Using Syslog Message Sequences for Predicting Disk Failures

R. Wesley Featherstun and Errin W. Fulp
Department of Computer Science
Wake Forest University

August 24, 2010

Abstract

Mitigating the impact of computer failure is possible if accurate failure predictions are provided. Resources, and services can be scheduled around predicted failure and limit the impact. Such strategies are especially important for multi-computer systems, such as compute clusters, that experience a higher rate of failure due to the large number of components. However providing accurate predictions with sufficient lead time remains a challenging problem.

This research uses a new spectrum-kernel Support Vector Machine (SVM) approach to predict failure events based on system log files. These files contain messages that represent a change of system state. While a single message in the file may not be sufficient for predicting failure, a sequence or pattern of messages may be. This approach uses a sliding window (sub-sequence) of messages to predict the likelihood of failure. Then, a frequency representation of the message sub-sequences observed are used as input to the SVM. The SVM associates the messages to a class of failed or non-failed system. Experimental results using actual system log files from a Linux-based compute cluster indicate the proposed spectrum-kernel SVM approach can predict hard disk failure with an accuracy of 80% about one day in advance.

1 Introduction

Clusters are quickly growing in size in terms of both computing power and storage space. It is predicted that by 2018, large systems could have over 800,000 disks. Out of these 800,000 disks, it is possible that 300 of them may be in a failure state at any given time [13]. Since multicore processors are becoming more prevalent, even one disk being unavailable means that multiple processors may be unable to perform their work. Assuming one could predict these failure events, the distribution of work on the cluster could be altered to avoid effected disks before they failed or jobs could be paused while the necessary data is backed up.

Much work has been done in the field of hardware failure predictions. Hammerly *et al.* [5] used a naive Bayesian classifier on SMART data and managed to predict disk failures that would occur in the next 48 hours with 52% accuracy. A team from IBM [8] used data from a specialized logging system on its BlueGene cluster. While the team achieved high accuracy, its data set may be too specialized to be of use by the general public. Peter Broadwell [2] used a supervised Bayesian approach to predict SCSI cable failures. While he was able to create an effective prediction method, the approach presented in the paper is not scalable. Murray *et al* [11] compared the effectiveness of data mining techniques such as SVMs, clustering, and a rank-sum test for failure predictions using SMART data. Finally, Turnbull *et al* [16] proposed an approach similar to the one presented in this paper. However, Turnbull focused on predicting system board failures instead of disk failures.

The prediction process in this paper uses the `syslog` event logging service as data to predict failure events. The `syslog` facility is common to all Linux distributions as well as Unix variants. Using blocks

of `syslog` entry tag and message strings, a Support Vector Machine [3] creates a model of the data which isolates patterns of log information that indicate future disk failures. The main focus of this approach is to predict disk failures at least one day before the failure. One day's notice gives administrators enough time to make any necessary changes to the scheduling process or ensure that they can obtain another hard drive of the correct model [4].

The remainder of this paper is organized as follows. Section 2 provides a description of the Unix `syslog` facility and SMART messages. Section 3 describes the approach to failure predictions proposed in this paper. Section 4 describes the Support Vector Machine data mining technique while section 5 discusses experimental results. Finally, section 6 summarizes this paper and discusses some areas for future work.

2 System Log Facilities and Messages

`Syslog` is a standard Unix logging facility, which means that every computer running Linux is able to use `syslog` [9]. The ubiquity of `syslog` means that performing an analysis on `syslog` data allows for the creation of a failure prediction approach which can be used by anyone using a Linux or Unix system. `Syslog` records any change of system state, such as a login or a program failure.

As seen in Table 1, the standard `syslog` message contains six fields [9]. However, the approach in this paper uses only the timestamp, tag number, and message fields. The tag is a numerical representation of the importance of the message. The tag number is an integer, where a lower number indicates a higher importance. For example, a message with a tag number of 1 is more urgent than a tag number of 20. The tag number field corresponds to the priority field in the actual `syslog` packet, whose value is determined by multiplying the facility by 8 and then adding the level. Therefore, it provides a numerical representation of both the facility which posted the message and how important the message is. The time field records the time at which the message was posted, commonly in Linux epoch time. The final field is the message field, which consists of a plain text string of varying length. The message is an explicit description of the associated event. While the other fields only indicate how important the event was and when it took place, the message field tells an observer that the event was, for example, a login attempt or a disk failure [9].

