SHStream: Self-healing Framework for HTTP Video-Streaming

Carlos Augusto Cunha
Centre for Informatics and Systems
University of Coimbra
ccunha@dei.uc.pt

Luis Moura Silva
Centre for Informatics and Systems
University of Coimbra
luis@dei.uc.pt

ABSTRACT

HTTP video-streaming is leading delivery of video content over the Internet. This phenomenon is explained by the ubiquity of web browsers, the permeability of HTTP traffic and the recent video technologies around HTML5. However, the inclusion of multimedia requests imposes new requirements on web servers due to responses with lifespans that can reach dozens of minutes and timing requirements for data fragments transmitted during the response period. Consequently, web-servers require real-time performance control to avoid playback outages caused by overloading and performance anomalies. We present *SHStream*, a self-healing framework for web servers delivering video-streaming content that provides (1) load admittance to avoid server overloading; (2) prediction of performance anomalies using online data stream learning algorithms; (3) continuous evaluation and selection of the best algorithm for prediction; and (4) proactive recovery by migrating the server to other hosts using container-based virtualization techniques. Evaluation of our framework using several variants of *Hoeffding trees* and *ensemble algorithms* showed that with a small number of learning instances, it is possible to achieve approximately 98% of *recall* and 99% of *precision* for failure predictions. Additionally, proactive failover can be performed in less than 1 second.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.4.5 [**Reliability**]: Fault-tolerance

Keywords

Self-healing, Video-streaming, Failure Prediction

1. INTRODUCTION

Video-streaming is experiencing dramatic growth. Its use has been potentiated by the convergence of TV with the Internet, the emergence of new services (e.g., VoD and e-learning) and the ever increasing market of mobile video streaming boosted by the rollout of the 4G service.

The real-time characteristic of video services demands infrastructures provided with efficient performance monitoring and failure recovery capabilities. These capabilities should circumvent playback interruptions due to performance problems that compromise the upfront time invested by end-users watching the videos, leading to expensive abandonment costs.

Performance problems usually occur server-side or in the network. Network-related problems have standard solutions based on graceful degradation of video quality using *adaptive streaming* techniques [40] (e.g., the recent MPEG-DASH standard [39]). Other approaches use temporal data redundancy [16] and spatial data redundancy [34]. At server level, it is known that resource exhaustion is the main cause of performance failures in web servers [31]. It occurs as a consequence of excess of client workloads and performance anomalies. Workload-related failures can be anticipated by effective load control mechanisms [35]. By contrast, performance anomalies are difficult to detect, as they represent unexpected server states that are complex and non-reproducible. This paper focuses on this type of failures.

HTTP streaming (also known as *pseudo-streaming*) dominates the delivery of video content in the Internet. This approach uses web servers to provide streaming data to users [37], often extended with additional modules for advanced features, as *adaptive streaming*, *traffic shedding* and *timeline positioning*. Despite using the same server application, the request is no longer the unit of performance control, as it is typical for *web page services* [22] and *n-tier web applications* [6]. This is explained by long streaming responses that can last for dozens of minutes, during which they require: (1) monitoring of transmission bitrates; and (2) resumption of request responses at any point if recovery is required.

Resumption of request responses is challenging, as both the server application state and TCP connections should be recovered. State recovery is usually ensured by checkpointing techniques [15]. However, those techniques add complexity and overheads [13] which confronts the streaming real-time constraints. Additionally, checkpointing and recovering of TCP connection states requires operating system changes [41]. Proactive recovery using failure prediction is an alternative to checkpointing, by providing an opportunity to gracefully handle failures before potential outages occur.

MMSYS'13 February 27- March 1, 2013, Oslo, Norway.

Previous work in prediction of performance failures address forecasting of resource exhaustion due to memory leaks[33][25][19], correlation of temporal and spacial events [26], prediction of resource values using sequential patterns [20] and context-aware prediction [42]. These techniques were designed for *batch learning* (learning data should be available in advance and do not allow further updates) and target other types of services that do not require real-time prediction of performance failures.

This paper presents SHStream, a self-healing framework for HTTP streaming, driven by prediction of server performance failures, with the following features:

- **Load admittance**. Limits the load accepted by the server to the server nominal capacity;
- **Online learning**. Learns incrementally the server normal and abnormal behaviors and adapts to behavioral changes;
- **Failure prediction and classification**. Predicts and classifies performance anomalies for proactive recovery;
- **Model evaluation**. Picks continuously the best model from a pool of models created using several algorithms;
- **Fast recovery**. Performs proactive failover to another host when a failure is predicted, using *virtual containers*.

We present evaluation results of several variants of *Hoeffding trees* and *ensemble* algorithms, using our SHStream framework to build models of *pre-failure patterns* for failure prediction of performance failures. This paper answers the following fundamental research questions:

- How to perform online learning and prediction of performance failures in streaming services using an open framework for learning algorithms?
- Do online learning models accurately capture *pre-failure patterns* in web servers providing streaming content?
- Which algorithms have the best performance?
- Do online learning models accurately classify failure types?
- How many learning instances are required to stabilize prediction performance?
- What is the recovery cost?

The rest of this paper is structured as follows. Section 2 presents the related work. Section 3 formalizes the problem. Section 4 presents the algorithms used for failure prediction and diagnosis, implemented in Section 5. Section 6 shows results of the experimental work done to evaluate our approach. Section 7 presents conclusions.

