Online Black-Box Failure Prediction for Mission Critical Distributed Systems

Roberto Baldoni[1], Giorgia Lodi[1], Luca Montanari[1], Guido Mariotta[1], and Marco Rizzuto[2]

[1] "Sapienza" University of Rome, Via Ariosto 25, 00185, Rome, Italy
[2] Selex Sistemi Integrati, Finmeccanica Group, Rome, Italy

Abstract. This paper introduces a novel approach to failure prediction for mission critical distributed systems that has the distinctive features to be *black-box*, *non-intrusive* and *online*. The approach combines Complex Event Processing (CEP) and Hidden Markov Models (HMM) so as to analyze symptoms of failures that might occur in the form of anomalous conditions of performance metrics identified for such purpose. The paper describes an architecture named CASPER, based on CEP and HMM, that relies on sniffed information from the communication network of a mission critical system, only, for predicting anomalies that can lead to software failures. An instance of CASPER has been implemented, trained and tuned to monitor a real Air Traffic Control (ATC) system. An extensive experimental evaluation of CASPER is presented. The obtained results show (i) a very low percentage of false positives over both normal and under stress conditions, and (ii) a sufficiently high failure prediction time that allows the system to apply appropriate recovery procedures.

1 Introduction

Context and Motivation. Distributed mission critical systems such as air traffic control, battlefield or naval command and control systems consist of several applications distributed over a number of nodes connected through a LAN or WAN. The applications are constructed out of communicating software components that are deployed on those nodes and may change over time. The dynamic nature of applications is principally due to (i) the employed policies for resilience to software or hardware failures, (ii) the adopted load balancing strategies or (iii) the management of new comers. In such complex real time systems, failures may happen with potentially catastrophic consequences for their entire functioning. The industrial trend is to face failures by using, during operational system life, supervision services that are not only capable of detecting and certificating a failure, but also predicting and preventing it through an analysis of the overall system behavior. Such services shall have a minimum impact on the supervised system and possibly no interaction with the operational applications. The goal is to plug-in a "ready-to-use observer" that acts at run time and is both *non-intrusive* and *black-box*, i.e., it considers nodes and applications as black boxes.

F. Ortmeier and P. Daniel (Eds.): SAFECOMP 2012, LNCS 7612, pp. 185–197, 2012.

In mission critical systems, a large amount of data deriving from communications among applications transits on the network; thus, the "observer" can focus on that type of data, only, in order to recognize many aspects of the actual interactions among the components of the system. The motivation to adopt this non-intrusive and black-box approach is twofold. Firstly, applications change and evolve over time: grounding failure prediction on the semantic of the applications' communications would require a deep knowledge of the specific system design, a proven field experience, and a non-negligible effort to keep aligned the supervision service to the controlled system. Secondly, interactions between the service and system to be monitored might lead to unexpected behaviors, hardly manageable as fully unknown and unpredictable.

Contribution. In this paper we introduce the design, implementation and experimental evaluation of a novel online, non-intrusive and black-box failure prediction architecture we named CASPER that can be used for monitoring mission critical distributed systems. CASPER is (i) *online*, as the failure prediction is carried out during the normal functioning of the monitored system, (ii) *non-intrusive*, as the failure prediction does not use any kind of information on the status of the nodes (e.g., CPU, memory) of the monitored system; only information concerning the network to which the nodes are connected is exploited as well as that regarding the specific network protocol used by the system to exchange information among the nodes (e.g., SOAP, GIOP); and (iii) *black-box*, as no knowledge of the application's internals and of the application logic of the system is analyzed. Specifically, the aim of CASPER is to recognize any deviation from normal behaviors of the monitored system by analyzing symptoms of failures that might occur in the form of anomalous conditions of specific performance metrics. In doing so, CASPER combines, in a novel fashion, Complex Event Processing (CEP) [1] and Hidden Markov Models (HMM) [2]. The CEP engine computes at run time the performance metrics. These are then passed to the HMM in order to recognize symptoms of an upcoming failure. Finally, the symptoms are evaluated by a failure prediction module that filters out as many false positives as possible and provides at the same time a failure prediction as early as possible. Note that we use HMM rather than other more complex dynamic bayesian networks [3] since it provides us with high accuracy, with respect to the problem we wish to address, through simple and low complexity algorithms. We deployed CASPER for monitoring a real Air Traffic Control (ATC) system. Using the network data of such a system in the presence of both steady state performance behaviors and unstable state behaviors, we first trained CASPER in order to stabilize HMM and tune the failure prediction module. Then we conducted an experimental evaluation of CASPER that aimed to show its effectiveness in timely predicting failures in the presence of memory and I/O stress conditions.

