SHStream: Self-healing Framework for HTTP Video-Streaming

Carlos Augusto Cunha *, Luis Moura e Silva †
Centre for Informatics and Systems of University of Coimbra
* *ccunha@dei.uc.pt*, † *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.**

I. 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 [1] (e.g., the recent MPEG-DASH standard [2]). Other approaches use temporal data redundancy [3] and spatial data redundancy [4]. At server level, it is known that resource exhaustion is the main cause of performance failures in web servers [5]. It occurs as a consequence of excess of client workloads and performance anomalies. Workload-related failures can be anticipated by effective load control mechanisms [6]. 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 often web servers to provide streaming data to users [7], 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* [8] and *n-tier web applications* [9]. 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.

Recovery of streaming service failures caused by server performance anomalies presents several challenges. Firstly, they should be filtered from other types of incidents caused by network and client-side faults. Secondly, all the connections established with the faulty server should be migrated to other server without breaking TCP connections, to ensure client-side transparency of recoveries. Finally, failure detection and recovery should be efficient to avoid client-side interruption of playback.

This paper addresses these challenges by exploring failure prediction and virtualization technologies to support proactive recovery of connections. Failure prediction provides an opportunity to gracefully handle failures before potential outages occur. Prediction models identify *pre-failure patterns* of server behavior to anticipate failures, avoiding the use of expensive checkpointing techniques (e.g., *lock-stepping*) [10] to recover streaming connections. These techniques confront the real-time constraints of streaming due to their complexity and overheads [11]. Virtualization techniques are further used to migrate the server state and connections established with clients. Virtualization allows generalization of our approach, as migration of TCP connections require operating system instrumentation [12].

Previous work in prediction of performance failures address forecasting of resource exhaustion due to memory leaks[13][14][15], correlation of temporal and spacial events [16], prediction of resource values using sequential patterns [17] and context-aware prediction [18]. These techniques were designed for *batch learning* (model training is performed at once and further updates are avoided) and do not cover short-term prediction of server performance failures in services with long running connections delivering packets with strictly time constraints.

This paper presents SHStream, a self-healing framework for HTTP streaming implemented in Java, driven by prediction of server performance failures, with the following features: *failure detection*, *load admittance*, *online learning* and *evaluation* of models, *failure prediction*, and *failure recovery*. This framework was used on the evaluation of *pre-failure patterns* models for failure prediction of performance anomalies using several variants of *hoeffding trees* and *ensemble* algorithms. Experimental evaluation aims to answer the following fundamental research questions:

- How to perform online learning and evaluation of models for prediction of performance anomalies in streaming services?
- Do online learning models accurately capture *pre-failure patterns* in web servers providing streaming content?
- Which learning algorithms have the best performance?
- How many learning instances are required to stabilize prediction performance?

The rest of this paper is structured as follows. Section II presents the related work. Section III formalizes the problem. Section IV presents the algorithms used for failure prediction and diagnosis, implemented in Section V. Section VI shows results of the experimental work done to evaluate our approach. Section VII presents conclusions.

II. 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 [13] 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. [19] applied time series methods, rule-based classification and Bayesian network models to prediction of anomalous events in commercial and scientific applications. These models showed acceptable errors when evaluated with a production dataset. Hoffmann et al. [20] studied the use of several modeling techniques to predict resource consumption of the Apache Web server. Results showed that UBFs yields the best results for free physical memory prediction and SVMs performed better predicting server response times. Cherkasova et al [21] 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 [22] 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. That 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 [23]. Liang et. al. [16] addressed failure prediction in clusters of scientific applications. They explored temporal and spacial locality of previous failures to predict future failures and used 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 [17] 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. [18] 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.

Most previous research in this area focus on long latency errors (e.g., memory leaks), through analysis of the temporal distance between events [16] or the consumption of a single resource [13] during long periods of time. Our work is the first attempt to perform short-term online prediction of server performance failures in services with long running connections delivering packets with strict time constraints. We address that problem by building an integrated self-healing framework for streaming services that learns and evaluates models of *pre-failure patterns* iteratively and performs proactive recovery using migration of the server application and respective connections using virtualization techniques.

