field of self-checking via assertions, is the *RunTime Reflection Project* [Arafat, Bauer et al. 2005]. The project establishes methods to dynamically analyze reactive distributed systems at runtime. The approach is layered and modular; it provides the means for both detecting system failures (through runtime verification) and identifying the failure causes (through a detailed diagnosis). Diagnoses can be subsequently used in order to trigger recovery measures, or to

store detailed log-files for off-line analysis. The approach has many points in common with the proposal of the Dependability Manager [Porcarelli, Bondavalli et al. 2002]; the main difference is that runtime verification has as its main purpose *verification* of the system, while the main purpose of the Dependability Manager is the *reconfiguration* of the system based on runtime evaluations.

There is a significant body of research dedicated to the automatic synthesis of executable assertions (i.e., the implementation of checks to be performed during runtime). A significant part of the work appears in the Runtime Verification Workshop series [RVW 2008], such as [Rosu and Havelund 2001], [Finkbeiner and Sipma 2004], [Gerth, Peled et al. 1995].

## References

[Alexander, K. and L. Heiko 2003] "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services" Journal of Network and Systems Management 11(1): 57-81.

[Apple 2006]" Apple Crash Reporter" accessed Jan. 2009, from <http://developer.apple.com/technotes/tn2004/tn2123.html>

[Arafat, O., A. Bauer, et al. 2005] "Runtime Verification Revisited". München, Technischen Universität München.

[Ayache, J. M., P. Azema, et al. 1979] "Observer, a Concept for On-Line Detection for Control Errors in Concurrent Systems". Ninth International Symposium on Fault-Tolerant Computing, Madison.

[BSI-BritishStandards 2001] "BS EN 50128".

[Campanile, F., A. Cilaro, et al. 2007] "Adaptable Parsing of Real-Time Data Streams". Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, IEEE Computer Society.

[Cfengine 2008]" Cfengine - a Datacentre Management Platform" accessed Dec. 2008, from <http://www.cfengine.com/>

[Chandler, W. W. 1983] "The Installation and Maintenance of Colossus" Annals of the History of Computing, IEEE 5(3): 260-262.

[Chandra, T. D. and S. Toueg 1996] "Unreliable Failure Detectors for Reliable Distributed Systems" Journal of the ACM (JACM) 43(2): 225-267.

[Chen, Y., P. Li, et al. 2006] "Measuring the Dependability of Web Services for Use in e-Science Experiments". Third International Service Availability Symposium, ISAS 2006, Helsinki, Finland, Springer.

[Cisco 2008]" QoS Monitoring Tools" accessed Dec. 2008, from [http://www.cisco.com/en/US/tech/tk543/tk759/technologies\\_tech\\_note09186a0080094bc3.shtml](http://www.cisco.com/en/US/tech/tk543/tk759/technologies_tech_note09186a0080094bc3.shtml)

[Cristian, F. 1989] "Probabilistic Clock Synchronization" Distributed Computing 3: 146-158.

[Cukier, M., R. Berthier, et al. 2006] "A Statistical Analysis of Attack Data to Separate Attacks". Proceedings of the International Conference on Dependable Systems and Networks (DSN06).