Related Work. A large body of research is devoted to the investigation of approaches to online failure prediction. [4] presents an error monitoring-based failure prediction technique that uses Hidden Semi-Markov Model (HSMM) in order to recognize error patterns that can lead to failures. This approach is

event-driven as no time intervals are defined: the errors are events that can be triggered anytime. [5] describes two non-intrusive data driven modeling approaches to error monitoring: one based on a Discrete Time Markov Model, and a second approach based on function approximation. The difference between our approach and these works is twofold: firstly we propose a symptoms monitoring system in contrast to error monitoring, moreover, our approach is not event-based but it can be defined period-based [6] as we use Hidden Markov Models (discrete time) to recognize, in the short term, patterns of specific performance metrics exhibiting the evidence of symptoms of failures. In the context of symptoms monitoring mechanisms, there exist research works that use black-box approaches [7]. [8] presents ALERT; an anomaly prediction system that considers the hosts of the monitored system as black-boxes and that collects metrics concerning CPU consumption, memory usage, input/output data rate. The authors in [9] consider the problem of discovering performance bottlenecks in large scale distributed systems consisting of black-box software components. The system introduced in [9] solves the problem by using message-level traces related to the activity of the monitored system in a non-intrusive fashion (passively and without any knowledge of node internals or semantics of messages). [10] analyzes the correlation in time and space of failure events and implements a long-term failure prediction framework named hPrefects for such a purpose.

We differ from the earlier mentioned works as we employ an approach that is not only black-box but also non-intrusive. It is true that we use message-level traces as [9]; however, we combine in a novel fashion both CEP for network performance metrics computation and HMM to infer the system state. Note that typically HMM is widely used in failure prediction to build a components' state diagnosis [11]. In our architecture the entire system state is modeled as a hidden state and thus inferred by HMM.

Finally, there exist other types of researches [12] that apply online failure prediction to distributed stream processing systems, mostly using decision tree and classifiers. We do not specifically target these systems, and we use different techniques (CEP and HMM) to predict failures.

Structure of the Paper. Section 2 presents the failure and prediction model we adopt in the design of CASPER described in section 3. Section 4 introduces CASPER implementation for a real ATC mission critical system. Section 5 discusses the experimental evaluation of CASPER we have conducted in ATC. Finally, section 6 concludes the paper highlighting our future works.

2 Failure and Prediction Model

We model the distributed system to be monitored as a set of nodes that run one or more services. Nodes exchange messages over a communication network. Nodes or services can be subject to failures. A failure is an event for which the service delivered by a system deviates from its specification [13]. A failure is always preceded by a fault (e.g., I/O error, memory misusage); however, the

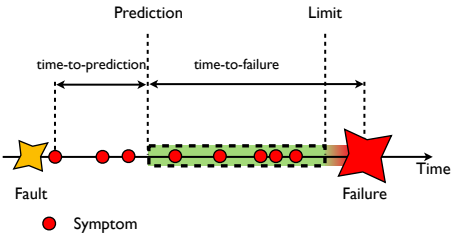

Fig. 1. Fault, Symptoms, Failure and Prediction

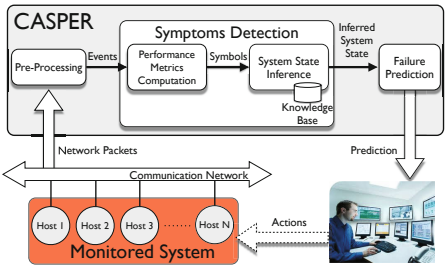

Fig. 2. The modules of the CASPER failure prediction architecture

vice versa might not be always true. i.e., a fault inside a system could not always bring to a failure as the system could tolerate, for example by design, such fault.

