Autonomous Fault Detection in Self-Healing Systems: Comparing Hidden Markov Models and Artificial Neural Networks

Chris Schneider
University of St Andrews
School of Computer Science
Fife, Scotland
chris.schneider@st-andrews.ac.uk

Adam Barker
University of St Andrews
School of Computer Science
Fife, Scotland
adam.barker@st-andrews.ac.uk

Simon Dobson
University of St Andrews
School of Computer Science
Fife, Scotland
simon.dobson@st-andrews.ac.uk

ABSTRACT

Autonomously detecting and recovering from faults is one approach for reducing the operational complexity and costs associated with managing computing environments. We present a novel methodology for autonomously generating investigation leads that help identify systems faults. Specifically, when historical feature data is present, Hidden Markov Models can be used to heuristically identify the root cause of a fault in an unsupervised manner. This approach improves the state of the art by allowing self-healing systems to detect faults with greater autonomy than existing methodologies, and thus further reduce operational costs.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Applications and Expert Systems; I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods—*Plan execution, formation, and generation*

Keywords

self-healing systems; fault detection; machine learning; autonomic computing; artificial neural networks, hidden markov models;

1. INTRODUCTION

The operational costs of large-scale computing environments are continuing to increase. In order to address this problem, self-managing systems are being developed that reduce the supervisory needs of computing environments. Self-healing systems are one such example, and operate by autonomously detecting then recovering from faults. Although there have been numerous advances in both of these aspects, most self-healing systems continue to require periodic human oversight [1, 2, 3, 4]. This constraint poses challenges for the continued reduction of costs, and restricts self-healing recovery strategies to reactive approaches [5]. The importance of reducing human oversight in managing computing environments is multi-faceted. Although numerous direct benefits exist–such as the reduction staff involvement and their associated operating costs–further achievements can also be realised. Notably, self-healing systems have properties that are showing inherent benefits to change control schemas, and preserving baseline configurations [6].

The lack of change control or a baseline configuration can both introduce faults and present problems in determining their respective sources. Additionally, self-healing systems methodologies are also showing the capability to both detect and resolve faults without human supervision [5, 7, 8]. This is important when considering the costs and time requirements associated with training technical members of staff necessary to achieve these same results. If a system can find an appropriate recovery solution without the need for a subject matter expert, the associated costs can be immediately recovered.

However, achieving these goals is non-trivial and has posed notable challenges in both Machine Learning, and Artificial Intelligence. There is no assurance, for example, that self-healing systems leveraging evolutionary or search-space algorithms will find an appropriate solution for a given fault, or that any solution found will be optimal. Furthermore, computational costs of approaches that leverage these methodologies are typically higher than others, and impart a certain amount of risk of failing to identify or resolve faults. Anecdotal evidence suggests that in professional computing environments the failure to recognise or mitigate a fault is never an acceptable state. It is clear, however, that such circumstances do happen under human supervision, and they may be inevitable. The fact remains that moving to a software based approach poses challenges and questions regarding accountability–currently associated with human administrators–and liability. Both of these topics are outside of the scope of this paper, but the preference in supervised management approaches lends evidence to the desirability of these criteria [9, 10, 11, 12, 13, 14, 15, 16, 17]. The question remains: How can we further the autonomous behaviours of self-healing systems whilst reducing the operating costs of large-scale computing environments?

Previous research has shown that it is possible to autonomously synthesise new systems configurations [7], and determine common relationships between features [18]. This

ADAPT '14, Jan 22 2014, Vienna, Austria.
ACM 978-1-4503-2515-1/14/01.
http://dx.doi.org/10.1145/2553062.2553065.

has helped to reactively build recovery solutions in an unsupervised fashion and predict results of specific systems' configurations, respectively. The ability to autonomously identify anomalies has also been demonstrated by using a special type of unsupervised artificial neural network (ANN) [5] called a self-organising map [19]. This approach emphasises predictive behaviours by leveraging historical configuration data collected from a local system. However, at present there are no performance evaluations of self-healing systems utilising these methodologies. In order to understand how effective these approaches are they must be compared.