- [Delgado, N., A. Quiroz Gates, et al. 2004] "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools" IEEE Transactions On Software Engineering 30(12).
- [Diaz, M., G. Juanole, et al. 1994] "Observer-A Concept for Formal On-Line Validation of Distributed Systems" IEEE Transactions on Software Engineering 20(12): 900-913.
- [Falai, L. 2007] "Observing, Monitoring and Evaluation Distributed Systems". "Resilient Computing Lab", University of Florence. PhD.
- [Fetzer, C. 2007] "Automatic Collection of Failure Traces". "Reliability Analysis of System Failure Data workshop, RAF07". Cambridge, UK.
- [Finkbeiner, B. and H. Sipma 2004] "Checking Finite Traces Using Alternating Automata" Formal Methods in System Design 24(2): 101-127.
- [Formal-Systems-Laboratory 2008]" Monitoring-Oriented Programming" accessed Oct. 2008, from [http://fsl.cs.uiuc.edu/index.php/Monitoring-Oriented\\_Programming](http://fsl.cs.uiuc.edu/index.php/Monitoring-Oriented_Programming)
- [Gao, Y., Z. Li, et al. 2006] "A DoS Resilient Flow-level Intrusion Detection Approach for High-speed Networks". Proceedings of 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06).
- [Garzia, M., M. Khambatt, et al. 2007] "Assessing End-User Reliability Prior To Product Ship". "Reliability Analysis of System Failure Data". Microsoft Research, Cambridge, UK, March 2007.
- [Gashi, I., P. Popov, et al. 2007] "Fault Tolerance via Diversity for Off-the-Shelf Products: A Study with SQL Database Servers" IEEE Transactions on Dependable and Secure Computing (TDSC) 4(4): 280-294.
- [Gerth, R., D. Peled, et al. 1995] "Simple on-the-fly Automatic Verification of Linear Temporal Logic". Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification.
- [GNOME 2008]" FreeBSD GNOME Project: Reporting a Bug" accessed Dec. 2008, from <http://www.freebsd.org/gnome/docs/bugging.html>
- [Google 2008a]" Google BreakPad" accessed Dec. 2008, from <http://code.google.com/p/google-breakpad/>
- [Google 2008b]" Socorro" accessed Dec. 2008, from <http://code.google.com/p/socorro/>
- [Gray, J. and D. Siewiorek 1991] "High-Availability Computer Systems" Computer 24(9): 19.
- [Gupta, D., R. Gardner, et al. 2005] "XenMon: QoS Monitoring and Performance Profiling Tool", HP Research Labs.
- [HewlettPackard 2008]" HP Software" accessed Dec. 2008, from <http://welcome.hp.com/country/us/en/prodserv/software.html>
- [Hiltunen , M. A. 1995] "Membership and System Diagnosis". Proceedings of the 14th Symposium on Reliable Distributed Systems, IEEE Computer Society.
- [HoneyNetProject 2008]" The HoneyNet Project" accessed Dec. 2008, from <http://www.honeynet.org/>

- [Hsueh, M. C. 2008] "Failure Characterization of a Large Enterprise Computing Environment". Int. Workshop on Resilience Assessment and Dependability Benchmarking (RADB 2008), in DSN2008, Anchorage, Alaska, IEEE Computer Society.
- [IBM 2008a]" Autonomic Computing" accessed Dec. 2008, from <http://www.research.ibm.com/autonomic/index.html>
- [IBM 2008b]" Tivoli Software" accessed Dec. 2008, from <http://www-01.ibm.com/software/tivoli/products/mgt-framework/>
- [IETF 1990]" Request for Comments (RFC) 1157" accessed Dec. 2008, from <http://tools.ietf.org/html/rfc1157>
- [IETF 1995]" Request for Comments (RFC) 1757" <http://tools.ietf.org/html/rfc1757>
- [IETF 1997]" Request for Comments (RFC) 2021" <http://tools.ietf.org/html/rfc2021>
- [IETF 2002]" Request for Comments (RFC) 3413" accessed Dec. 2008, from <http://www.rfc-editor.org/rfc/rfc3413.txt>
- [IPSec 2008]" Intrusion Detection and Prevention Tree" accessed Jun. 2009, from <http://ipsec.pl/intrusion-detection/prevention-systems-classification-tree.html>
- [Iyer, R. K. 1995]. Experimental Evaluation. Proceedings of the 25th International Symposium on Fault-Tolerant Computing, FTCS-25, Pasadena, California.
- [Jiang, Y., C. K. Tham, et al. 2000] "Challenges and Approaches in Providing QoS Monitoring" International Journal of Network Management 10(6): 323-334.
- [Joyce, J., G. Lomow, et al. 1987] "Monitoring Distributed Systems" ACM Transactions on Computer Systems (TOCS) 5(2): 121-150.
- [Kaiser, G., P. Gross, et al. 2002] "An Approach to Autonomizing Legacy Systems". "Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN)". New York City, NY, USA.
- [KDE 2008]" KDE Bug Tracking System" accessed Dec. 2008, from <https://bugs.kde.org/>
- [Kiczales, G., J. Lamping, et al. 1997] "Aspect-Oriented Programming". Proceedings of the European Conference on Object-Oriented Programming.
- [Lamport, L. 1978] "Time, clocks, and the ordering of events in a distributed system" Communications of the ACM 21(7): 558-565.
- [Lamport, L. 1987] "Synchronizing Time Servers", Digital Equipment Corporation.
- [Leurre.com 2008]" The Leurre.com Honeynet Project" accessed Dec. 2008, from <http://www.leurrecom.org/>
- [Levy, R., J. Nagarajarao, et al. 2003] "Performance Management for Cluster Based Web Services". IFIP/IEEE Eighth International Symposium on Integrated Network Management, 2003.
- [Li, P. L., M. Ni, et al. 2008] "Reliability Assessment of Mass-Market Software: Insights from Windows Vista®". 19th International Symposium on Software Reliability Engineering, ISSRE 2008, Seattle, WA, USA.

