Software Faults Diagnosis in Complex OTS Based Safety Critical Systems

G. Carrozza[1], D. Cotroneo[1], S.Russo[1]
(1) Dipartimento di Informatica e Sistemistica - Università degli Studi di Napoli Federico II
Via Claudio 21, 80125 - Naples, Italy
{*ga.carrozza, cotroneo, sterusso*}*@unina.it*

Abstract

This work addresses the problem of software fault diagnosis in complex safety critical software systems. The transient manifestations of software faults represent a challenging issue since they hamper a complete knowledge of the system fault model at design/development time. By taking into account existing diagnosis techniques, the paper proposes a novel diagnosis approach, which combines the detection and location processes. More specifically, detection and location modules have been designed to deal with partial knowledge about the system fault model. To this aim, they are tuned during system execution in order to improve diagnosis during system lifetime. A diagnosis engine has been realized to diagnose software faults in a real world middleware platform for safety critical applications. Preliminary experimental campaigns have been conducted to evaluate the proposed approach.

1 Introduction

As the use of Off-The-Shelf (OTS) based software systems in the context of safety critical applications is increasing, the problem of diagnosing software faults is of paramount importance. Indeed, these systems are carried out with reliable and redundant hardware configurations, making software faults the major concern. A real world application domain comes from Air Traffic Control (ATC) systems which have to fulfill strict safety and dependability requirements. Appropriate international bodies (e.g., Eurocontrol[1] in Europe) take care of delivering regulations and development methodologies that have to be followed by standard ATC systems in order to fulfill safety requirements. As for example, systems have to be able to mitigate serious incidents *during normal system execution*, thus inducing the need for on-line error detection, fault location, and treatment mechanisms.

Fault diagnosis is the process of determining the cause of errors, both in location and in nature. On-line diagnosis is conducted during system execution. Conversely, off-line diagnosis is performed through a *post-mortem* analysis of system failure data. However, the way the root cause(s) of a failure can be identified depends on the system fault model. The cause of a software error is always a bug (also called *software defect*)[2], which is a permanent fault. A plenty of studies, from both dependability and software engineering communities, demonstrated that the manifestation of a software fault may be permanent as well as transient. According to the classification proposed in [1], *Bohrbugs* result in permanent errors. They can be discovered by testing and static analysis activities. *Heisenbugs*, instead, elude the testing process and they cannot be reproduced systematically since they result in transient manifestations. Studies on field data analysis show that most of software faults are due to overloads, timing and exception errors or race conditions [2, 3], hence they are transient in nature. Transient manifestations of software faults represent the trickiest issue for diagnosis in that they hamper the definition of an exhaustive system fault model at design/development time. Furthermore, they cannot be faced by traditional diagnosis approaches, that mainly cope with hardware-induced errors and their symptoms within a software system. In fact, most of these approaches have the ultimate aim of discriminating transient manifestations, since they could induce the isolation of a node which is not actually affected by a fault. This would reduce available resources and impact the reliability level of the overall system [4]. Hence, a novel diagnostic approach is needed in charge of coping with software faults. Focus of this paper is on software fault diagnosis in safety

This work has been partially supported by the Consorzio Interuniversitario Nazionale per l'Informatica (CINI) and by the Italian Ministry for Education, University, and Research (MIUR) within the frameworks of the Centro di ricerca sui sistemi Open Source per la applicazioni ed i Servizi MIssion Critical (COSMIC) Project (`www.cosmiclab.it`). It is also in the context the "Iniziativa Software" Project, an Italian Research project which involves Finmeccanica company and several Italian universities (`www.iniziativasoftware.it`).

[1] `www.eurocontrol.int`

[2] In this work we refer to bugs as undeliberate programming mistakes thus neglecting code corruptions that result into maliciuos faults.

978-0-7695-3138-0/08 $25.00 © 2008 IEEE
DOI 10.1109/EDCC-7.2008.26

IEEE computer society

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 4, 2008 at 10:47 from IEEE Xplore. Restrictions apply.

critical systems. More specifically, it proposes the design and the implementation of a system for the on-line diagnosis of software faults in a middleware platform for safety critical applications. In order to deal with the transient manifestations, our approach combines both detection and location, similarly to system level diagnosis approaches, such as the one presented in [5]. Being the proposed approach applied to the context of safety critical systems, a special care is devoted to the confidence of the performed diagnosis. In other words, the output of the diagnosis process is the fault location and description along with a confidence level, which indicates the goodness level of the performed diagnosis (i.e., low values for confidence suggest that there is a high probability that the diagnosis was incorrect). CARDAMOM, a middleware for ATC applications, jointly developed by two leading European companies in the context of ATC, namely SELEX-SI and THALES, has been used as real case study. Starting from system logs, the proposed diagnostic system is in charge of performing error detection and fault identification. Its capability of managing ignorance about the system fault model is due to a learning strategy. Preliminary experiments have been conducted to evaluate the effectiveness of the proposed approach.
The rest of paper is organized as follows. The next section gives an overview of related work. The proposed approach is described in section 3, whereas the resulting architecture is discussed in section 4. Section 5 presents the preliminary experimental campaign. Final remarks and directions of future work are discussed in section 6.