III. Problem Statement

This section describes the failure prediction and recovery problems addressed by SHStream. SHStream addresses two main types of faulty behaviors: (1) *fail-stop*, when the server stops answering to new requests and transmitting data on the established connections; and (2) *service quality degradation*, when servers put data on the network at rates below the video encoding bitrates required for playback.

Assuming that the server never exceeds its nominal capacity (ensured by load admittance), any performance failure is imputed to server anomalies. These anomalies can be caused by hardware and software faults manifested by server abnormal behaviors. Since the resultant errors can be covered by our metrics (e.g., abnormal CPU utilization for the current client workloads), they can be potentially predicted by our models. However, we only recover connections when the error occurs outside the server application process, as we need to copy the last application state during failover to the target host.

A. Failure Prediction

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 to 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 $n > 0$.

Several challenges arise when modeling server states that precede failures (i.e., *pre-failure patterns*). 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.

B. Recovery

SHStream implements two recovery techniques: *connection redirection* and *server failover*. *Connection redirection* is implemented by the HTTP protocol through the REDIRECT command. REDIRECT commands can be issued by the server when clients request the establishment of a new connections. The server then answers with the network address of the target host, which will be used by the client to reissue the connection. New connection requests can be redirected to another host when the server reaches its capacity or is experiencing performance problems. *Server failover* migrates the server application to another host. This technique presents two main challenges: (1) synchronization of application states between the faulty and failover servers; and (2) server migration transparently to clients, requiring moving the server IP address and TCP connection states to the target host to avoid breaking the current sessions established with players.

IV. Learning Algorithms

Online learning algorithms overcomes traditional batch learning limitations to fulfill the online learning requirements of dynamic systems [24]: (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*.

A. Hoeffding trees

Decision trees are powerful, interpretable and efficient classifiers - with n examples and m attributes, the average cost of basic decision tree induction is $O(mn \log n)$. *Hoeffding decision trees* promise performance levels similar to batch decision trees (e.g., C4.5 [25]). VFDT *(Very Fast Decision Tree)* is a state of the art algorithm for creating Hoeffding trees proposed by Domingos and Hulten [26]. VFDT builds the tree iteratively by splitting each node when the number of learned examples satisfies the *Hoeffding Bound* (1).

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

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.

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

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. 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 [27]. *Adaptive Hoeffding Trees* is another variant of *Hoeffding trees* explored that uses ADWIN [28] to evolve trees with new server behaviors, by monitoring performance of branches on the tree and replace them with new branches with higher accuracy.

B. 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 [29]. We assess four ensemble algorithms in our experiments that uses *Hoeffding Trees* as base models: *Weighted Majority Algorithm* [30], *OzaBoost* [31], *OzaBag* [31] and *Option Trees* [32].

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 $O(\log |A| + n)$.

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* [33] 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.

3) OzaBag: Learning from a bootstrap replicate of examples drawn randomly from the training dataset according to the $Poisson(1)$ distribution, similarly to *Bagging* [34] 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) 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 [35].

V. 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** [36]: 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** [37]: for migration of virtual containers.

The SHStream framework runs within the server's host to ensure: (1) *Scalability* by avoiding centralized data gathering and analysis; (2) *Timeliness* by enabling data gathering and server control with minimum communication delays; and (3) *Integration* through direct control 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 are used for load admittance, failure prediction 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* [38] and *mod_SHS*. The *H264 Streaming Module* ensures advanced features to streaming users - e.g. *time shifting seek*. *mod_SHS* gathers application-level metrics data and redirects new connections to another server when the *load admittance* component determines 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.