Faults that lead to failures, independently of the fault's root cause, affect the system in an observable and identifiable way. Thus, faults can generate side-effects in the monitored systems till the failure occurs. Our work is based on the assumptions that a fault generates increasingly unstable performance-related symptoms indicating a possible future presence of a failure, and that the system exhibits a steady-state performance behavior with a few variations when a non-faulty situation is observed [14,15,7]. In Figure 1 we define *Time-to-failure* the distance in time between the occurrence of the prediction and the software failure event. The prediction has to be raised before a time *Limit*, beyond which the prediction is not sufficiently in advance to take some effective actions before the failure occurs. We also consider the *time-to-prediction* which represents the distance between the occurrence of the first symptom of the failure and the prediction.

3 The CASPER Failure Prediction Architecture

The architecture designed is named CASPER and is deployed in the same sub-network as the distributed system to be monitored. Figure 2 shows the principal modules of CASPER that are described in isolation as follows.

Pre-processing Module. It is mainly responsible for capturing and decoding network data required to recognize symptoms of failures and for producing streams of events. The network data the Pre-Processing module receives as input are properly manipulated. Data manipulation consists in firstly decoding data included in the headers of network packets. The module manages TCP/UDP headers and the headers of the specific inter-process communication protocol used in the monitored system (e.g., SOAP, GIOP, etc) so as to extract from them only the information that is relevant in the detection of specific symptoms (e,g., the timestamp of a request and reply, destination and source IP addresses of two communicating nodes). Finally, the Pre-Processing module adapts the extracted network information in the form of *events* to produce streams for the use by the second CASPER's module (see below).

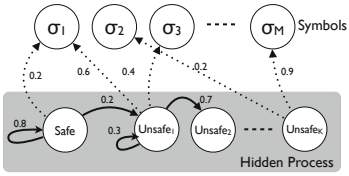

Fig. 3. Hidden Markov Models graph used in the system state inference component

Symptoms Detection Module. The streams of events are taken as input by the Symptoms detection module and used to discover specific performance patterns through complex event processing (i.e., event correlations and aggregations). The result of this processing is a system state that must be evaluated in order to detect whether it is a safe or unsafe state. To this end, we divided this module into two different components, namely a *performance metrics computation* component and a *system state inference* component.

The performance metrics computation component uses a CEP engine for correlation and aggregation purposes. It then periodically produces as output a representation of the system behavior in the form of *symbols*. Note that, CASPER requires a *clock mechanism* in order to carry out this activity at each *CASPER clock cycle*. The clock in CASPER allows it to model the system state using a discrete time Markov chain and let the performance metrics computation component coordinate with the system state inference one (see below). The representation of the system behavior at run time is obtained by computing *performance metrics*, i.e., a set of time-changing metrics whose value indicates how the system actually works (an example of network performance metric can be the round trip time). In CASPER we denote symbols as σ_m (see Figure 3), where $m = 1, \ldots, M$. Each symbol is built by the CEP engine starting from a vector of performance metrics: assuming P performance metrics, at the end of the time interval (i.e. the clock period), the CEP engine produces a symbol combining the P values. The combination of performance metrics is the result of a discretization and a normalization: each continuous variable is discretized into slots of equal lengths. The produced symbol represents the system status during the clock period[1].

The system state inference component receives a symbol from the previous component at each CASPER clock cycle and recognizes whether it is a correct or an incorrect behavior of the monitored system. To this end, the component uses the Hidden Markov Models' forward probability [2] to compute the probability that the model is in a given state using a sequence of emitted symbols and a knowledge base(see Figure 2). We model the system state to be monitored by means of the *hidden process*. We define the states of the system (see Figure 3) as *Safe*, i.e., the system behavior is correct as no active fault [13] is present; and *Unsafe*, i.e., some faults, and then symptoms of faults, are present.