2 Related research

Existing diagnosis techniques can be classified from several perspectives. One is to consider methods that require system models, which are usually referred to as *model based* techniques, and methods that rely on system data, which are usually defined as *data based* techniques. Model and data based techniques differ in the nature of *a priori* knowledge they use for diagnosing a fault. As for the former, base knowledge comes from a deep understanding of system behavior. The latter, instead, take their knowledge from system data (current and/or past executions data). According to this perspective, a thorough classification is provided in [6, 7, 8].
As for model based diagnosis techniques, they use a system model to predict system normal behavior via a set of observable variables. When something is detected which differs from observations, diagnosis procedures are triggered in order to identify fault candidates. Several means have been used to formalize system model, such as Hidden Markov Chains and Bayesian networks. Hidden Markov Chains (HMC) [9] are well suited to represent problems where the internal state of a certain entity is not known and can only be inferred from external observations. An important examples is provided in [10], where a diagnostic mechanism is proposed by accounting for the full chain of Monitored Component, Deviation Detection and State Diagnosis. Bayesian networks are used to infer system insights and a priori system knowledge, by using probabilistic reasoning systems [11, 12]. If the system is properly modeled, these techniques lead to high diagnosis quality, in terms of accuracy and latency. On the other hand, models turn out to be not realistic in many cases, due to the simplifying assumptions required to face mathematical issues, thus they should be used only for medium-complex systems.
As for data based techniques, they do not require *a priori* knowledge about the system. They need, instead, data collected from system executions. For these techniques, data collection is a tricky process, thus what to collect and how to collect represent key issues to be addressed.
In general, data based techniques can be applied as off-line analysis procedures, or as on-line mechanisms. Once data has been collected, off-line procedures analyze system error data to derive trend analysis and to identify the nature of occurred faults. Among these procedures, it is worth citing the Dispersion Frame Technique, proposed in [13], a statistical technique which aims to identify failures root causes and underlying trends from actual system data. Other works propose similar techniques, such as [14] and [15], which are based respectively on the Principal Component Analysis (a statistical feature extraction methods) and neural networks, to extract relevant information from system data aiming to correlate the manifested error with the corresponding fault. Furthermore, expert and/or rule based systems have been widely used to diagnose faults [16]. However, they are specialized systems in charge of solving problems with respect to specific, narrow, application domains. These techniques require knowledge to be acquired and then coded into a knowledge base. Inference procedures have to be developed for diagnostic reasoning. Expert systems are easy to implement and they assure transparent reasoning; however, they are very system specific and difficult to be updated. Although off-line diagnostic procedures represent an effective strategy when system knowledge is partial and/or inadequate, conditions under which the system is observed can vary from an installation to another, thus casting doubts on the statistical validity of the results.
On-line mechanisms are mainly used to identify faulty components among a set of homogeneous ones, or to assess the nature of the fault affecting a component (e.g., to distinguish between transient and permanent/intermittent faults). An interesting mechanism for fault diagnosis has been presented in [17], where a set of simple threshold-based functions, namely *alpha-count*, are defined to discriminate permanent from transient faults. It has been also used in the context of COTS and legacy based software systems to di-

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 4, 2008 at 10:47 from IEEE Xplore. Restrictions apply.

agnose hardware induced errors [18]. It is also worth citing a recent work that proposes an on-line diagnosis and recovery framework combining count-and-threshold algorithms and existing distributed diagnosis approaches, to assess the health of a node [4]. Node isolation is accomplished if errors are detected on the node itself. Although it defines a complete framework for on-line diagnosis, and it is intended to become a fundamental milestone in the context of diagnosis, its view to transient manifestations considerably differs from that of the present work.
In some circumstances, hybrid approaches are the most effective way to overcome the limitations of single diagnosis techniques. A hybrid technique has been proposed in [12] combining Bayesian networks and threshold based mechanisms to discriminate transient from permanent faults. In [19], a model based technique has been combined with a rule-matching algorithm in order to identify system faults, whereas in [20] a data based technique (i.e., *active probing*) has been combined with a Bayesian network in order to update system information and validate diagnosis results. Hybrid approaches can be also used in order to exploit external system knowledge, i.e., knowledge coming from external sources, such as human operators who are skilled into the system. Such a knowledge, usually referred to as *experience*, can be helpful to improve quality diagnosis, thus it is desirable to integrate it into the diagnostic process. A hybrid approach is proposed in this work, for the purpose of combining data and model based techniques, aiming to improve system description and diagnosis over time.

3 The proposed approach

In the case of software faults diagnosis in complex modular systems, several sources of failures have to be taken into account. These are due to the integration of OTS items, which may exhibit unpredictable behaviors when used out of their intended profile (i.e., configuration and operating environment in which they have been tested) thus affecting the dependability of the overall system. Furthermore, a fault can be subject to several propagation traces due to the myriad of interactions among modules. This means that (i) the system fault model cannot be completely characterized at design/development time, and (ii) drawing a simple and realistic mathematical model, concerning both with normal and faulty conditions, becomes too complicated.
We assume target software systems to be equipped with redundant and reliable hardware configurations, hence we do not care about hardware related faults. This is reasonable since companies working in the context of safety critical systems usually perform *in-house* development and testing. This work addresses complex software systems, distributed on several nodes and which communicate through a network infrastructure. Each node is organized in software layers and it is made up of several *Diagnosable Units* (DUs), representing *atomic* software entities, at it is shown in Figure 1. Such a system model does not care neither about the number of nodes and DUs, nor about system topology.

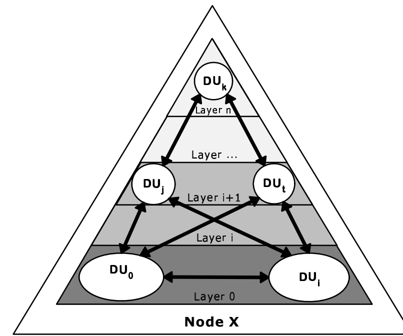

Figure 1. System's node model

A high level description of the proposed approach is represented in Figure 2. It encompasses both detection and location phases. Since we are addressing operational systems, we leverage data coming from system log files and we use them as the main source of knowledge. They are the input for the detection module, D, that is in charge of triggering an alarm if an error occurs. More specifically, D generates an Alarm Document (AD) containing information related to a condition that deviates from system normal behavior. This is achieved by means of anomaly detection, as it will be explained later. Alarm related information is processed by the location module, L, whose main goal is to identify the cause of the error, i.e., the fault. Of course, this is not trivial since the same alarm may be generated as an indication of many different faults.
The location problem has been addressed as a classification problem. Starting from alarm information (which is collected in the AD), L has to infer the presence of a fault, and to associate it to a *class*. More specifically, classification is the process of associating manifestations (i.e., errors) to the root cause (i.e., the fault). The identification of fault classes can be performed by leveraging existing log files, e.g., resulting from system testing campaigns, as well as knowledge coming from experts into the system.