In this paper we present a novel approach for autonomously evaluating the source of a fault within a system by using Hidden Markov Models (HMMs) to predict the state of a feature. This approach is shown to be more effective than using a simple artificial neural network (ANN), and can be leveraged in a similar fashion to self-organising maps. Additionally, a comparison is provided between the performance of both the HMM and the ANN. This is intended as a baseline for future evaluations of the relative performance of self-healing approaches. It is our intent to continue to develop this research further and to eventually demonstrate potential reductions in the cost of operating large-scale IT environments through automation.

The rest of this paper is organised as follows: Section 2 contains a detail of the approach. Sections 3 and 4 describe the implementation, and key components of the methodology, respectively. Section 5 presents some early experimental results whilst Section 6 concludes with some directions for future exploration.

2. APPROACH

Using HMMs it is possible to identify the source of faults within a system without human intervention. HMMs use a learning algorithm to evaluate and predict changes in feature behaviour by utilising historical performance and configuration data periodically gathered from the system. This data is then autonomously classified through the use of fitness tests as either valid or invalid. Results from these tests determine the overall state of the system, and subsequently categorise the data collected in an identical fashion. This information is then used to provide direction to the HMM.

If the system passes all of its fitness tests, the associated configuration is assumed to be valid. The feature data is then converted into vectors based state changes and used to train HMMs to recognise acceptable patterns in feature behaviour. As the system passes its fitness tests, the prediction capabilities produced by the HMM become stronger. However, as systems behaviour can and is expected to change over time, previously learned information is gradually expired. This allows for elasticity in predictions by limiting the information learned to a recent time-series.

If the system does not pass all of its fitness tests, the associated configuration is assumed to be invalid. Once an invalid state has been determined an evaluation is done for each feature's behaviours based on the learned information.

Features that are determined to have behaved in an unexpected manner are added to a list of potential faults, along with a confidence value. The confidence value is determined by how unlikely the behaviour is to have occurred according to the HMM. Using the confidence value, the list of potential faults is then sorted in descending order. This provides both a measure of effectiveness of the application for determining the root cause of the fault, and an ability to prioritise subsequent self-healing strategies.

3. IMPLEMENTATION

In order to achieve the aforementioned approach, this experiment leverages C# and the Windows Management Instrumentation (WMI) framework for data collection [20]. A small application periodically interfaces with the WMI service based on a polling interval. The polling interval determines two properties: How frequently the WMI framework is to be queried, and how much elasticity to account for in behavioural pattern analysis. Although both values are fully adjustable, for the purposes of this experiment the polling interval is set at 60 seconds, and the total size of the dataset collection is limited to 30 samples. Each dataset is referenced within a list, and contains a collection of tables that individually correspond to a WMI class. As the WMI framework is queried, these tables are populated, associated with their respective dataset, and then categorised. Lastly, the information to be gathered is determined at run-time via a dictionary that stores a unique identifier value and the names of the WMI classes to be queried.

The categorisation of dataset information is accomplished via fitness tests that validate the responsibilities of the virtual machine. In this case the virtual machine's primary purpose is to act a web-server for both internal and external clients. Rather than using unit tests to verify a series of specific properties, fitness tests emphasise the validation of high-level processes and functions. This allows the application rather than an administrator to find the specific cause the anomaly. Furthermore, the use of fitness tests in this experiment accomplishes three goals: 1.) It emulates more closely the use of policies than unit tests–a goal for self-managing systems described by prior research [21, 22, 23, 24], and 2.) It roughly mirrors standard practice in existing computing environments where operational readiness testing or service-level agreements are required, and 3.) It establishes the groundwork for feeding in the results of this experiment with planned future research.