2. RELATED WORK

Failure prediction techniques presented in previous work can be classified as: *regression of resource utilization*, *correlation of events*, *probabilistic sequential patterns* and *context-aware* models.

Regression techniques have been explored to model utilization of resources, for prediction of system and service performance. Powers et al [33] addressed the problem of forecasting system performance in enterprise systems to automate assignment of resources. Forecasting of service level objectives (SLOs) one hour ahead showed that: (1) multivariate regression and Bayesian Network Classifiers perform better than auto-regression methods; and (2) models are not reusable between machines without accuracy losses but helps bootstrapping models on machines where learning data are scarce. Sahoo et. al. [36] applied time series methods, rule-based classification and Bayesian network models to prediction of anomalous events in commercial and scientific applications. Models showed acceptable errors when evaluated with a production dataset. Hoffmann et al. [21] studied the use of several modeling techniques to predict resource consumption of the Apache Web server. Results shown that UBFs yields the best results for free physical memory prediction and SVMs performed better predicting server response times. Cherkasova et al [12] studied detection and classification of workload changes, performance anomalies and application changes in three-tier web servers using performance signatures and a regression model of CPU consumption. Kelly et al [23] investigated the use of multivariate regression models to classify performance anomalies in three types: overloading, application logic faults and configuration faults. The classification model aggregates response times of the transaction mix to discriminate between anomaly types. The approach was evaluated using three large data sets collected in global distributed systems.

Temporal and spatial correlation of failure events in computing systems was investigated previously for failure prediction [38]. Liang et. al. [26] addressed failure prediction in clusters of scientific applications. They explored temporal and spacial locality of previous failures to predict future failures and uses information about non-fatal events to predict application crashes. Experimental results using the RAS event logs of IBM BlueGene/L showed that a high number of failures can be avoided using their approach. Sequential patterns represent transitions between server states occurring according to specific probabilities. Gu [20] explored the combination of Bayesian classifiers and Markov models to predict both actual and future bottleneck failures in distributed data stream nodes. Their approach showed high levels of accuracy and precision in three scenarios: insufficient CPU, insufficient memory and memory leaks. Context-aware models groups anomalous occurrences into contexts. Tan et al. [42] proposed a context-aware anomaly prediction scheme combined with decision trees to classify component states. The approach showed better results than monolithic, incremental and ensemble approaches for several types of stream processing components with real-time prediction performance.

We extend the current state of the art by providing an integrated self-healing framework for web servers that meets

the service characteristics and time constraints of video-streaming. The novelty of our work resides in the: (1) evaluation of online incremental learning algorithms to build failure prediction models for services with characteristics similar to those of the video-streaming; (2) methodology to implement online learning through automatic labeling of learning instances; (3) open framework for learning algorithms that evaluates several models at each iteration to select the classification output of the model with best performance; and (4) evaluation of proactive failover using *container-based virtualization*. Additionally, our framework also implements a novel approach for dynamic admission of server load to avoid workload-related failures.

3. PROBLEM STATEMENT

This section describes the problems faced designing a self-healing system for streaming that addresses two main types of faulty behaviors: (1) *server fail-stop*, avoiding acceptance of new requests and transmission of video content to end-users; and (2) *reduction of transmission bitrate*, when servers put data on the network at rates below the bitrates required for playback.

3.1 Failure Prediction and Failure Type Classification

Failure prediction can be defined as follows. Being $F = \{normal, failure\}$ one server state and M the vector of values corresponding to application and system metrics, the problem resumes in incrementally learn a classifier that maps the space of possible values of M_t observed during periods $F_t = \{normal\}$ to a future failure state $F_{t+n} = \{failure\}$, being t the observation time and $n > 0$.

Modeling server *pre-failure patterns* has several challenges. Firstly, they should exist. Secondly, they should be captured by metrics. Thirdly, the model should correctly model them with a small error. Finally, the respective learning instances gathered from logs should be delimited in time to avoid mixing *pre-failure pattern* instances with *normal* instances, in the row of log instances preceding each failure.

Failure type classification provides guidance for future fault removal activities. We evaluate the ability of machine learning algorithms in determining the failure type of predicted failures. To assist the learning activity, each logged instance is labelled with the failure type involved in a fault (e.g., CPU, Memory, I/O). Being T the set of possible failure types and M the vector of values corresponding to application and system metrics, the failure type classification problem resumes in incrementally learning a classifier that maps the space of possible values of M_t to a failure type T_i, being t the time at which the failure is predicted.

3.2 Repair

SHStream requires the implementation of two recovery techniques: *connection redirection* and *web server failover*. New connection requests can be redirected to another host when the server reaches its capacity or is experiencing performance problems. Connection redirection is implemented by the HTTP protocol through the REDIRECT command. On the other hand, web server failover presents two main challenges: (1) synchronization of application states between the faulty and failover servers; and (2) transparent server migration, requiring moving the server IP address and TCP connection states to the target host to avoid breaking the current sessions established with players.

4. LEARNING ALGORITHMS

This section presents the online learning algorithms used for prediction and classification of performance failures. *Stream mining* is a recent class of data mining techniques that overcomes traditional batch learning limitations to fulfill the online learning requirements of dynamic systems [18]: (1) **Incremental learning** (parallel learning and classification of instances); (2) **Single pass through data**; (3) **Limited time and memory** (instances are processed in a small and constant time using an approximately constant amount of memory); and (4) **Any-time learning** (if stopped before its conclusion, the algorithm should provide the best possible answer). We evaluate three types of online algorithms in our framework: *decision trees*, *probabilistic classifiers* and *ensemble algorithms*.