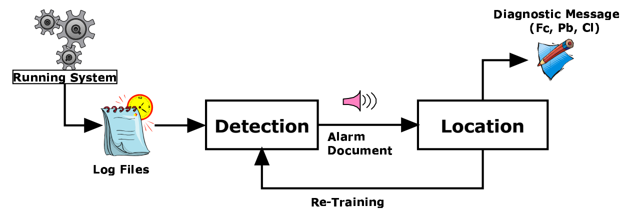

Figure 2. Overview of the proposed diagnosis approach

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 4, 2008 at 10:47 from IEEE Xplore. Restrictions apply.

Location output is structured into a Diagnostic Message (DM) containing the class of the fault (Fc), the associated probability value (Pb), and a confidence level in the performed diagnosis (Cl). This is helpful to improve the detection phase via a feedback action ('Re-training'in Figure 2), as it will be described later. The following subsections provides details about each phase of the proposed approach.

3.1 On the source of knowledge

When coping with log files, well-known filtering, coalescence and collection problems have to be addressed [21]. However, we focused our attention to the way they can be leveraged for diagnosis purposes.

The first point to be addressed concerns the tracing in the propagation of faults. This work encompasses ***inter-node*** and ***intra-node*** propagation phenomena. The former occurs when the cause of the occurred failure lies on a different machine, hence it has been propagated through the network. The latter, instead, is due to fault propagation within a single node, i.e., the fault is propagated among software layers. For this reason, data from all the software layers play a key role into the identification of propagation paths.

The second issue is how to 'interpret' log files, i.e., how logged data can be used for diagnosis. We exploited log files both in a ***direct*** and ***indirect*** way. Direct observation means that useful information is carried by log entries (e.g., a label indicating the severity of a registered event or a timestamp). Indirect observation, instead, means that useful insights into diagnosis are inferred, i.e., they are not explicitly reported into the log. As for example, by profiling log files with respect to logging frequency, we are able to discover errors: if a DU does not write entries for a *long time*, i.e., for a time which is much longer than the average, it is likely to be affected by a fault (e.g., it may be a process in hang). This is an effective strategy since logging mechanisms are generally set up to log information periodically even if nothing wrong is occurring (e.g., heartbeat).

3.2 Detection strategy

As previously stated, the system fault model can change over time when coping with software faults in complex systems. Errors can occur during system execution, which are the manifestation of faults that never occurred before. D would be not able to generate alarms in such a case, i.e., something unknown may go unnoticed thus resulting in a False Negative (FN). In order to avoid the presence of FNs, that have clearly to be minimized in a critical scenario, we propose a 'pessimistic' detection strategy. This means that an alarm is triggered when something occurs that deviates from system normal behavior, i.e., when the system is perceived to be in an erroneous state. We say that an alarm is generated due to a 'suspected error'.

Alarms are triggered by D starting from the interpretation of log files. The output of detection block is an Alarm Document AD, i.e., a document containing information related to detection of the erroneous state of the system. In particular, the IP source address, process and thread identifiers, timestamps, and the size of allocated memory are contained in the AD.

This detection strategy has been achieved by means of

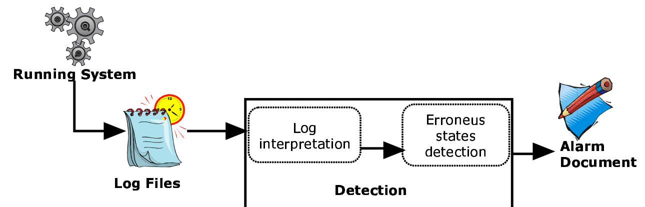

Figure 3. Detector block diagram

anomaly detection. This is for the purpose of discriminating normal behaviors from 'suspected errors'. More specifically, these represent anomalies in that they do not belong to set of normal events that have been used to train the detector which is, in fact, a binary classifier. Note that 'suspected errors' represent the whole set of errors that can affect the monitored system. Only a subset of them will correspond to actual errors, i.e., only a part of them represents actual erroneous states. The responsibility for establishing the actual nature of a 'suspected error' is given to the location block.

Well-known quality metrics used to evaluate the quality of classification are Precision, P, and Recall, R. The former deals with False Positives (FPs); it is the ratio between the number of errors that really occurred (True Positives, TPs) and the total number of alarms that have been triggered by the detector, hence $P = TPs/(TPs + FPs)$. It equals one in the best case. Note that, according to this detection strategy, FPs are a subset of 'suspected errors'. In fact, a 'suspected error' which is not the manifestation of an actual fault corresponds to a false alarm triggered by D. Clearly, in order to optimize P, FPs have to be minimized.

R, instead, encompasses FNs; more precisely, it is the ratio between the number of TPs and the total number of occurred errors, hence $R = TPs/(TPs + FNs)$.

As it will be discussed in section 5, we adopt these metrics to evaluate the quality of the proposed detection strategy. We also evaluated detection latency, i.e., interval of time between the error occurrence and the time of alarm triggering.

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 4, 2008 at 10:47 from IEEE Xplore. Restrictions apply.