Failure Prediction Module. It is mainly responsible for correlating the information about the state received from the system state inference component of

[1] For further details please refer to the technical report [16].

the previous CASPER module. It takes in input the inferred state of the system at each CASPER clock-cycle. The inferred state can be a safe state or one of the possible unsafe states. Using the CEP engine, this module counts the number of consecutive *unsafe* states and produces a failure prediction alert when that number reaches a tunable threshold (see below). We call this threshold *window size*, a parameter that is strictly related to the *time-to-prediction* shown in Figure 1.

3.1 Training of CASPER

The knowledge base concerning the possible safe and unsafe system states of the monitored system is composed by the parameters of the HMM. This knowledge is built during an initial training phase. Specifically, the parameters are adjusted by means of a training phase using the max likelihood state estimators of the HMM [2]. During the training, CASPER is fed concurrently by both recorded network traces and a sequence of pairs `<system-state,time>`. Each pair represents the fact that at time `<time>` the system state changed in `<system-state>`[2].

3.2 Tuning of CASPER Parameters

CASPER architecture has three parameters to be tuned whose values influence the quality of the whole failure prediction mechanism in terms of false positives and time-to-prediction. These values are (i) the length of the CASPER *clock period*; (ii) the *number of symbols* output by the performance metrics computation module; (iii) the length of the failure prediction, i.e., *window size*.

The length of the clock period influences the performance metrics computation and the system state inference: the shorter the clock period is, the higher the frequency of produced symbols is. A longer clock period allows CASPER to minimize the effects of outliers. The number of symbols influences the system state inference: if a high number of symbols is chosen, a higher precision for each performance metrics can be obtained. The failure prediction window size corresponds to the minimum number of CASPER clock cycles required for raising a prediction alert. The greater the window size, the more the accuracy of the prediction, i.e., the probability that the prediction actually is followed by a failure (i.e. a true positive prediction). The tradeoff is that the time-to-prediction increases linearly with the windows size causing shorter time-to-failure (see Figure 1); During the training phase, CASPER automatically chooses the best values for both clock period and number of symbols, leaving to the operator the responsibility to select the windows size according to the criticality of the system to be monitored.

4 Monitoring a Corba-Based ATC System with CASPER

CASPER has been specialized to monitor a real Air Traffic Control system. ATC systems are composed by middleware-based applications running over a

[2] As the training is offline, the sequence of pairs `<system-state,time>` can be created offline by the operator using network traces and system log files.

collection of nodes connected through a Local Area Network (LAN). The ATC system that we monitored is based on CORBA [17] middleware. CASPER intercepts GIOP messages produced by the CORBA middleware and extracts several information from themin order to build the representation of the system at run time. In this section we describe how the events are represented starting from the GIOP messages and how the performance metrics representing the system state are computed.

Event Representation. Each GIOP message intercepted by CASPER becomes an event feeding the CEP engine of the performance metrics computation component. Each event contains (i)*Request ID*: The identifier of a request-reply interaction between two CORBA entities; (ii)*Message Type*: A field that characterizes the message and that can assume different values (e.g., Request, Reply, Close Connection) and (iii)*Reply Status*: It specifies whether there were exceptions during the request-reply interaction and, if so, the kind of the exception. In addition, we insert into the event further information related to the lower level protocols (TCP/UDP) such as source and destination IP, port, and timestamp. In order not to capture sensitive information of the ATC system (such as flight plans or routes), CASPER ignores the payload of the messages.

Performance Metrics. Events sent to the CEP engine are correlated online so as to produce so-called performance metrics. After long time of observations of several metrics of the ATC CORBA-based system, we identified the following small set of metrics that well characterize the system, showing a steady behavior in case of absence of faults, and an unstable behavior in presence of faults:

- *Round Trip Time:* elapsed time between a request and the relative reply;
- *Rate of the messages carrying an exception:* the number of reply messages with exception over the number of caught messages;
- *Average message size:* the mean of the messages size;
- *Percentage of Replies:* the number of replies over the number of requests in a given spatial or temporal window;
- *Number of Requests without Reply:* the number of requests expecting a reply that do not receive the reply;
- *Messages Rate:* the number of messages exchanged in a fixed time.