4.1 Hoeffding trees

Decision trees are popular learners in both batch and data stream learning. They are powerful, interpretable and efficient classifiers - with n examples and m attributes, the average cost of basic decision tree induction is $O(m \cdot n \cdot logn)$. However, traditional decision trees algorithms (e.g., C4.5 [44]) requires all data available in advance and avoids further model updating. *Hoeffding decision trees* promise performance levels similar to batch decision trees with the requirements of online learning. VFDT *(Very Fast Decision Tree)* is a state of the art algorithm for creating Hoeffding trees proposed by Domingos and Hulten [14]. The performance of Hoeffding Trees has been shown comparable with traditional decision trees, Naive Bayes, kNN, and ensemble methods [10][8] but much faster and less memory consuming than traditional approaches while handling extremely large datasets. VFDT builds the three iteratively by splitting each node when the number of learned examples satisfies the *Hoeffding Bound* (1).

$$\epsilon = \sqrt{\frac{R^2 \cdot ln(\frac{1}{\delta})}{2 \cdot n}} \tag{1}$$

The *Hoeffding Bound* defines the split confidence, by stating that with probability $1-\delta$, the true mean of a random variable with range R does not differ from its estimated mean after n independent observations by more than ϵ. *Information gain* is the splitting criteria commonly used to build tree models. It is defined as the difference between the *weighted average entropy* of split subsets and the *entropy* of class distribution before splitting, being the *entropy* defined as in (2). *Entropy* measures the purity of subsets for a distribution of class labels consisting of fractions $p_1, ..., p_n$, summing to 1.

$$entropy(p_1, p_2, ..., p_n) = \sum_{i=1}^{n} -p_i \cdot log_2 p_i \tag{2}$$

We use two prediction strategies with *Hoeffding trees*. The

majority class is the strategy by default - i.e., filtering down the tree to a leaf and retrieving the most likely class label. This approach may lead to high prediction errors in failure prediction problems due to examples rarely seen. The second approach explores *Naive Bayes* to also account in the tree the conditional probabilities of the attribute values given that class. This allows inclusion of information about the number of times the attribute values were seen associated to each class in the prediction process. *Naive Bayes* is naturally incremental as it deals with heterogeneous data and missing values and is a very competitive algorithm for small datasets [7]. *Adaptive Hoeffding Trees* is another variant of *Hoeffding trees* explored that uses ADWIN [9] to evolve trees with new server behaviors, by monitoring performance of branches on the tree and replace them with new branches with higher accuracy.

4.2 Ensemble of Models

Ensembles group several models to perform classification by means of voting. That configuration has been proved to attain higher levels of accuracy than those obtained by single classifiers alone [43]. The main drawback of these algorithms is the loss of interpretability provided by several base classifiers (e.g., decision trees). We assess four ensemble algorithms in our experiments that uses *Hoeffding Trees* as base models: *Weighted Majority Algorithm* [27], *OzaBoost* [29], *OzaBag* [29] and *Option Trees* [32].

4.2.1 Weighted Majority Algorithm

Learn models by giving a positive weight to each model in the pool that correctly classifies one learning example and discounting a given ratio β of the weight of those that incorrectly classify learning instances. The number of mistakes was proven to be bounded in a sequence of predictions from a pool of algorithms A by $O \cdot log|A| + n$.

4.2.2 OzaBoost

Learns several models in a sequence, increasing weights of examples misclassified by former models in the sequence to reinforce their learning by the latter models, similarly to *AdaBoost* [17] for batch learning scenarios. This algorithm divides the total weights into two halves, giving one half to correctly classified examples and the other half to misclassified examples. Misclassified examples are reinforced intrinsically at the next sequence model by the classifier's accuracy - as it increases, the number of misclassified examples decreases, getting more weight per example.

4.2.3 OzaBag

Learning from a bootstrap replicate of examples drawn randomly from the training dataset according to a probability $Poisson(1)$, similarly to *Bagging* [11] for batch learning. Bootstrapping reduces variance errors caused by low frequency examples in the dataset, as they have low probability of being used to train models.

4.2.4 Hoeffding Option Trees

Typical *Hoeffding trees* with additional *option nodes* leading to multiple Hoeffding trees as separate paths. By representing several decision trees in a single compact structure it it possible to reduce the space required to save independent tree instances, as required for traditional ensembles. Additionally, contrasting with other ensemble models, model interpretability can be maintained if a small number of option nodes is used [24].

5. SHSTREAM IMPLEMENTATION

This section presents the architecture, algorithms and techniques implemented by the SHStream framework. The SHStream was developed in Java and has the following main dependencies:

- **SIGAR** [2]: an API to gather system reports;
- **MOA (Massive Online Analysis)**: the implementation of machine learning algorithms;
- **mod_SHS**: a *Lighttpd* module implemented by us to gather application-level metrics and control server load;
- **OpenVZ tools** [3]: for migration of virtual containers.

The SHStream framework runs within the server's host to ensure: (1) *Scalability* through avoidance of centralized data gathering and analysis; (2) *Timeliness* by enabling server adaption with minimum communication delays; and (3) *Integration* through direct control of the agent over the server.