3.3 Location strategy

Since D can trigger FPs, we designed a 'distrustful' location module, in charge of determining the actual nature and location of occurred faults.
When L receives an alarm, three circumstances can occur. First, the suspected error was not the manifestation of an actual fault, i.e., a FP has been triggered. Second, a fault actually occurred which L is able to identify. Third, the suspected error was actually due to a fault but L cannot identify it, let us say it is an 'UNKNOWN' fault. In other words, in this case L recognized that the erroneous state detected by D is actually a consequence of a fault, but it does not know which kind of fault occurred. As previously stated, location problem has been addressed as a classification problem. In particular, L has to associate the suspected error to the actual fault. Fault classes have to be defined in this case, which map actual faults along with their descriptions, in order to gain insight into the fault nature. Of course, the number of classes depends on the number of faults that can affect the system. In order to deal with FPs and 'UNKNOWN' faults, we introduced a special class, namely 'NO DANGER'. The confidence level, Cl, is crucial to assert whether the suspected error is due to an actual, and unknown, fault or it is a FP. In particular, we assume that an 'UNKNOWN' fault occurred if a 'NO DANGER' response is provided by L with a low value for Cl, otherwise a FP has been triggered.
Once an AD has been delivered by D, the document is processed by a Document Processing module (see Figure 4) in order to i) extract the information needed to locate the potential fault (e.g., IP source address and process identifiers), and ii) to extract the needed features to perform the classification. With respect to these, the Document Processing module is in charge of building the features vocabulary and of performing data conversion. This is in order to make data suitable for the Classification Module: features vocabulary is commonly built by selecting the words that appear in the document more frequently; these have to be converted in numerical values to perform classification. The Classification Module provide an n-dimensional array of probability values, $P_i[n]$ (n is the number of fault classes) indicating the probability that the occurred fault belongs to a given class. The maximum among P_i values establishes the fault class, Fc. We decided to integrate also external knowledge into the location process. To this aim, an 'Adjudicator' has been introduced which is in charge of combining this knowledge with P_i values, in order to calculate Pb, i.e., the actual probability that a fault belongs to a given class. Of course, this further step can cause the fault class Fc to be changed, let us say $F'c$ the final response. A Threshold Block (TB) is then used to calculate Cl basing on Pb. As previously stated, a DM is generated containing the results of the performed diagnosis in terms of $F'c$, Pb, and Cl. Figure 4 shows L internal organization.
This location strategy allows to improve both location and

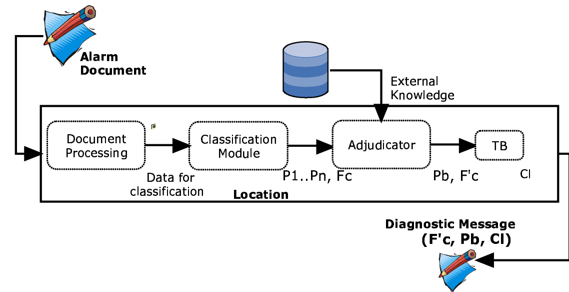

Figure 4. Location block diagram

detection phases during system lifetime. This is mainly due to Cl which plays a key role in the entire diagnosis process. First, by dealing with confidence, location can be improved in order to deal with new kinds of faults. This is because a low Cl suggests that an 'UNKNOWN' fault is likely to have occurred, hence further investigations are needed. They should bring to the definition of a new fault class to be added into the location classifier. Of course, the complexity of this updating operation depends on the type of classifier being used. It may require human operations to be even performed off line and which can take long time. This issue has to be carefully addressed when designing the location module. Of course, a good trade off between classification performances and updating overhead has to be achieved.
Second, the confidence level allows to improves detection precision (P) via the 're-training' action depicted in Figure 2. Indeed, if a 'NO DANGER' diagnosis is provided with a high confidence, there is a high probability that detection was incorrect in that it triggered a FP. Thus, the detector can be 're-trained' to reduce the number of FPs *cum cognitio causae*. Of course, this is a non trivial operation as well. In fact, 're-training' means that the classifier has to be trained by using a new training set which includes FPs in the set of normal examples. We think to this operation as a periodic task to be performed off-line. Actually, it is quite difficult to re-train the classifier on the fly, especially in the case of a diagnosis system that has to perform on-line diagnosis.

4 Diagnosis system architecture

The diagnosis approach proposed in section 3 results in the architectural scheme we describe in this section. Detection and a localization layers communicate through a publish-subscribe infrastructure. This is to keep decoupled detection and location systems thus preserving scalability.

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 4, 2008 at 10:47 from IEEE Xplore. Restrictions apply.

4.1 Detection layer

According to the system model depicted in Figure 1, detection system has been designed as shown in Figure 5. The