To compute the performance metrics we correlate the sniffed network data using the CEP engine ESPER [1]. This choice is motivated by its low cost of ownership compared to other similar systems (e.g. [18]) and its offered usability.

5 CASPER Experimental Evaluation

We deployed CASPER so as to monitor an Air Traffic Control system of Selex Sistemi Integrati, one of the major players of the ATC market. The first part of the work on the field has been to collect a large amount of network traces from the ATC underlying communication network when in operation. These traces represented steady state performance behaviors. Additionally, on the testing

environment of the ATC system we stressed some of the nodes till achieving software failure conditions, and we collected the relative traces. In our test field, we consider one of the nodes of the ATC system to be affected by either Memory or I/O stress (according to the experience of the ATC designers, these two stress conditions are typical of the observed system). After collecting all these traces, we trained CASPER. At end of the training phase, we deployed CASPER again on the testing environment of the ATC system in order to conduct experiments in operative conditions. Our evaluation assesses the system state inference component accuracy and the failure prediction module accuracy. In particular, we evaluate the former in terms of N_{tp} (number of true positives) the system state is unsafe and the inferred state is "system unsafe"; N_{tn} (number of true negatives): the system state is safe and the inferred state is "system safe"; N_{fp} (number of false positive): the system state is safe but the inferred state is "system unsafe"; and N_{fn} (number of false negatives): the system state is unsafe but the inferred state is "system safe". Using these parameters, we compute the following metrics that define the accuracy of CASPER: We evaluate the latter module in terms

$$\text{Precision: } p = \frac{N_{tp}}{N_{tp}+N_{fp}} \quad \text{Recall (TP rate): } r = \frac{N_{tp}}{N_{tp}+N_{fn}}$$
$$\text{F-measure: } F = 2 \times \frac{p \times r}{p+r} \qquad \text{FP Rate: } f.p.r. = \frac{N_{fp}}{N_{fp}+N_{tn}}$$

of N_{fp} (number of false positive): the module predicts a failure that is not actually coming and N_{fn} (number of false negatives): the module does not predict a failure that is coming.

Testbed. We deployed CASPER in a dedicated host located in the same LAN as the ATC system to be monitored (see Figure 2). This environment is actually the testing environment of the ATC system where new solutions are tested before getting into the operational ATC system. The testing environment is composed by 8 machines, 16 cores 2.5 GHz CPU, 16 GB of RAM each one. It is important to remark that CASPER does not know the application nor the service logic nor the testbed details.

5.1 Faults and Failures

The ATC testbed includes two critical servers: one of the server is responsible for disk operations (I/O) and another server is the manager of all the services. In order to induce software failures in the ATC system, we apply the following actions in such critical servers: (i)*memory stress*; that is, we start a memory-bound component co-located with the manager of all ATC services, to grab constantly increasing amount of memory resource; (ii)*I/O stress*; that is, we start an I/O-bound component co-located with the server responsible for disk operations, to grab disk resources. In both cases we brought the system to the failure of critical services. During the experiment campaign, we also considered the CPU stress; however, we discovered that due to the high computational power of the ATC nodes, the CPU stress never causes failures. For this reason we decided not to show the results of these tests.

5.2 Results

The results are divided in three subsections: the training of CASPER, the tuning of the parameters (both before the deployment), and the failure prediction evaluation (using network traces and deploying the system). We considered three performance metrics: *Number of request without reply*, *Round Trip Time* and *Message Rate*, the more influenced by the stress conditions.