SMART messages record and report information that relates solely to hard disks, such as their current health and performance and is deployed with most modern ATA and SCSI drives [1]. Since SMART disks monitor health and performance information, they are able to report and possibly predict hard drive problems. Some of the attributes monitored by SMART are the current temperature, the number of scan errors, and the number of hours for which a disk has been online. SMART checks the disk's status every 30 minutes and passes along any information regarding the possibility of an upcoming failure to `syslog`. Pinheiro *et al.* have shown that using individual SMART messages to build a prediction model is ineffective [12]. Therefore, the approach in this paper uses all `syslog` data, including, but not limited to, SMART data.

3 Approach

3.1 Sequential Data

As described by Pinheiro *et al.*, single messages are not sufficient for predicting failure [12]. However, examining sequences of messages may be a more effective means of failure predictions. Instead of considering messages in isolation, the approach in this paper analyzes sequences of messages, which provides context for individual messages.

A sliding window approach is used to isolate sequential data. In this method, a window of fixed length, n, is placed at the beginning of the list of data. All of the data that fall in that window is considered to be one sequence. Then, the sliding window is moved forward one item and the next n items are made into a sequence.

The type of information being examined alters how

Host	Facility	Level	Tag	Time	Message
node226	daemon	info	30	1205054912	ntpd 2555 synchronized to 198.129.149.215, stratum 3
node226	local4	info	166	1205124722	xinetd 2221 START: auth pid=23899 from=130.20.248.51
node165	local3	notice	157	1205308925	OSLevel Linux m165 2.6.9-42.3sp.JD4smp
node165	syslog	info	46	1205308925	syslogd restart.

Table 1: Example entries from a `syslog` file

the window moves across message boundaries. When classifying based on tag numbers, each tag number represents one message. Therefore, the sliding window indicates the criticality of the last n messages. However, this paper also examines the use of keystrings to predict disk failures. In the case of keystrings, there may be zero, one, or more keystrings in a given message. Since the keystrings are arranged by order of appearance in the logs, a given window can provide context either within a single message or two or more messages.

The spectrum kernel technique was devised by Leslie *et al.* [7] to leverage sequences of data for use with a classifier. For any $k \geq 1$, the k-spectrum of a given input sequence is defined as all of the subsequences of length k that the sequence contains. Given a sequence length k, an alphabet size b and a single member of the alphabet, e, the spectrum kernel representation of a given sequence can be obtained using Equation 1 [15]. The equation must be applied for each letter in the input.

$$f(t) = mod(b * f(t-1), b^k) + e \tag{1}$$

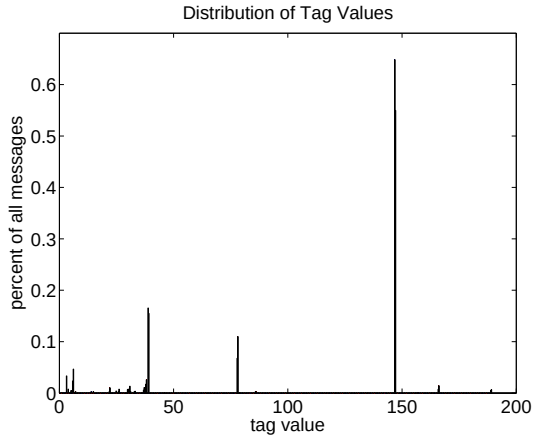
3.2 Tag-Based Features

Consider the tag numbers that occur within a message window. The order in which these messages appear forms a list of tag numbers. From this list of tag numbers, one can create a feature vector which combines two types of features: a count of the number of times each tag number and sequence number appears in a window. For example, the sequence of tag numbers shown in Table 2 correspond to a tag count vector of {40:1, 88:1, 148:2, 158:3, 188:3}.