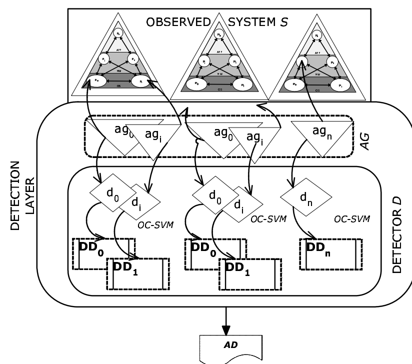

Figure 5. Detection System Architecture

Alarm Generation sublevel (AG) performs the task of interpreting log files. It activates the actual detection layer which performs, instead, the task of detecting 'suspected errors'. Detectors and alarm generators are as many as DUs are. When ag_i triggers an alarm, the detector d_i is invoked (arrows in Figure 5), as well as all the detectors d_j $\forall i \neq j$. This is for managing faults propagation. Pattern recognition techniques have been used to this aim, which will be described in section 5 with respect to the actual implementation of the diagnosis engine.
The alarm generator ag_i will provide d_i with a single chunk of the log file. This chunk contains the entry which caused the alarm, as well as all the entries written within a time interval of ws.[3] This is in order to shorten log files processing time, thus making faster the entire diagnosis process. The others generators, a_j, will provide the detectors d_j with a chunk of the log file related to the same time interval.
Once they have been invoked, all the detectors process the received log chunk in order to isolate those entries which refer to the error. As previously stated, this is achieved through a supervised anomaly detection algorithm. More specifically, we used a binary Support Vector Machine (SVM) classifier, namely One Class SVM (OCSVM). SVM technique has faced classification problems as the problem of finding a separating hyperplane in a multidimensional input space. In the case of OCSVM, training data lie in the first class, whereas the second class just encompasses the origin of this hyperplane. This is in order to find the maximal margin hyperplane which performs the best separation between training data and the origin. In our case, classifier has been trained by using only positive examples. Data close to the origin are classified as 'outliers', i.e., as suspected errors.
The output of each detector is a *Detection Document* (DD) containing only the log file entries which are suspected to be related to the error, and which have occurred *closely* to it. Hence, detectors provide data that have been already 'filtered'. A similar data mining approach is proposed in [22]. Detectors represent the publishers of the publish subscribe infrastructure. They publish DDs that are merged by the subscriber, into the AD.

4.2 Location layer

As previously stated, location is performed through document classification techniques i.e., techniques that aim to categorize documents on the base of their content (e.g., latent semantic indexing, naive Bayes classifiers...). To the best of our knowledge, this is a novel approach in the field of software faults diagnosis.
The location block, L, is in charge of classifying the received AD document[4] by processing its content via a Multiclass Probability SVM (MPSVM) classifier. Multiclass SVM (MSVM) classifiers generalize SVM to the multicategory case [23]. They have been widely used in the field of bioinformatics, face recognition, and document classification for web applications. MPSVM classifiers represent a further extension that provides also a probability value related to the performed classification (cfr. P_i in section 3).
External knowledge has been integrated into the location process by the means of a module implementing Dempster Shafer Theory (DST) [24], as shown in Figure 6. A similar approach has been proposed in [25].
DST is a generalization of the traditional Bayesian theory. It is an effective alternative to traditional probabilistic theory for the mathematical representation of uncertainty. DST is based on the idea of obtaining degrees of belief for one question from subjective probabilities for a related question, and uses Dempster's rule for combining such degrees of belief (when they are based on independent items

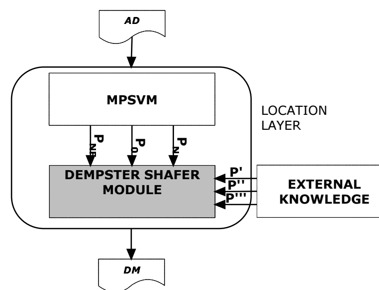

Figure 6. Location block architecture

[3] w is a tunable parameter of the proposed strategy. Sensitivity analysis has been conducted to set it properly which is not described in this paper.

[4] the subscriber process has to be considered a part of L

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 4, 2008 at 10:47 from IEEE Xplore. Restrictions apply.

of evidence). It allocates a probability mass to set or intervals, rather than to a single event, and does not require an assumption regarding the probability of the individual constituents of the set or interval. DST represents an effective strategy to be used when it is not possible to obtain a precise measurement from experiments, or when knowledge is obtained from experts. They key point of DST which has been of interest for the scope of this paper, is the combination of evidence obtained from multiple sources and the modeling of conflicts between them. In this work, external knowledge is given by the *a priori* probabilities that a fault belongs to a given class. They have been combined with P_i according to the Dempster Shafer combination rule. Pb has been calculated by the means of the *'maximum belief'* decision function encompassed by DST.
From implementation point of view, Cl calculation has been included into the Dempster Shafer module.
It is worth noting that such a strategy poses no limitations to the number of external sources which can be integrated into the process. Further work will be devoted to integrate additional knowledge, e.g., coming from Operating System level. A complete view of system architecture is shown in Figure 7. Note that detection and location have been per-

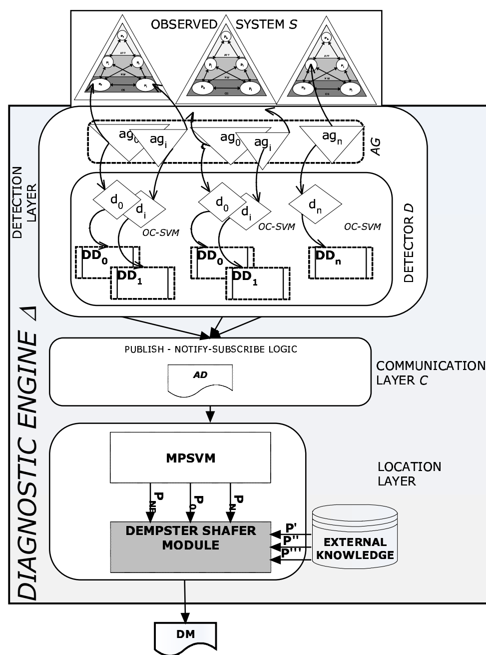

Figure 7. Diagnostic System Architecture

formed differently. The former is performed *locally* i.e., by means of alarms generators and detectors in charge of monitoring each DU in the system. This is because each DU can be a potential point of fault manifestation. The latter, instead, is performed *globally* i.e., at system level. This is because faults propagation is common in complex and modular distributed system, hence significant information can be gained by merging local data.