[Liblit, B. R. 2004] "Cooperative Bug Isolation". "School of Computer Science", University of California Berkeley. PhD.

[Ludwig, H., A. Keller, et al. 2003] "Web Service Level Agreement (WSLA) - Language Specification", IBM T.J. Watson Research Center.

[Mani-Chandy, K. and L. Lamport 1985] "Distributed Snapshots: Determining Global States of Distributed Systems" ACM Transactions on Computer Systems (TOCS) 3(1): 63-75.

[Mansouri-Samani, M. and M. Sloman 1992] "Monitoring Distributed Systems (A Survey)". "Imperial College Research". London, Imperial College, London: 49.

[McLarenElectronicSystems 2008]" Atlas- Advanced Telemetry Linked Acquisition System" accessed Dec. 2008, from [http://www.mclarenelectronics.com/Products/All/sw\\_atlas.asp](http://www.mclarenelectronics.com/Products/All/sw_atlas.asp)

[McLaughlin, L. 2004] "Automated Bug Tracking: The Promise and the Pitfalls" IEEE Software 21(1): 100-103.

[Micrium 2008]" Micrium - Empowering Embedded Systems" accessed Dec. 2008, from <http://www.micrium.com/>

[MicroDigital 2008]" Web Network Management Protocol" accessed Dec. 2008, from [http://www.smxrtos.com/pr/barracuda\\_wnmp.htm](http://www.smxrtos.com/pr/barracuda_wnmp.htm)

[Microsoft 2000]" Windows Management Instrumentation: Background and Overview" accessed Oct. 2008, from <http://msdn.microsoft.com/en-us/library/ms811553.aspx>

[Microsoft 2008]" Introducing Windows Error Reporting" accessed Dec. 2008, from <http://msdn.microsoft.com/en-us/isv/bb190483.aspx>

[Munawar, M. and P. Ward 2006] "Adaptive Monitoring in Enterprise Software Systems". In SIGMETRICS 2006 Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML).

[Murphy, B. 2004] "Automating Software Failure Reporting" ACM Queue 2(8).

[OSSEC 2008]" OSSEC - a Host-Based Intrusion Detection System" accessed Dec. 2008, from <http://www.ossec.net/>

[Porcarelli, S., A. Bondavalli, et al. 2002] "Model-Based Dynamic Reconfiguration in Complex Critical Systems". Fourth European Dependable Computing Conference, (EDCC-4), Fast Abstract Track, Toulouse, France.

[Porcarelli, S., M. Castaldi, et al. 2004] "A Framework for Reconfiguration-based Fault-Tolerance in Distributed Systems ". Architecting Dependable Systems II. R. De Lemos, C. Gacek and A. Romanovsky, Springer-Verlag.

[Prelude 2008]" Prelude - a Universal Security Information Management (SIM) System" accessed Dec. 2008, from <http://www.prelude-ids.com/en/welcome/index.html>

[Qin, F., J. Tucek, et al. 2005] "RX: Treating Bugs as Allergies - a Safe Method to Survive Software Failures". Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05), New York, NY, USA, ACM Press.

[Raimondi, F., J. Skene, et al. 2008] "Efficient Online Monitoring of Web-Service SLAs". Proceeding of 16th ACM SIGSOFT International Symposium on the Foundations of Software Engineering, FSE 16, Atlanta, Georgia, USA.

[Relex™ 2008]" FRACAS" <http://www.relex.com/>

[Rosu, G. and K. Havelund 2001] "Monitoring Java Programs with Java PathExplorer". Proceedings of the First Workshop on Runtime Verification (RV'01), Paris, France Elsevier.

[RVW 2008]" Runtime Verification Workshop" <http://www.runtime-verification.org/>

[Sahai, A., V. Machiraju, et al. 2002] "Automated SLA Monitoring for Web Services". Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Management Technologies for E-Commerce and E-Business Applications, Springer-Verlag.