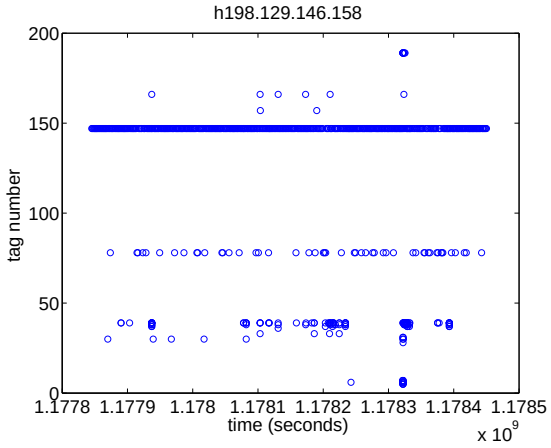
At first, the size of the alphabet is the number of unique tag numbers in `syslog`, which is 191. However, as the maximum tag number seen in the experimental data set is 189, the alphabet size is considered to be 189. Using sequences of length 5, the list of possible features is 189^5, which equates to over 241 billion unique combinations. Computing sequence numbers for all of these combinations will take an excessively long time. Therefore, the alphabet is reduced by assigning multiple tag numbers to a number of 0, 1, or 2 based on the tag's criticality [4].

Tag numbers which are less than or equal to a 10 are considered to be high priority. Tag numbers between 11 and 140 are considered medium priority and tag numbers above 140 are considered low priority. The size of the reduced alphabet and cutoff values are determined by examining the distribution of tag numbers as seen in Figure 2(a). Using an alphabet of size 3 reduces the possible number of features to 243.

Table 2 illustrates the process of assigning criticality scores to each tag number and then determining sequence numbers where $k = 5$. The left hand column contains a list of tag numbers. The middle column shows the sequence of the most recent 5 criticality scores, which are obtained by using the sliding window method and criticality cutoff values described earlier. The criticality score of the current tag number is placed on the righthand side of the criticality sequence. Finally, the righthand column shows the sequence number for the current sequence of criticality scores. The sequence number is determined using Equation 1. While intermediate sequence numbers are calculated for the first $k-1$ sequences, sequence numbers are not recorded until a full k-length sequence has passed. In this example, sequence numbers are only recorded starting at the fifth tag number.



(a) A histogram which indicates the percentages of messages in the data set which contained a given tag number. For example, about 60% of all messages in the data set had a tag number of 149.



(b) An example of the tag number distribution on a given host across time. Each circle represents the tag number for a single `syslog` message.

Figure 1: Illustrations of tag number distribution in the experimental data set

Tag	Translated	Sequence Number
148	2	
148	22	
158	222	
40	2221	
158	22212	239
188	22122	233
188	21222	215
88	12221	160
158	22212	239
188	22122	233

Table 2: An example of a tag list being translated into sequences of criticality scores and then assigned sequence numbers using these criticality scores. For this example, $k = 5$.

3.3 Tags With Timing Information

Timing information is another feature that may help improve failure predictions. The purpose of examining timing information is to discern whether or not a change in message rate can be used to predict failures. During the creation of the sequence numbers, the difference between time of the first message in the sequence and the time of the last message in the sequence is recorded. Doing so provides an indication of how quickly or how slowly those messages were posted. The differences in time are recorded in the same format as the tag and sequence numbers. This information has not been considered in previous work using this method [4].

3.4 Keystring-based Features

The myriad possible `syslog` configurations allow for a set up in which tag numbers are not present [9]. One thing an administrator is unlikely to remove is the message field itself. A string is defined as any space-delineated collection of characters in the message field. For example, a string can be an English word, an IP address, or a number. Tag numbers are not factored into this approach. The goal of this method is to discover some pattern of actual strings which will allow for failure prediction.

Unfortunately, the list of possible strings can be quite large. For example, the `syslog` data set used in

this paper contains over 2 billion unique strings, despite the fact that the English language only consists of about 1 million words [14]. Consider a message which posts the temperature of a disk drive. Even if the temperature only fluctuated by ten degrees across all log files, those ten values (assuming the message only posts integer values) are assigned unique identifiers in the alphabet. This alphabet results in a feature space of over 8×10^{27} when $k = 3$.

In an effort to reduce the number of strings, it is possible to isolate only the strings that the SVM finds useful in creating a model. To do this, the SVM is trained on the entire data set, using only the number of times each string appears. Once the SVM builds a model of the data, the feature space is examined to determine the most important strings. Any string which the classifier finds useful will henceforth be referred to as a keystring.