5 Case Study and Preliminary Results

The proposed approach has been implemented in the context of a middleware platform for ATC, namely CARDAMOM [5], a CORBA based platform intended for Linux environment. It is based on several OTS items, from CORBA ORBs to XML parsers, and it is organized in components. A system management component is responsible for platform and applications processes management. It performs system monitoring and treatment actions in case of failures. Logging facilities are provided by the middleware allowing to record information on system executions. The CARDAMOM based system, used to perform experiments, fits the system model depicted in Figure 1. In particular, each node in the system has a three-layered structure composed of operating system, middleware, and application levels; a DU is represented by a software process.

5.1 Implementation details

Classification tools. An Open Source library, namely LibSVM[6], has been used to perform classifications. It provides command-line tools to perform training phase (*svm-train*) and classification (*svm-predict*). Several steps are needed to properly setup these tools: (i) data conversion in a given format, (ii) data normalization (optional), (iii) SVM classifier selection (OCSVM, MSVM....), (iv) kernel function selection (radial basis function, sigmoidal...), (v) parameter setting, depending on the selected classifier and kernel function. As for data format, target class (i.e., the fault class used to train the classifier) and features have to be listed. Features and values have to be separated by ':' ($< feature_i : value >$) and they have to be listed in ascending order. We used these tools to realize both OCSVM and MPSVM classifications, and we used Radial Basis kernel Function (RBF) in both cases.

Workload application and training phase. In order to produce training data, we setup experimental campaigns. In particular, we develop a client server application and we let it run both under normal and faulty conditions. Normal executions were needed to gain positive examples used to train the OCSVM classifier. On the other hand, in order to train the location classifier and to identify fault classes, as well as to get *a priori* probabilities (i.e., external knowledge), we run the workload under faulty conditions. To achieve this aim, we conducted a fault injection campaign. The workload uses CARDAMOM Fault Tolerance (FT)

[5]An open source version is available at the URL: `http://cardamom.objectweb.org/`

[6]`http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/`

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 4, 2008 at 10:47 from IEEE Xplore. Restrictions apply.

facilities. Clients forward requests to a primary server, if the primary server fails, a new replica is elected (according to a warm passive replication style). Crash and server failures are encompassed by CARDAMOM (according to the FT CORBA specifications), and recovery actions are implemented for these classes of failures. As previously stated, application management policies are provided by CARDAMOM, thus two platform processes are launched on each node in charge of managing applications, as well as the a process providing FT facilities. For this reason, along with the server process (i.e., the injection target), we also monitored three CARDAMOM processes. They represent DUs, since they are in charge of producing log files.
Table 1 summarizes injected faults we used to train classifiers. The leftmost column indicates the fault class ID. The detection has been trained by using only log files from fault free executions (Class '0'), since it has just to perform anomaly detection. We take into account process crashes and hangs. Hangs have been divided in active hangs (class 2), i.e, the process is scheduled by the kernel even if it is not providing the actual service, and in passive hangs (class 4), i.e., the process is blocked indefinitely. The location module has been trained by using both information from normal and faulty executions since it has to be aware of normal and faulty behaviors to perform classification. In particular, log files generated by faulty runs have been processed in order to select features and to build the vocabulary for document classification.

As for injection, we injected faults into the application source code (server and servant processes), and we let them to be randomly activated. For instance, the `exit()` function has been invoked by using a pseudo-random variable, in order to emulate transient manifestations. Furthermore some of the injected faults can manifests differently, depending on the activation conditions (e.g., the observation can be different according to the invalid reference with respect to class 3). This is in order to take into account transient manifestations which characterize software faults. By observation we mean the perceived failure at client side.

Table 1. Injection Summary

INJECTED SOFTWARE FAULTS			
CLASS	**LOCAT/OBSERV**	**DESCRIPTION**	**INJECTION**
0	******	NORMAL	*****
1	Server (run( )) Application Stall	CRASH	exit( )
2	Server Application Stall	HANG	many threads allocation
3	Server (run( )) Clients Blocked	CRASH	invalid servant reference
4	Servant Clients Blocked	HANG	blocked mutex
5	Servant Clients Blocked	CRASH	exit( )

Tests have been executed repeatedly, accounting for a total of 233 runs. They have been conducted on 5 nodes of the cluster which is installed at our laboratory, equipped with a dual Xeon 8GHz processor and 4Gb of RAM memory.

Detection and Location. As for detection, log files have been processed by means of pattern recognition techniques. Alarm generators send to detectors a a portion of the log with $w = 1s$. Alarm generators have been implemented via Linux scripting. Log files associated to each DU have been continuously read by these scripts that are in charge of performing both *direct* and *indirect* observation of logs. The former is accomplished by invoking detectors when a given label ('WARNING', 'ERROR', and 'FATAL') is detected into the log. The latter, instead, consists of triggering detectors when the setup value for logging frequency is exceeded (this value has been calculated by processing log files of more than 100 CARDAMOM application log files. It resulted to be 0.2s on average).
As for features, tests have been performed on the location module to achieve a good trade off between classification performances and feature space dimensionality. In particular, we choose 17 features (i.e., our vocabulary is made up of 17 words). Even if SVM techniques behaves well with high numbers of features, we experienced that higher values did not provide significant improvement (up to 24); overfitting has been observed with more than 35 features.
We used a simple threshold function to calculate Confidence Level Cl. In particular:

$$Cl = \begin{cases} High & \text{if } Pb \geq 0.8 \\ Medium & \text{if } 0.3 \leq Pb < 0.8 \\ VeryLow & \text{if } Pb < 0.3 \end{cases}$$

Of course, this is quite a simple schema and we will devote future work to calculate Cl more precisely.

5.2 Results

Testing phase has been realized by running 11 different application scenarios. In order to evaluate system's capability of dealing with unknown faults, we injected a fault that had been not injected during the training phase. In particular, we caused the servant to crash by sending a `kill` signal to the process. In the following we refer to `'Tx_y'` testing applications, where `x` is the test ID, and `y` is the fault class. With `'?'` we indicate the unknown fault.