Training of CASPER. We trained CASPER (see Section 3.1) using the following two types of recorded traces: a type of trace between 10 and 13 minute long (5 traces in total) in which the ATC system exhibits a steady-state behavior. These traces are taken from the ATC system when in operation; and a second type of trace between 10 and 11 minute long (4 traces per each kind of injected stress, i.e., memory and I/O stress) in which one of the services of the ATC system fails. These traces are taken from ATC system's testing environment. During the training phase, the performance metrics computation component produces a symbol at each CASPER clock cycle. Thanks to the set of pairs `<system-state,time>` we are able to represent the emitted symbols in case of safe and unsafe system states. Figure 4 illustrates these symbols. Each symbol is calculated starting from a combination of three values. In this case, we have 6 possible values per each performance metric; the number of different symbols is therefore $6 \times 6 \times 6 = 216$. Observing Figure 4 we can notice that the majority of the emissions belong to the interval $[0, 2]$ for the *Round Trip Time*, and $[0, 1]$ for *Number of Request Without Reply* and *Message Rate*. Starting from the symbols represented in Figure 4, the HMM-based component builds its knowledge base.

Tuning of CASPER Parameters: Clock Period and Number of Symbols. After the training of HMM, CASPER requires a tuning phase to set the clock period and number of symbols in order to maximize the accuracy (F-measure, precision, recall and false positive rate) of the *symptoms detection module* output. This tuning phase is done by feeding the system with a recorded network trace (different from the one used during the training). We can see that the best choice of the clock period is 800 milliseconds. CASPER tries 4 different values of clock (100ms, 300ms, 800ms, 1000ms) and compute the F-Measure for each value and for each possible number of symbols. A clock period of 800 milliseconds yields a higher F-Measure value than the other clock values in most of the number of symbols considered, thus, CASPER set the clock period to 800 milliseconds. Once fixed this clock period, the second parameter to define is the number of symbols. Figure 5 shows the precision, recall, F-measure and false positive rate of the symptoms detection module varying the number of symbols. CASPER considers the maximum difference between the F-measure and the false positive rate in order to choose the ideal number of symbols (ideally, F-measure is equal to 1 and f.p.r. to 0). As shown in Figure 5, considering 216 symbols (6 values per performance metric) we obtain $F = 0.82$ and $f.p.r. = 0.12$ which is actually the best situation in case of memory stress.

Tuning of CASPER Parameters: Window Size. The window size is the only parameter that has to be tuned by the operator according to the tradeoff

discussed in Section 3.2. We experimentally noticed that during fault-free executions the system state inference still produced some false positives. However, the probability that there exists a long sequence of false positives in steady-state is very low. Thus, we designed the failure prediction module to recognize sequences of consecutive clock cycles whose inferred state is not safe. Only if the sequence is longer than a certain threshold CASPER rises a prediction. The length of these sequences multiplied by the clock period (set to 800ms) is the window size. The problem is then to set up a reasonable threshold in order to avoid false positive predictions during steady-state. Figure 6 illustrates the number of the false positive varying the window size. From this Figure it can be noted that the window size has to be set to at least 16 seconds in order not to incur in any false positives. Let us remark that the window size also corresponds to the minimum time-to-prediction. All the results presented below are thus obtained using a window size of 16 seconds.

Results of CASPER Failure Prediction. We run two types of experiments once CASPER was trained and tuned. In the first type, we injected the faults described in section 5.1 in the ATC testing environment and we carried out 10 tests for each type of fault[3]. In the second type, we observed the accuracy of CASPER when monitoring for 24h the ATC system in operation. These types of experiments and their related results are discussed in order as follows. As first test, we injected a memory stress in one of the node of the ATC system till a service failure. Figure 7 shows the anatomy of this failure in one test. The ATC system runs with some false positive till the time the memory stress starts at second 105. The sequence of false positives starting at second 37 is not sufficiently long to create a false prediction. After the memory stress starts, the failure prediction module outputs a prediction at second 128; thus, the time-to-prediction[4] is 23s. The failure occurs at second 335, then the time-to-failure is 207s, which is satisfactory with respect to ATC system recovery requirements. A failure caused by I/O stress happens after 408 seconds from the start of the stress (at second 190) and has been predicted at time 222 after 32 seconds of stress, with a time-to-prediction equal to 376 seconds before the failure. In general, we obtained that in the 10 tests we carried out, the time-to-failure in case of memory stress varied in the range of [183s, 216s] and the time-to-prediction in the range of [20.8s, 27s]. In case of I/O stress, in the 10 tests, the time-to-failure varied in the rage of [353s, 402s] whereas the time-to-prediction in the range of [19.2s, 24.9s]. Finally, we performed a 24h test deploying CASPER on the network of the ATC system in operation. In these 24 hours the system exhibited steady-state performance behavior. CASPER did not produce any false positive along the day. Figure 8 depicts a portion of 400 seconds of this run.