Now that there is a list of the most important strings, these strings are used to create the message list. A count of each string is used as well as a count of each sequence of strings. When building a sequence number, message boundaries are ignored. For example, if keystrings 0 and 29 are in one message, keystring 10 in the next message, and keystring 1 in the third message, the keystring sequence when $k = 4$ is {0, 29, 10, 1}.

Many keystrings may represent similar items. For example, each computer may have a unique ID number. While each of these keystrings is unique, they all fall under the general label of an ID number. In the interest of reducing the alphabet further, keystrings are grouped into general types, such as computer ID number, and a number is assigned to each type.

Timing information can also be included when using the keystring approach. As with the tag approach, a count is taken of the differences between the time at which the first message in a sequence is posted and that of the final message in a sequence.

4 Support Vector Machines

Each disk can be separated into one of two classes: a disk which failed or a disk which did not fail. The research in this paper uses an SVM to build a model of the two classes based on past `syslog` events.

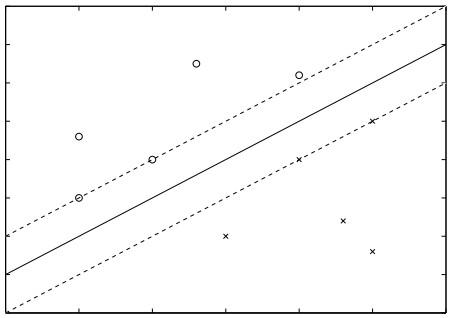


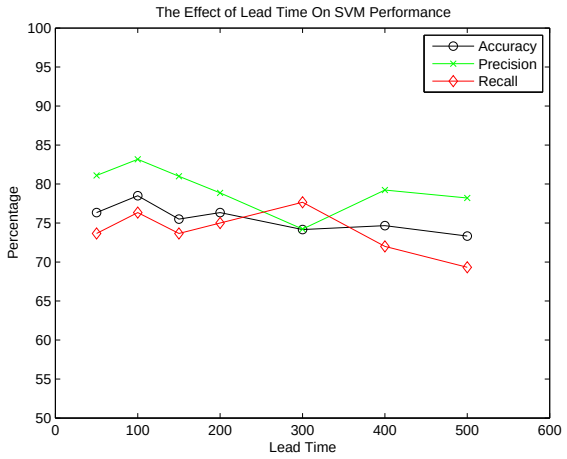
Figure 2: An illustration of the optimal 2-D hyperplane

An SVM is a classification method that takes a set of labeled training examples. Each training example is labeled to indicate which class the example belongs to. Since an SVM is a binary classifier [6], each training example must be in one of two, and only two, classes. The binary nature of the SVM makes it an ideal choice for predicting disk failures, as each example must either fail within the given window or not fail within the given window.

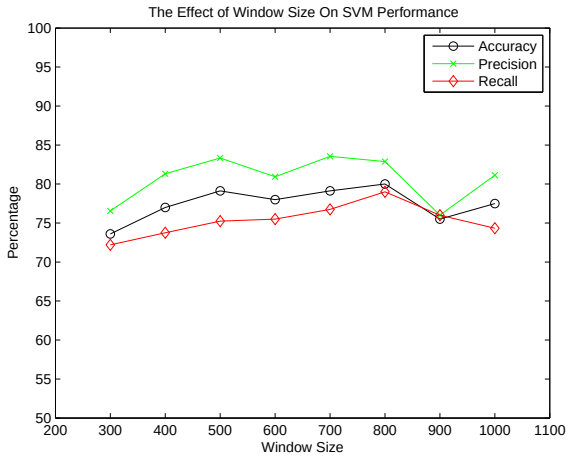
4.1 Optimal Hyperplane

Using an input set of labeled data points, the SVM attempts to find an optimal hyperplane to separate the data. Consider Figure 1, which provides an illustration of the optimal hyperplane between the class of circles and the class of crosses. The optimal hyperplane is the plane which maximizes the distance between the two classes. In the case of the figure, the optimal hyperplane is represented by the solid line.

To calculate the optimal hyperplane, one finds the planes that separate the data which are located closest to each class. In Figure 1, these hyperplanes are represented by dashed lines. Since these hyperplanes are defined by the points closest to the optimal hyperplane, only these few examples are needed to calculate the optimal hyperplane. These data points are called *support vectors*.