Detection. We evaluated detection in terms of precision, P, and recall, R. They have been measured by comparing data included into the Alarm Documents (ADs) to actual anomaly data. Table 2 shows P and R with respect to the performed tests. Note that P is never less than R, thus confirming the pessimistic strategy we want to develop in order to cope with software faults in safety critical scenarios.

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 4, 2008 at 10:47 from IEEE Xplore. Restrictions apply.

Table 2. Detection Precision and Recall.

TEST ID	P	R
T1_0	1	1
T2_5	0.96	0.92
T3_1	1	0.98
T4_2	0.96	0.92
T5_1	1	1
T6_3	1	1
T7_4	1	1
T8_4	1	1
T9_5	1	1
T10_?	1	1

Location. We focus on two experiments we conducted to evaluate the location phase. The first experiment aims to test module's capability of locating the source of an error that is the manifestation of a known fault. We injected a fault of class 1 into a CARDAMOM platform process, rather than into the application code. The Diagnostic Message (DM) that has been provided is shown in Figure 8. Location module has been able to classify the fault with high confidence, and the source of the failure has been identified in terms of IP address and Process ID (these are deduced by the log file of the faulty process).

```
********************* DIAGNOSIS MESSAGE *********************
DIAGNOSIS:   Fault
TYPE:        CRASH  (class 1)
Pb:                     0.98
EVENT-TIME:  12:44:59
DIAGNOSIS_TIME:   Sat Feb 2 12:45:03 CEST 2008
IP:      192.168.0.184    PID: 9964
**********************************************************
CONFIDENCE LEVEL IN THE DIAGNOSIS:      High
```

Figure 8. DM. A known fault has been located

The second experiment is related to the injection test T10_?. As previously stated, we injected an "unknown fault" into the servant. Although an alarm has been triggered by the detector, the location module has been not able to diagnose the fault. The provided DM is shown in Figure 9. Note that, as we would expect, a 'NO DANGER' has been diagnosed with a low confidence level.

```
********************* DIAGNOSIS MESSAGE *********************
DIAGNOSIS:   NO DANGER
TYPE:        no fault (class 0)
Pb: 0.2
EVENT-TIME:  19:59:57
DIAGNOSIS_TIME:   Mon Feb 4 20:0:00 CEST 2008
IP:      192.168.0.181    PID: 2132
**********************************************************
CONFIDENCE LEVEL IN THE DIAGNOSIS:      Very low
```

Figure 9. DM. A 'NO DANGER' has been diagnosed with low confidence

We will devote further work to update the location module by adding improved knowledge. We also have to perform further experiments to test how detection precision can be improved by re-training the detector when a 'NO DANGER' is diagnosed with high confidence.
Table 3 shows P and R experienced by the location module.

Table 3. Precision and recall wrt to injected faults.

FAULT CLASS	P	R
0	0.95	0.92
1	1	1
2	1	0.937
3	0.894	0.894
4	1	0.95
5	0.974	0.944

Diagnosis Latency. Preliminary evaluation campaigns aiming to measure diagnosis latency, i.e. the time between fault occurrence and its diagnosis, have also been conducted. The diagnosis process has been conceived as made up of four phases as it is shown in Table 4. This reports the minimum and the maximum time needed to perform each phase. Communication phase is the most time consumptive step as it should have been expected. However, both detection and location phases require less than 300 *ms* with a low value of standard deviation (less than 30*ms*).

Table 4. Diagnosis Phases

Diagnosis phase	Min time (*s*)	Max time (*s*)
Alarm generation phase	1	2
Detection phase	0.154	0.499
Communication phase	1	3
Location phase	0.181	0.208

Further Insigths. We installed a prototype of the diagnosis engine on a testbed we are using in the context of an industrial research activity on CARDAMOM robustness. An interesting phenomenon has occurred during these experimentations. An application server hanged and this eluded middleware logging and detection mechanisms. Indeed, even if the server was not able to serve incoming requests, it was considered *healthy* by application management processes and fault tolerance actions have not been initiated. Calling client remained blocked waiting for the server to serve requests. It has been possible to diagnose the occurred event, thanks to the indirect log observation.

6 Conclusions and Future Work

The paper addressed the problem of on-line software faults diagnosis in complex software systems. Transient manifestations of software faults represent the trickiest is-

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 4, 2008 at 10:47 from IEEE Xplore. Restrictions apply.

sue to be addressed since they hamper a complete knowledge of system fault model at design/development time. A diagnosis approach has been proposed, combining both error detection and fault location processes. A hybrid diagnosis approach is the core of the proposed strategy according to which both system knowledge and diagnosis performances improve during system execution. A pessimistic detector has been designed, which generates an alarm whenever the system shows anomalous behaviors. The location module is in charge of asserting the actual nature of the error that generated the alarm, i.e., it has to establish whether it was the manifestation of an actual fault or a false alarm. Additionally, it can exploit external knowledge about the system via a probabilistic reasoning approach. Feedback actions have been encompassed in order to update detector, and a confidence level has been introduced to gain insights into the goodness of the performed diagnosis. The approach has been implemented to diagnose software faults in a middleware platform for safety critical applications. Preliminary experimental campaigns have been conducted to evaluate the proposed approach. Ongoing work is focusing on the improvement of the location process by integrating other sources of *experience*, such as Operating System log files. Future work will be devote to setup a massive experimental campaign aiming to evaluate the effectiveness of the proposed approach.

References

[1] P. Jalote Y. Huang and C. Kintala. Two Techniques for Transient Software Error Recovery. In *Workshop on Hardware and software architectures for fault tolerance: experiences and perspectives*, pages 159–170. Springer-Verlag, 1994.