As previously stated, once a dataset is categorised as either valid or invalid the application will either update its predictive capabilities or it will look for anomalies, respectively. The dataset is determined to be valid if it passes all of its fitness tests. If this occurs, the each property within the collection of datasets is evaluated against itself. The hardest part of this procedure is uniquely identifying the objects that have been queried.

WMI does not provide a unique identifier for the values it produces, so an intersection is used to identify like-objects based on the lowest expected rate of change for a specific value within a given WMI class. This value, identified by column, is the primary reason for aforementioned WMI class dictionary's existence. After verifying that the application has no more than the maximum number of datasets, any changes–including removed or newly discovered properties–are catalogued and a vector is produced that contains change information. It is this vector that is used to autonomously train the anomaly detection frameworks (ADFs) in this experiment.

The ADFs in this experiment leverage one of two learning algorithms. The ADF's that utilise an HMM leverage the Baum-Welch algorithm [25, 26, 27]. This algorithm was chosen due to its suitability with HMMs inherent forward–

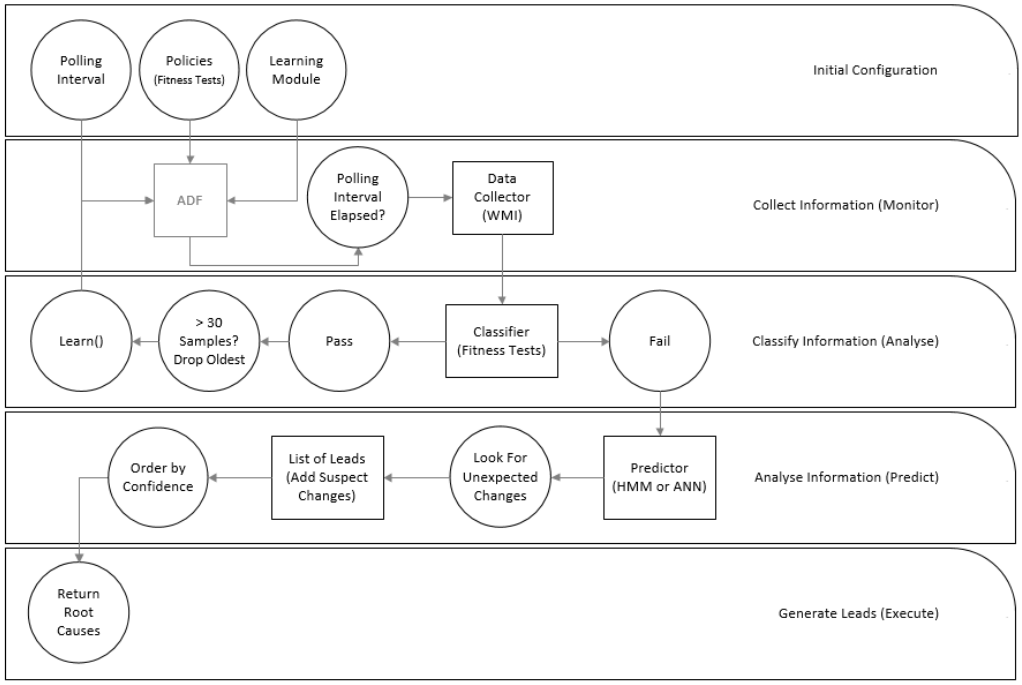

Figure 1: Anomaly Detection Framework Logic & Architecture Diagram

backward learning, and was implemented via the AForge-.NET [28] and Accord.NET Frameworks [29]. Conversely, the ADFs that leverage ANNs utilise a naïve Bayes approach. Each learning algorithm is responsible for processing observed feature behaviours into probabilities, which are used in conjunction with the ADF's classification of the datasets collected via WMI.