(a) The accuracy, precision and recall of tag-based methods as lead time before an event changes while window size remains constant



(b) The accuracy, precision, and recall of tag-based methods as the window size changes while lead time remains constant

Figure 3: The change in performance metrics as window size and lead time vary

5 Experimental Results

The `syslog` data used in these experiments is from a 1024 node Linux-based cluster managed by the Pacific Northwest National Laboratory. Each system contained multiple processors and disks. There were an average of 3.24 messages per machine per hour, which results in about 78 messages per system per day. There were 61 unique tag values, the distribution of which is shown in Figure 2(a). There were over 120 disk failures during the 24 months over which the data was collected.

To train the SVM, blocks of `syslog` messages from each system must first be isolated. The size of the message window specifies the number of messages to isolate prior to a failure. For example, if the window size is 500, then the 500 messages immediately preceding the failure message are isolated. However, if there are not enough messages before the failure, then that window of messages is not used. If a given failure comes within twenty four hours of a previous failure, then the failure is removed from the data set to keep any patterns or events which lead to the first failure from affecting predictions for subsequent failures. The removal of these failures result in 100 useable disk failures. If there are no failures on a given system, then a random window of 500 sequential messages is chosen.

Once all of the message windows are created, they have to be trimmed. To simulate lead time before a failure, a specified number of messages at the end of the window is removed. By deleting messages at the end of the window, there is a gap between the end of the message list and the event to be predicted. As an example, consider a window size of 1,200 messages. If the window is then trimmed by 200 messages, there are 1,000 messages left to classify on. By eliminating the last 200 messages, there is a gap of a little over two days between the final message in the block and the failure or non-failure.

All experiments are performed using hold out and 10-fold cross validation. Hold out means that for both the training and testing stages, an equal number of failure and non-failure examples are in the data set [17]. When using 10-fold cross validation, the data is broken up into ten sets of equal size. The classifier is

then trained on nine of these sets and tested on the final set. A different test set is then chosen from the ten groups, while the previous test set is added to the training set, so each of the ten slices eventually is used for testing [17].

The experiments performed in this paper use three metrics to determine the effectiveness of a model: accuracy, precision and recall [10]. Accuracy is the total number of predictions which the model made correctly. Precision is the true positive rate, which indicates the number of disks which were predicted to fail within the given window that actually did fail within that window. Finally, recall is the percentage of actual disk failures that the model successfully predicted.

5.1 Optimal Lead Time and Window Size

The following experiment uses tag sequences of length 5 and a window size of 1,200 messages [4]. The amount of lead time is varied to examine whether or not attempting to predict a failure closer to the failure event improves classification performance.

Figure 3(a) shows the accuracy, precision, and recall of varying the lead time for predictions. The x-axis indicates the lead time in number of messages before a failure event. A failure event occurs where $x = 0$. Each experiment uses a fixed window size of 1,200 messages; therefore, a smaller lead time means that the number of messages used to classify increases, while the time between the final message in the block and the failure event decreases. All three metrics peak with a lead time of 100 messages, which translates to a little over one day.

The recall dips as lead time increases beyond 300 messages due to the widening gap between the end of the window and the failure event. By adding more lead time before a failure, fewer of the messages and patterns which lead up to a disk failure may be present. While some disks might operate in a reduced state for a few days before failure, some start showing signs only a few hours or a day before the failure. By increasing lead time, the model is unable to predict the failure of disks which only provide warning signs closer to the failure event, as those events are no longer in the training or test set.

5.1.1 The Effect of Window Size Using Tag-Based Features

Figure 3(b) illustrates the effect of increasing the window size. Since the results of the previous experiment suggest that a lead time of 100 messages is the most effective, this experiment also uses a lead time of 100 messages. All three metrics increase until the window size hits 800 messages. After a window size of 800 messages, just like the previous experiment, recall begins declining. Recall declines because, as the window size increases, the SVM must classify using more and more information. The increased information can make the two classes begin to look similar. In the case of disk failures, the disks only produce warning signs for a certain period of time. Before these warning signs appear, they operate as normal disks. By adding information from before the disks start to fail, that disk acts more like a working disk than a failing disk.