[2] M. Sullivan and R. Chillarege. Software Defects and Their Impact on System Availability - A Study of Field Failures in Operating Systems. *21st Int. Symp. on Fault-Tolerant Computing (FTCS-21)*, pages 2–9, 1991.

[3] S. Biyani R. Chillarege and J. Rosenthal. Measurement of Failure Rate in Widely Distributed Software. In *FTCS '95:Proc. of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, page 424. IEEE Computer Society, 1995.

[4] N. Suri M. Serafini, A. Bondavalli. On-Line Diagnosis and Recovery: On the Choice and Impact of Tuning Parameters. *IEEE Trans. on Dependable and Secure Computing*, 4(4):295–312, 2007.

[5] N. H.Vaidya and D. K. Pradham. Safe System Level diagnosis. *IEEE Trans. on Computers*, 43(3):367–370, 1994.

[6] K. Yin V. Venkatasubramanian, R. Regnaswamy and S. N. Kavuri. A Review of Process Fault Detection and Diagnosis, Part I: Quantitative Model-based Methods. *Journ. of Computers and Chemical Engeneering*, 27(3):293–311, 2003.

[7] K. Yin V. Venkatasubramanian, R. Regnaswamy and S. N. Kavuri. A Review of Process Fault Detection and Diagnosis, Part II:Qualitative Models and Search Strategies. *Journ. of Computers and Chemical Engeneering*, 27(3):313–326, 2003.

[8] K. Yin V. Venkatasubramanian, R. Regnaswamy and S. N. Kavuri. A Review of Process Fault Detection and Diagnosis, Part III:Process history Based Methods. *Journ. of Computers and Chemical Engeneering*, 27(3):327–346, 2003.

[9] R. Pattipati J. Ying, T. Kirubarajan. A Hidden Markov Model Based Algorithm for Fault Diagnosis with Partial and Imperfect Test. *IEEE Transactions on systems, man and cybernetic*, 30(4):451–462, Nov. 2000.

[10] A. Daidone, F. Di Giandomenico, A. Bondavalli, and S. Chiaradonna. Hidden Markov Models as a Support for Diagnosis: Formalization of the Problem and Synthesis of the Solution. *Proc. of 25th IEEE Symposium on Reliable Distributed Systems, 2006. SRDS '06.*, pages 245–256, Oct. 2006.

[11] L. Jinling Z. Yongli, H. Limin. Bayesian Network-Based Approach for Power Systems Fault Diagnosis. *IEEE Transaction on Power Delivery*, 21(2):634–639, Apr. 2006.

[12] M. Pizza, L. Strigini, A. Bondavalli, and F. Di Giandomenico. Optimal Discrimination Between Transient and Permanent Faults. In *3rd IEEE High Assurance System Engineering Symposium*, pages 214–223, Bethesda, MD, USA, 1998.

[13] T. T. Y. Lin and D. P. Siewiorek. Error Log analysis: Statistical Modeling and Heuristic Trend Analysis. *IEEE Transactions on Reliability*, 39(4):419–432, 1990.

[14] J.MacGregor P. Nomikos. Monitoring Batch Processes Using Multiway Principal Component Analysis. *American Institute of Chemical Engineers Journal*, 40(8):1361–1375, 1994.

[15] L. Sansone A. Bernieri, M. D'Apuzzo and M. Savastano. A Neural Network Approach for Identification and Fault Diagnosis non Dynamic Systems. In *Proc. of the Instrumentation and Measurement Technology Conference imtc/93. Conference Record.* IEEE Computer Society, 1993.

[16] P.Lee W. Becraft. An Integrated Neural Network/Expert System Approach for Fault Diagnosis. *Computers and Chemical Engineering*, 17(10):1001–1014, 1993.

[17] F. Di Giandomenico A. Bondavalli, S. Chiaradonna and F. Grandoni. Threshold-Based Mechanisms to Discriminate Transient from Intermittent Faults. *IEEE Trans. on Computers*, 49(3):230 – 245, March 2000.

[18] D.Cotroneo A.Bondavalli, S.Chiaradonna and L. Romano. Effective Fault Treatment for Improving the Dependability of COTS- and Legacy-based Applications. *IEEE Trans. on Dependable and Secure Computing*, 1(4):223–237, 2004.

[19] P. Varadharajan G. Khanna and S. Bagchi. Self Checking Network Protocols: A Monitor Based Approach. In *SRDS '04: Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS'04)*, pages 18–30. IEEE Computer Society, 2004.

[20] S. Ma et al. I. Rish, M. Brodie. Adaptive Diagnosis in Distributed Systems. *IEEE Transaction on Neural Networks*, 16(5):1088–1109, Sep. 2005.

[21] Z. Kalbarczyk R.K. Iyer and M. Kalyanakrishnam. Measurement-Based Analysis of Networked System Availability. *Performance Evaluation Origins and Directions*, 2000.

[22] R. P. Jagadeesh Chandra Bose and S. H. Srinivasan. Data Mining Approaches to Software Fault Diagnosis. In *RIDE'05: Proc.of the 15th Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications*, pages 45–52. IEEE Computer Society, 2005.

[23] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[24] M. Fedrizzi RR. Yager, J. Kacprzyk. *Advances in the Dempster-Shafer Theory of Evidence*. John Wiley & Sons, Inc. New York, NY, USA, 1994.

[25] Y. Li et al. Z. Hu. Method of Combining Multi-class Svms Using Dempster-Shafer Theory and its Application. In *Proc. of American Control Conference (ACC'05)*. IEEE Computer Society, 2005.

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 4, 2008 at 10:47 from IEEE Xplore. Restrictions apply.