Fig. 1 presents the architecture of SHStream. It is divided into two groups of features: monitoring and recovery. The monitoring activity starts by aggregating data every α seconds, gathered from *application reports*, *system reports* and *web server probing status reports*. Later, these data is used for load admittance, failure prediction, failure classification and learning. Only the classification output of the model with best prediction performance is selected by the framework for prediction. When one failure is predicted, the recovery is launched to handle container migration to other hosts.

The *Lighttpd* web server is installed within an OpenVZ container with the *H264 Streaming Module* [1] and the *mod_SHS*. The former module ensures advanced features to streaming users, as *time shifting seek* and *virtual video clips* for adaptive streaming. *mod_SHS* gathers application-level metrics data and redirects new connections to another server when the *load admittance* component decides that the server reached its limit. The disk is used for communication between the virtual container and the SHStream tool, using a directory shared by the container and the host domain. The *recovery manager* communicates with OpenVZ tools for migration of containers between host machines.

5.1 Data Gathering

SHStream collects the 44 system metrics provided by SIGAR, including process-related metrics (e.g., *CPU* and *memory* consumed by the web server process). At application level, it collects several performance metrics: *response time* (time until transmission of first packet), *number of failed connections*, *number of network failures*, *bytes written at the current second*, *bytes read from disk*, *number of active connections*, *number of connections executed* and *number of con-*

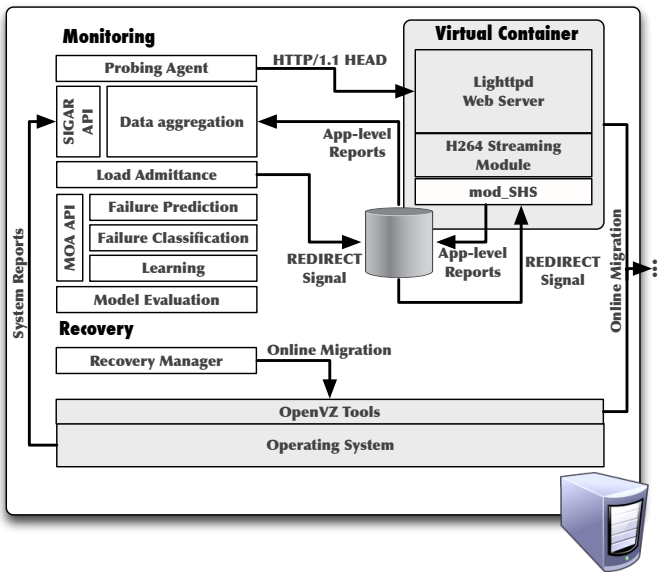


Figure 1: SHStream Architecture

nections recovered. Additionally, fail-stop failures at the process level is captured by the probing agent.

5.2 Failure Detection

We consider that the server is failing at any time t_i, if it became unresponsive or if, in average, the sum of transmitted bitrates for all active connections, since the beginning of transmission t_0 (i.e., the first byte transmitted) is smaller than the sum of their respective video encoding bitrates, calculated as in (3). That means that the server sent less bits to the network than the bits required by players to play videos up to time t_i.

$$\frac{\sum_{j=1}^{nConnections} \frac{BitsTransmitted_j(t_0(j),t_i)}{(t_i - t_0(j)) \cdot EncodingBitrate_j}}{nConnections} < 1 \qquad (3)$$

To avoid unfair reduction of transmission bitrates between connections, we also calculate the number of individual connections suffering from bandwidth reduction below their encoding bitrate (forcing the player to interrupt playback) at each time t_i. That connection condition is defined in (4):

$$\frac{BitsTransmitted(t_0, t_i)}{(t_i - t_0) \cdot EncodingBitrate} < 1 \qquad (4)$$

BitsTransmitted represents the bits transmitted by the server to the player, for a given connection, since it starts at time t_0 until the present time t_i. The player is forced to stall when that value is smaller than the video encoding bitrate.

Fail-stop failures are detected through evaluation of server responsiveness. Responsiveness is determined through server responses to HTTP/1.1 HEAD command requests, issued each α seconds and the absence of new application-level reports within the last α seconds.

5.3 Load Admittance

Our approach for load admittance resides in controlling admittance of new connections dynamically, using the bandwidth in excess being used by the connections being served. This rule allows easily adaption to changes on server and network conditions. New connections are accepted if the difference in the sum of the actual transmission bitrates TBR and the sum of the encoding bitrates EBR, for all actual established connections, is higher than a safe margin a, as shown in (5). The value of a should be carefully chosen to afford the bitrate of new connections.

$$\sum_{i=1}^{nCurrent} TBR(i) - EBR(i) > a \qquad (5)$$

SHStream controls load admittance using a flag file to inform the *mod_SHS* module that the server has reached its capacity. The existence of this file indicates that new connections should be redirected to a new server, by sending an *HTTP REDIRECT command* to the client.

5.4 Failure Prediction

The failure prediction learning process starts by delimiting the *pre-failure time window* which isolates *pre-failure states* from *normal states*. This is a non-trivial critical task that could introduce large errors in the learning process, since *pre-failure states* can be trained using data belonging to *normal states* and vice-versa. Instead of performing a sharp separation between normal and pre-failure periods, we consider very short pre-failure periods preceded by a *window of uncertainty* (Fig. 2a). Any data within this time window are ignored in the learning process, as they can belong to either state.