5.2 Tag-Based Features Without Timing Information

The previous experiments all use sequences of length 5. This experiment varies the sequence length between sequences of length 3 and sequences of length 8. While increasing the sequence length may increase the effectiveness of the model, the increase will also exponentially increase the feature space. As such, the time required to train and classify the data will increase. Therefore, a balance must be struck between the effectiveness of the model and the time required to train. Table 3 compares the accuracy, precision, and recall of training using each sequence length. All experiments used a window size of 800 messages and a lead time of 100 messages. The recall is maximized using sequences of length 6. On the other hand, the precision jumps to 85% at a sequence length of 7. The recall plummets using sequences of length 8. Henceforth, sequences of length 5 are used because they provide fewer false positives compared to a sequence of length 6 while achieving similar recall while using a

smaller feature space than either sequences of length 6 or of length 7.

Sequence Length	Accuracy	Precision	Recall
3	73.166	74.9003	75.0011
4	75.6666	80.8341	72.6681
5	79.9993	82.8838	79.0012
6	79.4994	80.5503	80.6674
7	80.999	85.4837	78.668
8	78.4992	85.7335	73.3339

Table 3: A comparison of sequence lengths when using tag-based features

5.3 Tag-Based Features With Timing Information

The performance of classification using timing information is compared to the performance without timing information in Tables 4 and 5. In neither case did the addition of time differentials significantly increase any of the three metrics. In the case of length 5 sequences, the recall actually gets substantially worse. When using sequences of length 7, the results with and without time are almost identical, as seen in Table 5. In both cases, the recall may dip because the message logging rate of nodes on a which a failure is going to occur within the next 100 messages is similar to the message logging rate of nodes which are not predicted to fail. Since the two rates are similar, the inclusion of timing information makes the two classes look more similar than when no timing information is included. Therefore, the addition of timing information not only does not provide improvement over tag sequences without timing information, but it also increases the feature space. As a result, tag based features are best when used without timing information.

Feature Space	Accuracy	Precision	Recall
Sequences Using Tags	79.9993	82.8838	79.0012
Sequences Using Tags and Time	77.8329	82.2338	71.667

Table 4: Comparing performance between features using only tags and features including time information using sequences of length 5

Feature Space	Accuracy	Precision	Recall
Sequences Using Tags	80.999	85.4837	78.668
Sequences Using Tags and Time	81.1661	86.9337	76.005

Table 5: Comparing performance between features using only tags and features including time information using sequences of length 7

5.4 Keystring Based Features Without Timing Information

The initial dictionary contains 54-keystrings. However, 25 of the strings in this dictionary are the names of nodes on the cluster. To see whether or not the SVM is learning what nodes tend to fail instead of actual patterns which lead to failures, another dictionary is tested. The second dictionary, made up of 24-keystrings, assigns all keystrings of a given type to a single number. For example, all node names are assigned a 0 and all number strings are assigned a 23. In the 24-keystring dictionary, all node names, even those not in the original 54-keystring dictionary, are included. The results of these experiments using a window size of 800 messages, lead time of 100 messages and sequences of length 3 are recorded in Table 6. The fact that the 54-keystring and 24-keystring dictionaries perform similarly well suggests that the SVM is not training on specific node names. Instead, the SVM is learning that the appearance of any node name is useful for predicting failures. The 24-keystring dictionary has the benefit of reducing the alphabet size dramatically when compared to the 54-keystring dictionary. As a result, all keystring experiments henceforth use the 24-keystring dictionary.

Dictionary	Accuracy	Precision	Recall
54	77.6661	81.8171	76.0008
24	77.6659	79.1004	78.6676

Table 6: A comparison of keystring dictionaries

Table 7 shows the change in performance as the sequence length increases when using the 24-keystring dictionary. Sequence of length 4 perform significantly better across the board than those of length 3. While length 5 sequences perform slightly better than those

of length 4, the improvement is not significant with respect to recall, although the false positive rate dips slightly. Since length 5 sequences significantly increase the feature space with marginal benefit, the 24-keystring dictionary performs best when using sequences of length 4.