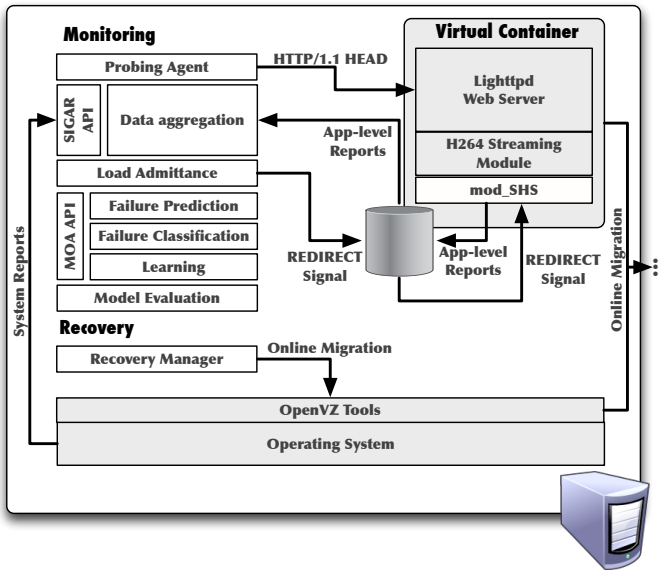


Fig. 1: SHStream Architecture

A. Data Gathering

SHStream collects the 44 system metrics provided by SIGAR, covering *CPU*, *Memory*, *I/O* and *Network* resources, as well process-related metrics (e.g., *CPU* and *memory* consumed by the web server process). The complete list of metrics can be found at [36]. 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 connections recovered.*

B. Failure Detection

As described in Section III, our approach covers both *server fail-stop* failures and *service quality degradation*. *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. On the other hand, *service quality degradation* are determined by a service global condition and a connection-level condition. The service global condition determines whether at any time t_i, the sum of transmitted bitrates for all active connections, in average, 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 data to the network than the data required by players to play videos up to time t_i.

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

The service global condition identifies degraded service states, but it unable to provide information about the number of requests suffering from service degradation. The connection-level condition determines, at any time t_i, for each connection,

if the transmitted bitrate is below the respective video encoding bitrate, as defined in (4):

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

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. When that value is smaller than the video encoding bitrate, the player stops video playback until enough data arrives.

C. Load Admittance

Admittance of new connections is allowed when the bandwidth in excess being used by the actual connections being served is sufficient to afford them. 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 (payload data) 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.

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

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.

This simple load admittance rule has been shown effective in our experiments to control server load. More elaborated admission schemes can be explored but this problem is out of scope of the main research topic of this paper.

D. Failure Prediction

The main research problems addressing prediction in SHStream address online iterative learning, evaluation of models and classification of *pre-failure patterns* on logged monitoring data instances. The 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.

Algorithm 1 shows how models are handled for prediction, learning and evaluation. One monitoring instance containing 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. 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, they are learned as *normal*.

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

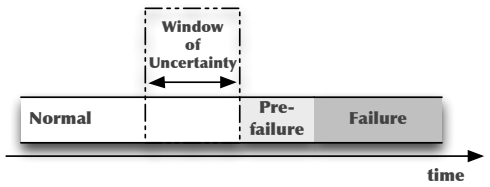
E. Model Evaluation Metrics

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).

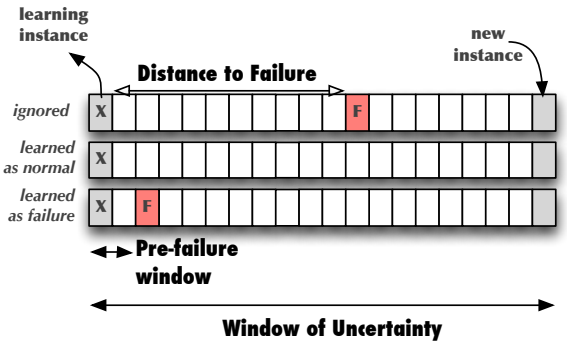
$$\text{Recall} = \frac{TP}{TP + FN} \qquad \text{Precision} = \frac{TP}{TP + FP} \quad (6)$$

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.



(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

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

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

F. Recovery

Virtualization technology is explored by SHStream to isolate the web server application 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.