Algorithm 1 shows how models are handled for classification, learning and evaluation of instances. One monitoring instance containing system and application-level metrics data is picked at the time. Such instance is classified by each learning algorithm and the classification given by the model with higher performance at the moment is chosen to decide if recovery should be performed. The remaining classifications will be used later for statistics. After classification, each new instance is stored at the end of a buffer, to be used later for learning (Fig. 2b). In the same iteration, the algorithm picks the first buffer instance and evaluates it in terms of its ability to predict the failures observed since it was gathered. One failure is considered predicted by a given previous instance, if the distance between both is less or equal than the *pre-failure window size*. The buffer size is dimensioned by the *window of uncertainty*'s size, ensuring that all instances within the buffer preceding the *pre-failure window* are ignored for learning, unless all the buffer instances were marked as *normal*. In such case, all instances are learned as *normal*.

5.5 Failure Diagnosis

SHStream discriminates failures into five types: *CPU*, *Memory*, *I/O* and *misc*. The *misc* type is reserved for failures that do not fit any of the other types. Learning and evaluation of algorithms respect the failure type observed during the *pre-failure window*.

Algorithm 1 Classification, learning and evaluation using online models for failure prediction and diagnosis.

Require: $size(buffer)$ is $WindowOfUncertainty$
  loop
    $I \Leftarrow readNewInstance()$
    $f \Leftarrow isFailState(I)$
    ▷ Classification
    for $i = 1$ to $nModels$ **do**
      $p_i \Leftarrow classifyFailurePrediction(Model_i, I)$
      $c_i \Leftarrow classifyFailureType(Model_i, I)$
    end for
    if $p_{mostAccurate}$ is $true$ **then**
      $launchRecovery(c)$
    end if
    ▷ Learning
    $L \Leftarrow buildLearningInstance(I, f, p, c)$
    if not $isBufferFull(buffer)$ **then**
      jump to next loop iteration
    end if
    $addToEnd(buffer, L)$
    $F \Leftarrow removeFirst(buffer)$
    if $distanceFail(buffer) \in [1, preFailWindow]$ **then**
      for $i = 1$ to $nModels$ **do**
        $learn(Model_i, F, prefailure)$
        $updateModelStatistics(p_i, c_i, prefailure)$
      end for
    else if $distanceFail(buffer, F)$ is ∞ **then**
      $learn(Model_i, F, normal)$
      $updateModelStatistics(p_i, c_i, normal)$
    end if
    ▷ Evaluation
    $mostAccurate \Leftarrow evaluateBestModel(Model)$
  end loop

5.6 Model Evaluation Metric

Prediction metrics are calculated using the number of *true positives* (TP), *false negatives* (FN) and *false positives* (FP). *True positives* are failure scenarios predicted correctly as failures. *False negatives* are unpredicted failure scenarios and *false positives* represent normal scenarios mispredicted as failures. We use those values to calculate *recall* and *precision* (6).

$$\text{Recall} = \frac{TP}{TP + FN} \qquad \text{Precision} = \frac{TP}{TP + FP} \tag{6}$$

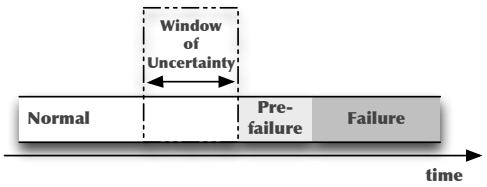
Recall represents the percentage of failure instances detected by the classifier. This metric is important to evaluate the coverage of our failure detector. *Precision* captures the true positive rate of instances classified as failures.

F-measure (7) is a standard information retrieval metric, calculated as the weighted harmonic mean of *recall* and *precision*. It provides a single value for comparison of classification performance between prediction models.

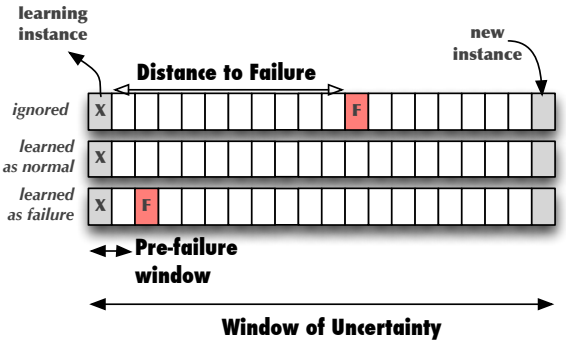
$$\text{F-measure} = \frac{2 \cdot precision \cdot recall}{(precision + recall)} \tag{7}$$

5.7 Recovery

Virtualization technology is explored by SHStream to isolate the web server and respective modules from the other processes and to build an unit of migration between physical hosts. Virtualization also avoids pre-reservation of resources in the failover host, allowing the other applications within the target host to fully use resources until they are reclaimed during migration.



(a) The window of uncertainty and the temporal location of normal, pre-failure and failure states.



(b) The three different scenarios of the learning buffer

Figure 2: Data segmentation for learning of failure prediction models.

We adopted OpenVZ, a OS-level server virtualization technology based on containers. Container-based virtualization has significantly smaller overheads than typical virtualization and paravirtualization technologies [30] because they run on top of the operating system. The container size is significantly smaller than virtual machines created by other virtualization technologies, where the guest operating system runs within the virtual machine. This characteristic ensures smaller migration downtimes, as the volume of data to be transmitted between the primary and failover hosts is smaller.

We implemented the failover mechanism by copying the container file to the failover host. Containers are migrated between hosts, carrying their network IP address. During migration, the container is suspended in the primary host, then is copied to the failover host (using *rsync* [4]) and finally, is instantiated in the failover host with the execution state of the primary host. Our tests showed that online migration is performed transparently to the user in less than 1 second. Delays of that order can be absorbed by video-player buffers.