Sequence Length	Accuracy	Precision	Recall
3	77.6659	79.1004	78.6676
4	79.4996	82.9838	80.6676
5	82.1428	85.0008	80.9543

Table 7: Performance of the 24-keystring dictionary as sequence length increases

5.5 Keystring Based Features With Timing Information

Despite the ineffectiveness of combining timing information with tag sequences, the usefulness of timing with regards to the keystring based approach is tested. Table 8 compares the performance of the 24-keystring approach both with and without timing information. The sequence length used for both experiments is 4. The accuracy and recall values of both approaches are essentially the same. However, when using time information, there is a slightly lower false positive rate. Since a lower false positive rate means fewer instances when a node goes into a preemptive maintenance stage, minimizing the false positives is a worthy goal. While the feature space increases, a system administrator may be willing to endure the longer training and classification time if it results in fewer false positives. Thus, time information keystring sequences are added to the final experiment.

Experiment	Accuracy	Precision	Recall
Without Time Info	79.4996	82.9838	80.6676
With Time Info	80.1657	85.567	78.6679

Table 8: A comparison of the 24-keystring dictionary with and without the addition of time information

5.6 Combination Results

Classifying works almost identically well when using tag number sequences of length 5 as when using keystring sequences of length 4. The use of tag number sequences achieves a slightly higher true positive rate while keeping a similar accuracy and recall. If the tag-based approach and the keystring-based approach are learning on different patterns, then perhaps combining the two approaches will result in better classifications.

The window size for this experiment is 800 messages and the lead time is 100 messages. Tag sequences of length 5 are used, while keystring sequences are 4 keystrings long. Since the addition of temporal features is useful with keystring based features, time differences are calculated for sequences of length 4.

Approach	Accuracy	Precision	Recall
Tags Without Time	80.999	85.4837	78.668
Keystrings With Time	80.1657	85.567	78.6679
Combination Without Time	77.9995	82.317	74.334
Combination With Time	80.6664	88.567	74.6673

Table 9: A comparison of tag based, keystring based, and combination methods

Table 9 provides a comparison among the best performing tag based approach, the best keystring approach, and a combination approach both with and without time information. Neither a combination of tag and keystring features with or without additional timing information offers any substantial increase in accuracy and both see a dip in recall, which means the combination model predicts fewer of the failures that occur. The recall dips because the message rate does not provide a good indicator of a failure. As a result, the inclusion of time information makes the two classes look more similar. However, this increased similarity results in higher precision. The precision increases because disks which were predicted to fail with low confidence when omitting timing information are now predicted to continue working. Therefore, only disks that have a high confidence score for failure are still predicted to fail. With the combination of approaches and time information, the decrease in the number of failures predicted is balanced by

an increased true positive rate, meaning that almost 89% of the disk failures predicted by this approach do fail within the next 30 hours. In fact, this model has the highest true positive rate of any of the experiments, which means that the combination approach in conjunction with timing information provides a useful improvement over other models. Whether or not it is the best model depends on whether a higher recall rate or fewer false positives is the most desired trait in a given situation.

6 Conclusions and Future Work

To determine the overall best method, this section considers the true positive rate as well as the recall. In addition, this section proposes an event logging system which requires less storage space than the current `syslog` utility.

If a high recall is the more important goal, the best approach is to use either tag sequences without timing information or keystring sequences using the 24-keystring dictionary with timing information. Both approaches hit almost 80% recall. In addition, both had very few misclassified failures. If a high true positive rate is the most desired classification trait for a given situation, then there is only one choice: combining tag sequences with keystring sequences and timing information, as this approach has a true positive rate of 89% while still predicting 75% of disk failures.

The PNNL data set used for this experiment contained, on average, 78 `syslog` messages an hour for each node. As a result, there are approximately 699,678,720 messages on the cluster every year, which requires 41.7 GB to store.

Now consider that using only keywords or tag numbers to predict failures is rather effective. If one can predict events using only tag numbers or keywords, then perhaps one could keep only the fields required for these predictions.

While the approach which marries tag numbers and timing information is the best combination of speed and accuracy, combining the keywords with timing information performs the best overall. Keeping keywords provides another benefit over just keeping tags: some amount of semantic data is retained. Assume all words are kept. Keeping all of the words allows the same data to be broken up using a different set of keywords if a user is trying to predict another type of event or if a more effective keyword list for the current problem is found. In this case, the only fields that are necessary are the timing information and the message itself, as the level, facility, and tag numbers add nothing to this prediction approach. As a result, each message will be 31 bytes on average, which will take up 20.2 GB per year for a 51.563% reduction on the overall storage space needed.