We adopted OpenVZ, a OS-level server virtualization technology based on containers. *Container-based virtualization* allows: (1) significantly smaller overheads than typical virtualization and paravirtualization technologies [39] because they run on top of the operating system; (2) server application isolation from the rest of the system, avoiding migration of errors originated outside the container during recovery; (3) smaller migration downtimes, as the volume of data 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* [40]) 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.

VI. 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 I.

A. 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 machines running a script that coordinates execution of *httperf* [41] instances to avoid client-side overloading. The script also commands server fault injection through *ssh commands* that invokes the *Stress* tool [42] 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.

B. 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.

C. Results

We ran experiments one first time without SHStream to determine its overhead. The maximum number of connections per second is the same when SHStream is running and when it is not running. 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 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. This is a fair assumption, given that recovery can be performed in less than 1 second (as observed for container migration). 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. 4), followed by the *HoeffdingOptionTree*, *OzaBag* and *OzaBoost*.

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

VII. 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 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.

Our research work focused on failure prediction, despite providing solutions for load admission and recovery problems. Research on these two last topics will be developed in future work.

Acknowledgment

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).

References

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

[2] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *Multimedia, IEEE*, vol. 18, no. 4, pp. 62 –67, april 2011.

[3] N. Feamster and H. Balakrishnan, "Packet loss recovery for streaming video," in *12th International Packet Video Workshop*. Pittsburgh, 2002.

[4] 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*, vol. 1, oct. 1999, pp. 342 –346 vol.1.

[5] S. Pertet and P. Narasimhan, "Causes of failures in web applications," CMU Parallel Data Laboratory, Tech. Rep., 2005.

[6] 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*, vol. 1, dec. 2004, pp. 531 –536 Vol.1.

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

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

[9] T. Abdelzaher and K. Shin, "Performance guarantees for web server end-systems: a control-theoretical approach," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 1, pp. 80 –96, jan 2002.

[10] 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.*, vol. 34, no. 3, pp. 375–408, Sep. 2002.

[11] 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*, ser. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 161–174.

[12] 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*, 2002, pp. 469 – 470.

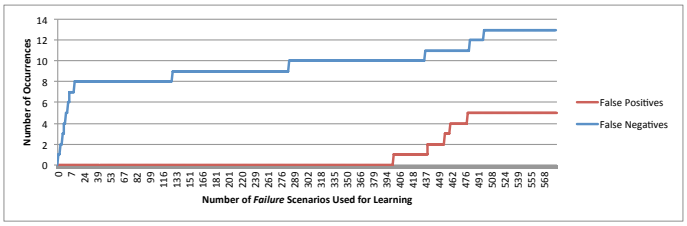
[13] R. Powers, M. Goldszmidt, and I. Cohen, "Short term performance forecasting in enterprise systems," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, ser. KDD '05. New York, NY, USA: ACM, 2005, pp. 801–807.

[14] L. Li, K. Vaidyanathan, and K. Trivedi, "An approach for estimation of software aging in a web server," in *Empirical Software Engineering. Proceedings of the International Symposium on*, 2002, pp. 91 – 100.

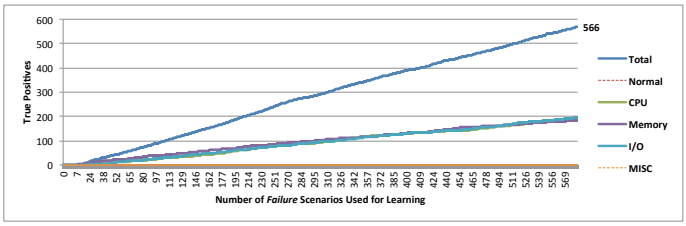
[15] M. Grottke, L. Li, K. Vaidyanathan, and K. Trivedi, "Analysis of software aging in a web server," *Reliability, IEEE Transactions on*, vol. 55, no. 3, pp. 411 –420, sept. 2006.

[16] 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*, june 2006, pp. 425 –434.