6. EXPERIMENTAL WORK

This section presents the testbed, workloads and fault loads used in the experimental work to evaluate our approach. It further presents and discusses the results obtained to answer the research questions stated in the Section 1.

6.1 Testbed

Our tests were performed on a tested composed by four machines connected by a 100Mbps Ethernet Network: two servers (i.e., *primary* and *failover* servers), and two ma-

chines running a script that coordinates execution of *httperf* [28] instances to avoid client-side overloading. The script also commands server fault injection through *ssh commands* that invokes the *Stress* tool [5] in the server. The container contains the *Lighttpd web server 1.4.30* installed with the *H264 Streaming Module* (mod_h264_streaming version 2.2.7) and the *mod_SHS* module. The SHStream tool was installed normally outside the virtual container and was configured to gather performance metrics every 2 seconds. All machines were configured with an Intel(R) Pentium(R) D CPU 3.00GHz, 2Gb RAM, running the Linux 2.6.18-92.1.22.el5 Kernel.

6.2 Workloads and Fault Loads

We ran a single test during 94 hours using ten H.264 videos, encoded with bitrates of approximately 600 Kbps (standard quality) and 2 Mbps (high quality). Three workload types were devised with those encodings:

- **Cached**: the same file is streamed by all requests;
- **Disk**: each request streams exclusively one video file;
- **Mix**: two-third of requests stream the same file and the other one-third of requests streams one file exclusively.

The workload type impacts considerably the number of streams being served by the server. In our experiments we observed that the number of connections supported by the server for *Cached* type workloads is several times the number of connections allowed by the *Disk* configuration. This phenomenon is explained by the bottleneck accessing disk-stored content. Each workload type was submitted recurrently in sequence with the order *Cached*, *Disk* and *Mix*, with inter-request rates randomly set between 500 milliseconds and 5 seconds. The number of connections varies sinusoidally between 0 and n, being n higher than the server limit. The timespan delimited by each of those workload types is named a *scenario*. Each *scenario* is associated to a single fault type: *normal* (no fault), *CPU*, *Memory*, *I/O* and *Misc*. During the lifespan of each *scenario*, the associated fault type is injected randomly with the intensity required to cause a service failure.

6.3 Results

We ran experiments one first time without SHStream to determine its overhead. The maximum number of connections per second is the same with and without SHStream. The explanation for this observation is that streaming content is resource intensive and the SHStream overhead is negligible when compared with the resources consumed by streaming requests.

Fig. 3 shows the results of the experimental tests. One failure is considered predicted if it has a look-ahead time of at least 2 seconds to provide a temporal margin for recovery. Fig. 3a relates the number of false positives and false negatives with the number of failure instances used for learning, for the best classifier chosen at each moment. It is noticeable that the 5 false positives observed occur after a high number of learning instances. By contrast, the number of false negatives stabilizes after 15 learning instances, occurring infrequently afterwards. After stabilization, the classifier predicted almost all failures consistently (Fig. 3b). All algorithms, except **Adwin* and *Naive Bayes*, achieved high levels of *recall* (Fig. 3c) and *precision* (Fig. 3d) in the test. *Recall* stabilizes at 98% and precision is around 99% for all ensemble algorithms and 98% for standard Hoeffding trees, a little below its ensemble variant counterparts. Ensemble algorithms achieved the highest prediction performance but with small differences comparatively to Hoeffding trees. The *Weighted Majority Algorithm* has the highest *F-measure* (Fig. 3e), followed by the *HoeffdingOptionTree*, *OzaBag* and *OzaBoost*.

Fig. 3f presents the number of correct and incorrect classifications of failure types, showing that the number of misclassifications exceeds the number of correct classifications. Those results showed that online learning algorithms have problems capturing patterns associated to specific failure types. Thus, classification based on *pre-failure patterns* is unable to separate between failure types of predicted instances (i.e., *CPU*, *Memory*, *I/O* and *misc*).

6.4 Discussion

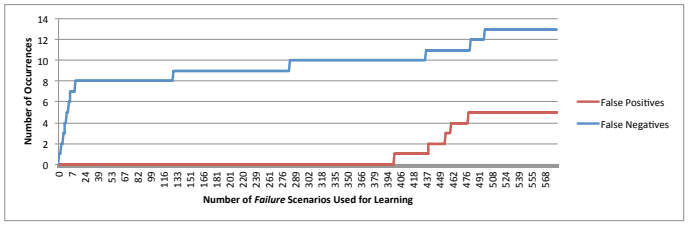
Experimental results showed that online learning algorithms can be applied to creation of models of *pre-failure patterns* with high failure prediction performance. The absence of false positives during the early stages of learning avoids unnecessary recoveries when the model still has low discriminatory power, due to the small number of learning instances available. Another observation is that false negatives are consistent for the first learned failure scenarios (up to 15 scenarios), but rare afterwards. Consequently, it is possible to train models with high prediction performance with a small number of failure scenarios available.

Ensemble algorithms outperform other algorithms, if *F-measure* is used for comparison. However, *Recall* can be a better metric when recovery is performed with a small cost and the impact of false positives on the service is small. According to our tests, containers can be migrated to other hosts in less than 1 second. Consequently, with such small overhead, the 1% reduction in *precision* in standard *Hoeffding trees* over ensemble models can be rewarded by the interpretability and efficiency (only one model is required) advantages of *Hoeffding trees*.