If the dataset is determined to be invalid, the feature's behaviours are analysed by the ADFs for unexpected changes. Any property that does not match the ADF's predicted values is added to a list of potential faults, along with a confidence value. As long as the fault source is collected within the WMI data, and the feature behaviours are sufficiently predictable, the root cause of a fault should be detected by the ADF–determining what constitutes as sufficient it is one of the primary goals of this experiment.

Although other learning algorithms are available, the comparison of their advantages and disadvantages comparing them to these two approaches remain beyond the scope of this experiment. This is primarily due to space and complexity constraints, but this is an intended area that hopefully will be explored in the future. Lastly, the underlying differences between HMMs and ANNs are not fully explored within this paper as we anticipate readers will be versed in these topics.

4. METHODOLOGY

This experiment leveraged 2 virtual machines running Windows 7, Internet Information Services (IIS) 7.5, and one of two versions of the ADF. Each virtual machine was cloned from a single initial image, and consisted of identical base configurations in hardware. The hardware itself was unremarkable being a standard image with 1GB of RAM, and a single disk partition divided into three volumes–one for the OS, the ADF, and the IIS webroot, respectively. The software was identical up until the point at which the ADFs were allowed to run for a period of 30 minutes on the machines.

During this time, the fitness tests were evaluated once every 60 seconds. If a system passed all of its fitness tests, each respective ADF would save both the configuration and metric data it gathered along with an XML schema file to a local data store. These files served as a mechanism for loading known good systems configurations quickly and, as a consequence, allowed for more rapid testing. Additionally, by approaching the experiment in this fashion we were able to reduced the opportunity for drift in each virtual machines' configuration. Each machine, once trained, was injected with either or a fault or a configuration change that was expected to trigger a fault.

The ADFs were then responsible for detecting the presence of the fault and generating a potential root cause, as well as reporting on several key attributes including: The total number of true positives, true negatives, false positives, and false negatives, the time taken in "ElapsedTicks" from the point in which a fault was detected until the completed generation of the ordered list of potential root causes, and the number of potential root causes (*i.e.*, 'leads').

True and false positives were determined when a fault was detected and whether or not it was or was not present, respectively. Conversely. true and false negatives were determined when a fault was not detected. However, due to the nature of false negatives, the number of faults not detected by the application had to be done by hand. This was as expected as there was no way, by definition, for the application to detect such a state without external validation. It is also the reason that faults in this experiment were injected

with the source already being known. From this information inferential metrics such as precision, time-taken, and leads generated. This data was then combined to produce charts showing the performance of the ADFs relative to the same tests.

The type of faults we injected had two variants: Adverse Configuration Changes (ACCs), and Direct Fault Injections (DFIs). The former consisted of shutting off services or making changes to the system using normal administrative methods. These changes were made in such a way that were expected to intentionally generate faults. This included changing disk structures, service states, and other properties that administrators would normally have access to. The latter consisted of copying code directly into the address space of another process, which in turn was expected to produce a controlled crash.

The ACCs we instantiated included: Disabling the network card, disabling the W3SVC service, removing the volume upon which the IIS webroot was contained, removing all free space from any of the three volumes, and disabling network access from one hop above the virtual machine's purview. The DFIs we instantiated included crashing various services such as: The IIS 7.5 W3SVC service, the Windows IPv4 network stack, and the Windows DNS service. Each ACC or DFI was run 6 times on the same ADF using 5, 10, 15, 20, 25, and 30 configuration samples. This allowed us to realise trends within each approach, and to see differences in both output and ADF confidence during each specific test.

The confidence values for each approach were generated using different methodologies based on their respective learning algorithms. In the case of the HMM, the confidence value was provided natively using the Baum–Welch algorithm. However, the ANN's use of simple vector analysis to estimate the likelihood of a feature's behaviour required setting a minimum confidence value. This was done as there was no reinforcement learning from which the approach could dynamically weight its expectations. As such, the ANN used a Naive Bayes approach based on the last observed behaviour in a known good state, less the probability of change for the last number of up to 30 samples. Any features that were not predicted to did not meet this 80% threshold were ignored.