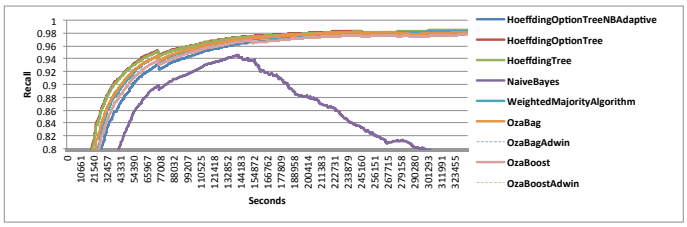
[17] X. Gu and H. Wang, "Online anomaly prediction for robust cluster systems," *Data Engineering, International Conference on*, vol. 0, pp. 1000–1011, 2009.



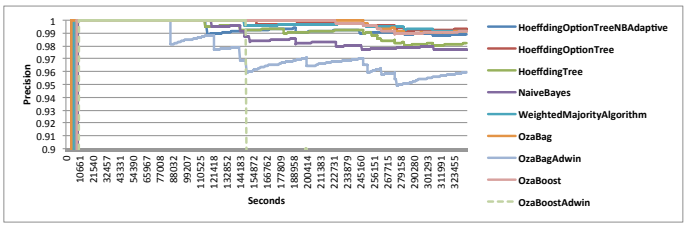
(a) Relation of the number of False Positives and False Negatives with the number of failure scenarios used for learning.



(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

Fig. 3: Failure prediction performance. 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.

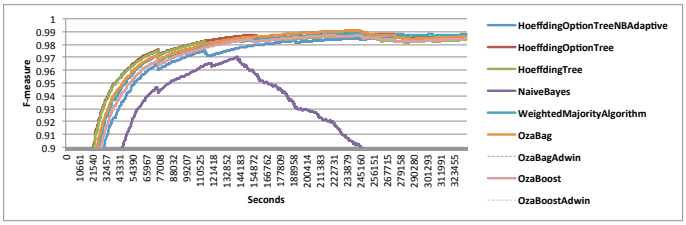


Fig. 4: Prediction F-measure of each model.

[18] 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*, ser. PODC '10. New York, NY, USA: ACM, 2010, pp. 173–182.

[19] 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 *Proc. of the ninth ACM SIGKDD international conference on knowledge discovery and data mining*, ser. KDD '03. NY, USA: ACM, 2003, pp. 426–435.

[20] G. Hoffmann, K. Trivedi, and M. Malek, "A best practice guide to resource forecasting for computing systems," *Reliability, IEEE Transactions on*, vol. 56, no. 4, pp. 615 –628, dec. 2007.

[21] 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*, june 2008, pp. 452 –461.

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

[23] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 337 –351, oct.-dec. 2010.

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

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

[26] 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*, ser. KDD '00. New York, NY, USA: ACM, 2000, pp. 71–80.

[27] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: a performance perspective," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 5, no. 6, pp. 914 –925, dec 1993.

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

[29] K. Tumer and J. Ghosh, "Error correlation and error reduction in ensemble classifiers," *Connection science*, vol. 8, no. 3-4, pp. 385–404, 1996.

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

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

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

[33] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Computational Learning Theory*, ser. Lecture Notes in Computer Science, P. Vitnyi, Ed. Springer Berlin / Heidelberg, 1995, vol. 904, pp. 23–37.

[34] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

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

[36] (2012, Apr.) Hyperic's system information gatherer (sigar) api. http://sourceforge.net/projects/sigar/files/.

[37] (2012, Apr.) Openvz. http://wiki.openvz.org/Main_Page.

[38] (2012, Apr.) H264 streaming module. http://h264.code-shop.com/trac.

[39] 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.

[40] (2012, Apr.) Rsync. http://everythinglinux.org/rsync/.

[41] D. Mosberger and T. Jin, "httperf - a tool for measuring web server performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 31–37, Dec. 1998.

[42] (2012, Apr.) Stress tool. http://weather.ou.edu/ apw/projects/stress/.