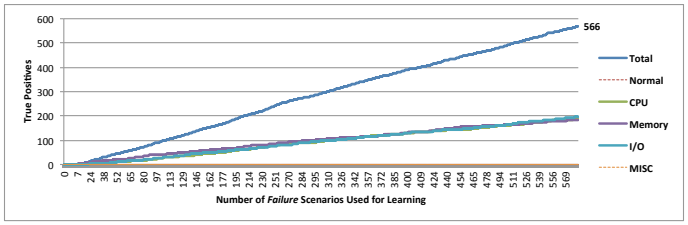
Classification of *pre-failure patterns* were shown inappropriate using our models. From the analysis of individual metric values, we observed that this phenomena is explained by the dependence of resources (exhaustion of one resource leads to exhaustion of others), which avoids observation of patterns specific to each failure type.

7. CONCLUSION

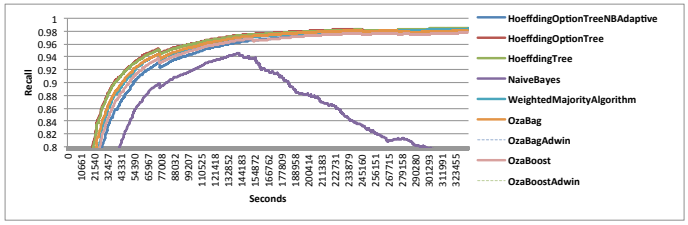
This paper presents a self-healing framework for streaming servers that uses online failure prediction as the core technique to avoid performance anomalies by means of proactive recovery using *container-based virtualization* techniques. This framework provides load admittance, failure prediction, model evaluation and failover functionalities. Evaluation results showed that performance failures can be predicted with high



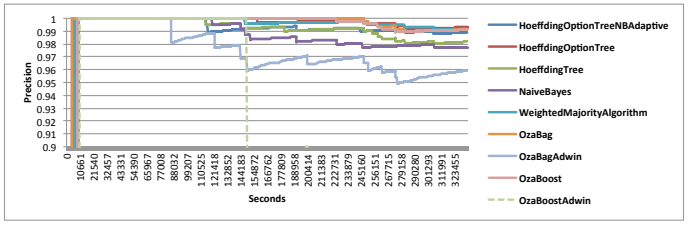
(a) Relation of the number of False Positives and False Negatives with the number of failure scenarios used for learning.



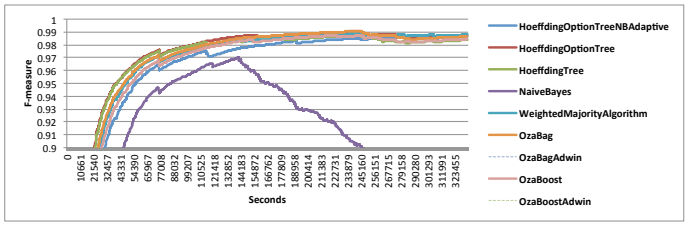
(b) Relation of the number of True Positives with the number of failure scenarios used for learning.



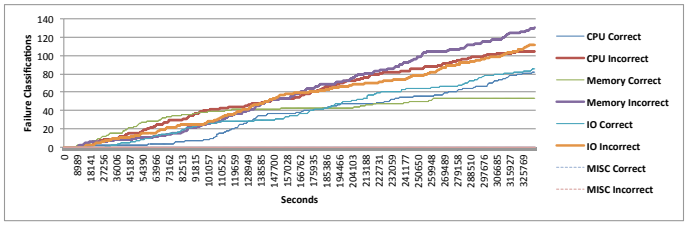
(c) Recall of each classifier



(d) Precision of each classifier



(e) Prediction F-measure of each classifier



(f) Number of failure types correctly and incorrectly classified

Figure 3: Failure prediction and classification of failure types. Slashed lines represent small values below the amplitude window shown or the zero value. Failures accounted as predicted were anticipated with a look-ahead time of at least 2 seconds.

levels of *prediction* and *recall*. Additionally, despite underperforming ensemble algorithms by a small margin, the interpretability of *Hoeffding trees* may justify their use when inexpensive recovery techniques like ours are used.

Failure classification is of the interest of failure removal activities. Ambiguous *pre-failure patterns* led to low classification performance of failure types during exhaustion of *CPU*, *Memory* and *I/O*. As future work, we will explore sequential patterns, as they can be used to model sequences of resource levels before failure, which may indicate the resource responsible for the failure.

8. ACKNOWLEDGMENTS

This work was partially supported by FCT-Portugal under grant SFRH/BD/35784/2007 and CISUC (Centre for Informatics and Systems of University of Coimbra).

9. REFERENCES

[1] H264 streaming module. http://h264.code-shop.com/trac, Apr. 2012.

[2] Hyperic's system information gatherer (sigar) api. http://sourceforge.net/projects/sigar/files/, Apr. 2012.

[3] Openvz. http://wiki.openvz.org/Main_Page, Apr. 2012.

[4] Rsync. http://everythinglinux.org/rsync/, Apr. 2012.

[5] Stress tool. http://weather.ou.edu/ apw/projects/stress/, Apr. 2012.

[6] T. Abdelzaher, K. Shin, and N. Bhatti. Performance guarantees for web server end-systems: a control-theoretical approach. *Parallel and Distributed Systems, IEEE Transactions on*, 13(1):80 –96, jan 2002.