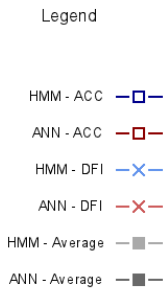

Figure 2: The following graphs use a shared convention when describing their data. Red and Blue lines represent data values for the W3SVC DFI and ACC tests, respectively. Grey lines represent averages for all conducted test results.

5. RESULTS

The successful evaluation of our experiment focused on whether or not faults could be autonomously evaluated using HMMs, and if a subsequent root cause could be correctly identified without human intervention. This approach was contrasted by using a simple ANN to achieve the same results in order to establish a baseline for understanding the effectiveness of this and future approaches. In summary, the results of each ADF's performance were mixed, but overall show support for our hypothesis.

It is possible for both methodologies to reliably detect faults and generate an accurately ordered list of potential root causes. The HMM performed more quickly and with greater accuracy then the simple ANN in nearly all circumstances. However, there were a few instances where the ANN performed similarly to or better than the HMM. This behaviour typically occurred much later in the observable series of events. Thus, while our initial results indicate the HMM is generally superior to a simple ANN, the supposition that this is always the case is not entirely conclusive.

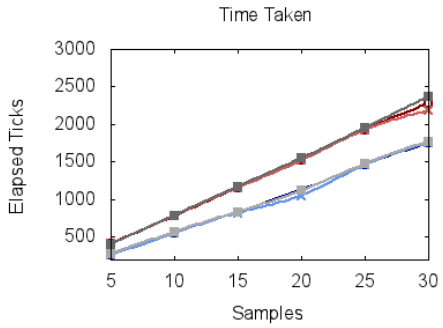

Figure 3: Time Taken represents the number of "ElapsedTicks" between when a fault was detected and the return of an ordered list of potential root causes based on confidence value.

The ability to determine the likelihood of a root cause and the time needed to train each ADF can be seen in the confidence and time taken graphs. Time taken represents the number of "ElapsedTicks" used by the system once the fault was detected, to the point where the list of potential root causes and their respective confidence values were generated (Figure 3). This value was used as it automatically accounts for CPU frequencies when performing timing calculations as well as minor but important discrepancies in how timing properties are measured in C#. This is not to be confused with "Elapsed Ticks", which are similar but do not take into account this discrepancy. In every case, the HMM was able to provide a list of potential root causes to the faults in less time than the ANN.

Confidence illustrates how likely the ADF believes a given lead is the root cause of a fault (Figure 4). In this case, the most likely root causes selected by the HMM and ANN approaches, respectively, were organised at the top of the list of faults. However, there was no assurance or guarantee that the ADF would select or provide a high degree of confidence for any of the tests that were run. The values in this chart illustrate the confidence values that were associated with the leads at the top of the sorted list, regardless of correctness. In addition to the the HMM taking less time to produce a

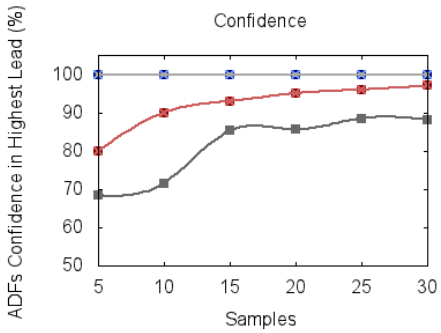

Figure 4: Confidence is a value used to convey how likely the ADF's suspect a given lead is associated with the root cause of the detected fault. In this case, the highest confidence values within the list are displayed which illustrates the learning rate of each approach.

list of potential faults than the ANN, the HMM identified more leads with greater confidence. Although this was the expected result, the supporting data did not match our exact expectations.