Maximizing precision requires that one keep the tag numbers as well. Keeping tag numbers as well as timestamps and the message field uses 22.8 GB per year. Therefore, one would need to keep about 2.6 more GB per year than when using only keystrings to maximize recall. However, this still represents a marked improvement over the space required by standard `syslog` and is the best choice if one wishes to minimize the false positive rate.

There are two branches this research can take immediately. The first direction is to try different classification methods. This research only examines the effectiveness of the SVM approach to classifying nodes as likely to fail. Future work can explore the effectiveness of both unsupervised learning methods and other supervised learning methods.

Another direction this research could move in is to try to predict other events. Perhaps this same approach could be used to predict whether or not an entire node is going to go offline or if a RAID controller is going to fail. One would simply need to find these events in the logs and label each feature vector appropriately before training. Otherwise, the approach, as far as finding sequence numbers or keywords, is exactly the same.

The generalizability of this approach should also be examined by applying the approach to different data sets. For example, a cluster may have a different `syslog` configuration, average message rate, or applications which are installed than those seen in the data set used for this thesis. Perhaps these fluctuations in configuration also affect the usefulness of

the proposed approach. As another example, perhaps the tag number distribution in a given set up is different than that of the PNNL data set. In this case, it may be necessary to alter either the alphabet size or the cutoff values for each criticality score.

References

[1] Bruce Allen. Monitoring hard disks with smart. *Linux Journal*, 1(117), January 2004. Available at: http://www.linuxjournal.com/magazine/monitoring-hard-disks-smart. Accessed on April 19, 2010.

[2] Peter Broadwell. Component failure prediction using supervised naive bayesian classification, December 2002. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.4641. Accessed on April 19, 2010.

[3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[4] Errin W. Fulp, Glenn A. Fink, and Jereme N. Haack. Predicting computer system failures using support vector machines. In *First USENIX Workshop on the Analysis of Logs (WASL)*, 2008.

[5] Greg Hamerly and Charles Elkan. Bayesian approaches to failure prediction for disk drives. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICLM)*, June 2001.

[6] Andrew Karode. Support vector machine classification of network streams using a spectrum kernel encoding. Master's thesis, Wake Forest University, December 2008.

[7] Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for svm protein classification. In *Proceedings of the Pacific Symposium on Biocomputing 7*, January 2002.

[8] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. Failure prediction in ibm bluegene/l event logs. In *Proceedings of the Sevent IEEE International Conference on Data Mining*, 2007.

[9] C. Lonvick. The bsd syslog protocol, 2001. Available at: http://www.faqs.org/rfcs/rfc3164.html. Accessed on: April 19, 2010.

[10] John Makhoul, Francis Kubala, Richard Schwartz, and Ralph Weischedel. Performance measures for information extraction. In *Proceedings of DARPA Broadcast News Workshop*, pages 249–252, 1999.

[11] Joseph F. Murray, Gordon F. Hughes, and Kenneth Kreutz-Delgado. Hard drive failure prediction using non-parametric statistical methods. In *Proceedings of ICANN/ICONIP*, June 2003.

[12] E. Pinheiro, W.D. Webe, and L.A. Barroso. Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX Symposium on File and Storage Technologies (FAST '07)*, February 2007.

[13] Bianca Schroeder and Garth A Gibson. Understanding failures in petascale computers. *Journal of Physics: Conference Series*, 28, 2007.

[14] John Simpson and Edmund Weiner, editors. *Oxford English Dictionary*, volume 1. Oxford University Press, second edition, 1989.

[15] William H. Turkett, Andrew V. Karode, and Errin W. Fulp. In-the-dark network traffic classification using support vector machines. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2008.

[16] Doug Turnbull and Neil Alldrin. Failure prediction in hardware systems, 2003. Available at: http://www.cs.ucsd.edu/ dturnbul/Papers/ServerPrediction.pdf. Accessed on: April 19, 2010.

[17] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kauffman, second edition, 2005.