[7] R. Agrawal, T. Imielinski, and A. Swami. Database mining: a performance perspective. *Knowledge and Data Engineering, IEEE Transactions on*, 5(6):914 –925, dec 1993.

[8] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM.

[9] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, pages 443–448, 2007.

[10] R. R. Bouckaert. Choosing between two learning algorithms based on calibrated tests. In *In ICMLâĂŹ03*, pages 51–58. Morgan Kaufmann, 2003.

[11] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[12] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni. Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 452 –461, june 2008.

[13] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: high availability via asynchronous virtual machine replication. In *Proceedings of the 5th*

USENIX Symposium on Networked Systems Design and Implementation, NSDI'08, pages 161–174, Berkeley, CA, USA, 2008. USENIX Association.

[14] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM.

[15] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, Sept. 2002.

[16] N. Feamster and H. Balakrishnan. Packet loss recovery for streaming video. In *12th International Packet Video Workshop*. PA: Pittsburgh, 2002.

[17] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In P. VitÃąnyi, editor, *Computational Learning Theory*, volume 904 of *Lecture Notes in Computer Science*, pages 23–37. Springer Berlin / Heidelberg, 1995.

[18] J. a. Gama, P. Medas, and R. Rocha. Forest trees for on-line data. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, pages 632–636, New York, NY, USA, 2004. ACM.

[19] M. Grottke, L. Li, K. Vaidyanathan, and K. Trivedi. Analysis of software aging in a web server. *Reliability, IEEE Transactions on*, 55(3):411 –420, sept. 2006.

[20] X. Gu and H. Wang. Online anomaly prediction for robust cluster systems. *Data Engineering, International Conference on*, 0:1000–1011, 2009.

[21] G. Hoffmann, K. Trivedi, and M. Malek. A best practice guide to resource forecasting for computing systems. *Reliability, IEEE Transactions on*, 56(4):615 –628, dec. 2007.

[22] A. Iyengar, E. MacNair, and T. Nguyen. An analysis of web server performance. In *Global Telecommunications Conference, 1997. GLOBECOM '97., IEEE*, volume 3, pages 1943 –1947 vol.3, nov 1997.

[23] T. Kelly. Detecting performance anomalies in global applications. In *Proceedings of the 2nd conference on Real, Large Distributed Systems - Volume 2*, WORLDS'05, pages 42–47, Berkeley, CA, USA, 2005. USENIX Association.

[24] R. Kohavi and C. Kunz. Option decision trees with majority votes. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 161–169, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[25] L. Li, K. Vaidyanathan, and K. Trivedi. An approach for estimation of software aging in a web server. In *Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n*, pages 91 – 100, 2002.

[26] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo. Bluegene/l failure analysis and prediction models. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 425 –434, june 2006.

[27] N. Littlestone and M. Warmuth. The weighted majority algorithm. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 256 –261, oct-1 nov 1989.

[28] D. Mosberger and T. Jin. httperf - a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37, Dec. 1998.

[29] N. C. Oza and S. Russell. Online bagging and boosting. In *In Artificial Intelligence and Statistics 2001*, pages 105–112. Morgan Kaufmann, 2001.

[30] P. Padala, X. Zhu, Z. Wang, S. Singhal, K. Shin, et al. Performance evaluation of virtualization technologies for server consolidation. *HP Laboratories Technical Report*, 2007.

[31] S. Pertet and P. Narasimhan. Causes of failures in web applications. Technical report, CMU Parallel Data Laboratory, 2005.

[32] B. Pfahringer, G. Holmes, and R. Kirkby. New options for hoeffding trees. In M. Orgun and J. Thornton, editors, *AI 2007: Advances in Artificial Intelligence*, volume 4830 of *Lecture Notes in Computer Science*, pages 90–99. Springer Berlin / Heidelberg, 2007.

[33] R. Powers, M. Goldszmidt, and I. Cohen. Short term performance forecasting in enterprise systems. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 801–807, New York, NY, USA, 2005. ACM.

[34] R. Puri and K. Ramchandran. Multiple description source coding using forward error correction codes. In *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference on*, volume 1, pages 342 –346 vol.1, oct. 1999.

[35] A. Robertsson, B. Wittenmark, M. Kihl, and M. Andersson. Admission control for web server systems - design and experimental evaluation. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 1, pages 531 –536 Vol.1, dec. 2004.

[36] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 426–435, New York, NY, USA, 2003. ACM.

[37] M. Saxena, U. Sharan, and S. Fahmy. Analyzing video services in web 2.0: a global perspective. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '08, pages 39–44, New York, NY, USA, 2008. ACM.

[38] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance computing systems. *Dependable and Secure Computing, IEEE Transactions on*, 7(4):337 –351, oct.-dec. 2010.

[39] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *Multimedia, IEEE*, 18(4):62 –67, april 2011.

[40] T. Stockhammer. Dynamic adaptive streaming over http –: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, MMSys '11, pages 133–144, New York, NY, USA, 2011. ACM.

[41] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory tcp: connection migration for service continuity in the internet. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 469 – 470, 2002.

[42] Y. Tan, X. Gu, and H. Wang. Adaptive system anomaly prediction for large-scale hosting infrastructures. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, PODC '10, pages 173–182, New York, NY, USA, 2010. ACM.

[43] K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection science*, 8(3-4):385–404, 1996.

[44] I. Witten, E. Frank, and M. Hall. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2011.