When the fault was introduced to the system via DFI, the ANN predicted the problem with greater confidence. Upon further investigation it seems that this is likely due to the fact that ACCs generate more changes to the systems configuration data than DFIs. This provides a greater number of leads for each ADF to investigate, but does not provide any kind of differentiation within the data itself. Since all changes associated with the fault are equally weighted diagnosing ACC generated faults are more challenging. Regardless, when attempting to ascertain the root cause the HMM was still able to identify the correct feature regardless of excess information. This is seen readily in the fault position results.

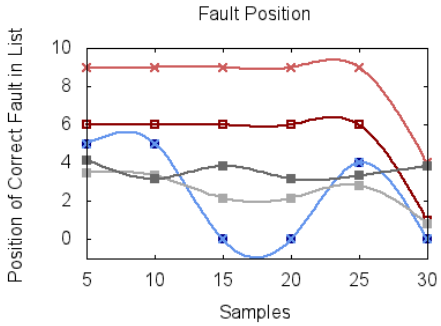

Figure 5: The position of the correct fault by ADF is represented in this graph. Notably, at 15 and 30 samples there are notable improvements in each approach.

Fault position describes where the correct root cause of a given fault exists in the ordered list (Figure 5). This is a human generated metric based on knowing the cause of the fault, *a priori*, and monitoring the effects of either a DFI or ACC. The lower the value, the more correct the diagnosis made by the ADF. This information, combined with the number of leads, illustrates how likely an ADF was to select the correct root cause of a fault and how many potential avenues for investigation were generated, respectively.

The combination of both higher confidence values, and a greater likelihood to pick the correct root cause when using an HMM may be due to the rate at which the HMM generates its confidence values. The HMM's use of forward–backward learning via the Baum–Welch algorithm allows it to inferentially predict patterns within its learned data. The ANN is at a disadvantage in this respect because it anticipates a certain amount of information before making a prediction. Although, theoretically, this should even out as more information is provided to the ANN.

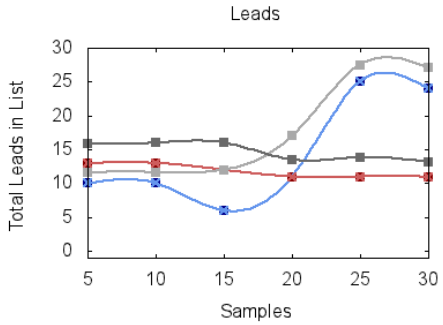

Figure 6: Each ADF is responsible for generating leads when a fault is detected. This graph represents the total number of suspect features by each approach, and their respective averages.

Converse to fault position, the number of leads is an autonomously generated metric that represents the total number of features that the ADF inferred as being a potential root cause in a detected fault (Figure 6). In both the HMM and ANN's, changes in a feature's behaviour alone were not enough to trigger an addition to the list of leads. The changes had to cross a certain threshold before they were determined to be 'unexpected'. This is where the differences in the learning algorithms are most obvious. The HMM is capable of setting a dynamic threshold for feature behaviours, where as the ANN must use a statically assigned 80% confidence value. The result was that the HMM generated more leads when compared to the ANN.

Although more leads can indicate greater sensitivity in detecting faults, higher values in this instance are not always better. In a perfect scenario only the exact root cause(s) should be provided. Interestingly, ADFs using HMMs were also more likely to select the correct root cause. Although the former is a strong indicator of the HMMs performance as being more the desirable than the ANN, the generation of larger numbers of leads is perhaps not.

In some cases the faults were not correctly positioned within the list–particularly when the ADF leveraged an HMM. This was due to second order sorting problems where the HMM had equally weighted two possibilities but the list had ordered the leads alphabetically. By having a greater number of leads, the HMM was not able to differentiate which feature was more relevant, though often they were related. For example, when the IIS web service were disabled using the ACC approach, 4 properties would be returned that were all correctly associated with the change. Because these properties were equally weighted, it pushed the correct lead

further down the list. One improvement might be to associate the root attributes of a lead and then list properties in a hierarchy. This would allow the root cause to be diagnosed as a subset of a specific feature and may lead to greater precision in future approaches.