[3] The number of tests was limited by the physical access to the ATC testing environment. In fact, every experiment takes actually 2 hours to be completed due to data storage, the stabilizing and rebooting of the ATC system after the failure.

[4] The prediction time depends on how the system reacts to the injected stress and on the injected stress itself.

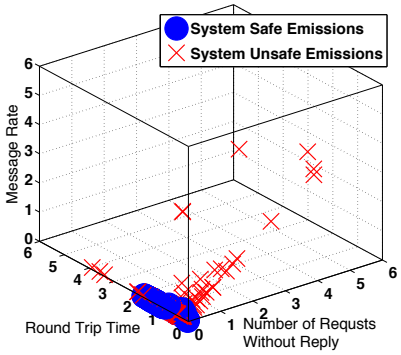

Fig. 4. Symbols emitted by the performance metrics computation component in case of a recorded trace subject to memory stress

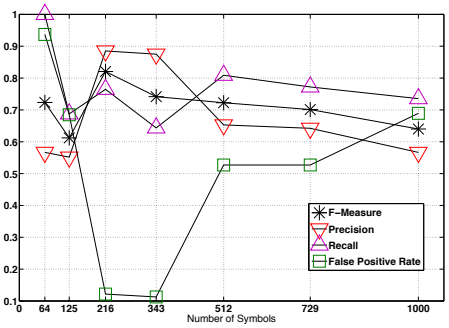

Fig. 5. Performance of the symptoms detection module varying the number of possible symbols in case of a recorded trace subject to memory stress

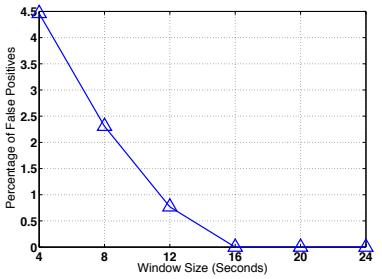

Fig. 6. False positives varying the window size. CASPER is fed with a recorded trace behaving in steady-state.

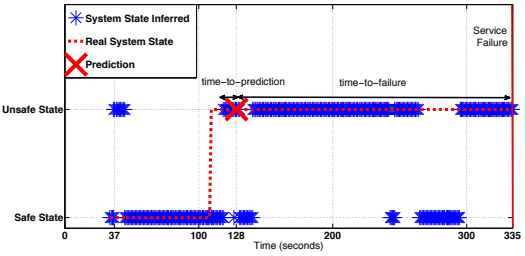

Fig. 7. Failure prediction in case of memory stress starting at second 105. time-to-prediction 23s, time-to-failure 207s.