[Sahoo, R. K., M. S. Squillante, et al. 2004] "Failure Data Analysis of a Large-Scale Heterogeneous Server Environment". Proceedings of the Int. Conference on Dependable Systems and Networks (DSN'04), Florence, Italy, IEEE Computer Society.

[Salfner, F., M. Lenk, et al. In print] "A Survey of Online Failure Prediction Methods" ACM Computing Surveys

[Scarfone, K. and P. Mell 2007] "Guide to Intrusion Detection and Prevention Systems (IDPS)", National Institute of Standards and Technology (NIST).

[Schroeder, B. A. 1995] "On-line Monitoring: a Tutorial" Computer 28(6): 72-78.

[Siewiorek, D. and R. Swarz 1998] "General-purpose Computing". Reliable Computer Systems: Design and Evaluation. Natick, Massachusetts, A K Peters.

[Smith, D. J. 2001] "Reliability, Maintainability And Risk", Newnes (Elsevier Inc).

[Snort.org 2008]" SNORT® Network Intrusion Prevention and Detection System" accessed Dec. 2008, from <http://www.snort.org/>

[Sousa, P., A. N. Bessani, et al. 2007] "Resilient Intrusion Tolerance through Proactive and Reactive Recovery". Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007), IEEE Computer Society.

[SRI 2008]" Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD)" accessed Dec. 2008, from <http://www.sdl.sri.com/projects/emerald/>

[Stancev, B. and L. Gavrilovska 2001] "Implementation of Accounting Model within SNMPv3 Architecture". Proceedings of the Ninth IEEE International Conference on Networks, ICON 2001.

[Stanford-Clark, A. 2002] "Integrating Monitoring and Telemetry Devices as Part of Enterprise Information Resources", WebSphere MQ Development, IBM Software Group: 14.

[StanfordLinearAccelerationCentre 2008]" Network Monitoring Tools" <http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html>

[Strunk, J. D., G. R. Goodson, et al. 2002] "Intrusion Detection, Diagnosis, and Recovery with Self-Securing Storage". Pittsburgh, School of Computer Science, Carnegie Mellon University: 30.

[TimberlineTech 2008]" Alphabetical List of Intrusion Detection Products" accessed Dec. 2008, from <http://www.timberlinetechnologies.com/products/intrusiondtct.html>

- [TimeRover 2009]" StateRover Programming Environment" accessed June 2009, from <http://www.time-rover.com/staterover.pdf>
- [Tipton, H. F. and M. Krause 2003] "Information Security Management Handbook", CRC Press.
- [van Moorsel, A. 2009] "State of the Art on Assessing, Measuring and Benchmarking Resilience", AMBER Coordination Action: 137.
- [van Renesse, R., K. P. Birman, et al. 2003] "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining" ACM Transactions on Computer Systems (TOCS) 21(2): 164-206.
- [Vandiver, B., H. Balakrishnan, et al. 2007] "Tolerating Byzantine Faults in Transaction Processing Systems Using Commit Barrier Scheduling". Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles, Stevenson, Washington, USA, ACM (NY, USA).
- [Voas, J. 2000] "Deriving Accurate Operational Profiles for Mass-Marketed Software". Proceedings of the 4th International Conference on Empirical Assessment & Evaluation in Software (EASE 2000). Keele University, Staffordshire, UK, Springer.
- [Voas, J. 2008]" A Schema for Collection of Operational Profile Data, US Patent 6862696" accessed October 2008, from <http://www.patentstorm.us/patents/6862696/description.html>
- [W3C 2008]" Web Services Glossary" accessed Oct. 2008, from <http://www.w3.org/TR/ws-gloss/>
- [Watterson, C. and D. Heffernan 2007] "Runtime Verification and Monitoring of Embedded Systems" IET Software 1(5): 172-179.
- [Williams, B. C., M. D. Ingham, et al. 2003] "Model-based Programming of Intelligent Embedded Systems and Robotic Space Explorers" Proceedings of the IEEE 91(1): 212 - 237.
- [Williams, B. C. and U. Nayak 1996] "A Model-based Approach to Reactive Self-configuring Systems". Proceedings of the Thirteenth National Conference on Artificial Intelligence AAAI-96, Portland, Oregon.
- [Xen 2008]" The Xen® Hypervisor" accessed Dec. 2008, from <http://www.xen.org/>
- [Zheng, Z. and M. R. Lyu 2008] "WS-DREAM: A Distributed Reliability Assessment Mechanism for Web Services". Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2008, Anchorage, Alaska, USA.