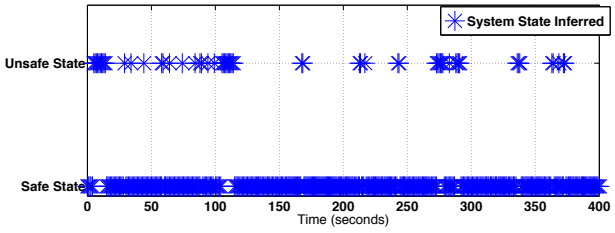

Fig. 8. 400 seconds of a steady-state run of the ATC system in operation

6 Conclusions and Future Work

We presented an architecture to predict online failures of mission critical distributed systems. The failure prediction architecture, namely CASPER, provides accurate predictions of failures by exploiting only the network traffic of the monitored system. In this way, it results non intrusive with respect to the nodes hosting the mission critical system and it executes a black-block failure prediction as no knowledge concerning the layout and the logic of the mission critical distributed system is used. To the best of our knowledge, this is the first failure detection system exhibiting all these features together. Let us remark that the black-box characteristic has a strategic value for a company developing such systems. Indeed from a company perspective the approach is succeeding as long as the failure prediction architecture is loosely bound to the application logic as this logic evolves continuously over time. This non intrusiveness of our approach has the advantage that no additional load to the monitored system is introduced and it can be applied in all the existing middleware based systems without modifications of the architecture. As future work we are developing a CASPER version capable of executing online training. i.e., the training is done just connecting to the monitored system without any human intervention. This will make CASPER a complete "plug-and-play" failure prediction system. The advantage of the online training solution is that CASPER can analyze a huge amount of network data. The disadvantage is that the training phase can last for long time as CASPER does not have any external clue concerning the safe or faulty system state.

References

1. Esper: Esper project web page (2011), `http://esper.codehaus.org/`
2. Rabiner, L., Juang, B.: An introduction to hidden markov models. IEEE ASSP Magazine 3(1), 4–16 (1986)
3. Murphy, K.: Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis, UC Berkeley, Computer Science Division (2002)
4. Salfner, F.: Event-based Failure Prediction: An Extended Hidden Markov Model Approach. PhD thesis, Department of Computer Science, Humboldt-Universität zu Berlin, Germany (2008)
5. Hoffmann, G.A., Salfner, F., Malek, M.: Advanced Failure Prediction in Complex Software Systems. Technical Report 172, Berlin, Germany (2004)
6. Yu, L., Zheng, Z., Lan, Z., Coghlan, S.: Practical online failure prediction for blue gene/p: Period-based vs event-driven. In: Proc. of IEEE/IFIP DSN-W 2011, pp. 259–264 (2011)
7. Williams, A.W., Pertet, S.M., Narasimhan, P.: Tiresias: Black-box failure prediction in distributed systems. In: Proc. of IEEE IPDPS 2007, Los Alamitos, CA, USA (2007)
8. Tan, Y., Gu, X., Wang, H.: Adaptive system anomaly prediction for large-scale hosting infrastructures. In: Proc. of ACM PODC 2010, pp. 173–182. ACM, New York (2010)

9. Aguilera, M.K., Mogul, J.C., Wiener, J.L., Reynolds, P., Muthitacharoen, A.: Performance debugging for distributed systems of black boxes. SIGOPS Oper. Syst. Rev. 37, 74–89 (2003)
10. Fu, S., Zhong Xu, C.: Exploring event correlation for failure prediction in coalitions of clusters (2007)
11. Daidone, A., Di Giandomenico, F., Bondavalli, A., Chiaradonna, S.: Hidden markov models as a support for diagnosis: Formalization of the problem and synthesis of the solution. In: SRDS 2006, Leeds, UK, pp. 245–256 (2006)
12. Gu, X., Papadimitrioul, S., Yu, P.S., Chang, S.P.: Online failure forecast for fault-tolerant data stream processing. In: ICDE 2008, pp. 1388–1390 (2008)
13. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.E.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Sec. Comput. 1(1), 11–33 (2004)
14. Hood, C., Ji, C.: Proactive network-fault detection. IEEE Transactions on Reliability 46(3), 333–341 (1997)
15. Thottan, M., Ji, C.: Properties of network faults. In: NOMS 2000, pp. 941–942 (2000)
16. Baldoni, R., Lodi, G., Mariotta, G., Montanari, L., Rizzuto, M.: Online Black-box Failure Prediction for Mission Critical Distributed Systems. Technical report (2012), http://www.dis.uniroma1.it/~midlab/articoli/MidlabTechReport3-2012.pdf
17. Object Management Group: CORBA. Specification, Object Management Group (2011)
18. IBM: System S Web Site (2011), http://domino.research.ibm.com/comm/research_projects.nsf/pages/esps.index.html