This result seems to highlight an important discovery–that sorting the potential cause of faults based on a hierarchy may be a useful way determine or reinforce confidence.

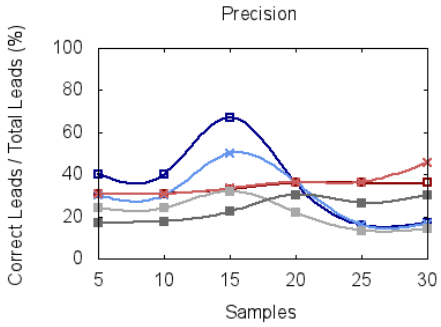

Figure 7: Correctly diagnosing the source of a fault is chiefly represented in the Precision metric. It illustrates where the correct source of the fault is in the already ordered list of leads. The higher the value on this chart, the fewer false leads generated by the ADF.

Precision is metric that represents the number of correct leads over the total leads generated by the ADF (Figure 7). Having only leads that were correctly associated with the fault showed higher precision. In the case of the HMM, most leads were categorised with a greater degree of precision than the ANN. However, as the number of samples increased this position reversed itself and the ANN showed greater precision than the HMM. This seems an obvious result as the number of root causes suspected by each respective approach changed in number. However, based on the number of samples it would appear that this metric is best served at about half of the expected number of values.

When this experiment was first developed, the expectation was that a greater number of configuration samples would yield more precise results. In reality it appears to be that precision increases at about half of the expected number of samples. It'd be worth exploring a larger sample size of systems' configurations to see if either ADF's precision values continue to peak at 15 samples, or if there is a trend towards $\frac{2}{x}$, where x is the number of samples the ADF had access to when the fault was detected. A confirmation of this result could yield novel approaches for predicting elasticity metrics within large–scale computing environments–a major factor in operational costs.

Similarly, improvements in notification of faults within large-scale computing environments may be possible where virtual machines are leveraged. In cloud computing environment, for example, it is often–but not always–the case that systems leverage the same baseline configuration[30]. As such, the use of the demonstrated fault detection methodologies could be used in a number of fashions including root–cause aggregation, and exploration of feature modelling via stationarity[31]. Both are topics that are beyond the immediate scope of this paper, but could be further explored.

Lastly, there were a few other notable differences in the data which we had not anticipated. The training rate of the HMM was faster than expected. This is most clearly illustrated in the differences between the confidence values between the HMM and the ANN. The differences in training rate led to some missed opportunities for additional testing, but a quick exploratory analysis reveals the HMM is capable of generating confidence values greater than 90% within as few as three samples. Whether or not the predictions made using these values would be correct, however, has not been examined.

6. CONCLUSION

In this experiment we successfully demonstrated that combining fitness functions and HMMs can autonomously detect faults and provide a list of their potential root causes. We compared this approach with a simple ANN, and helped to establish a baseline for further comparisons within the field. However, although the results from our experience were positive, we have not been able to demonstrate that these approaches would result in reduced operating costs for large-scale computing environments.

In addition to the improvements mentioned in the results section, the field would benefit greatly by a live study. Anecdotal evidence suggests that most if not all of the aforementioned algorithms that are capable of detecting faults would do so faster than human counterparts. However, there is limited evidence to support this supposition and it should be further explored.

More sophisticated approaches should be compared to better understand the advantages and disadvantages of certain technologies. The direct comparison of this approach with a SOM would possibly yield a better understanding of unsupervised approaches, collectively. Similarly, comparing methodologies that require human administration to those that do not may provide an avenue for understanding their relative advantages and disadvantages.

Using fitness functions we were able to presume that some faults effecting the areas of the system the fitness functions were monitoring were likely to be detected. As such, no specific model needed to be provided nor observed to ascertain faults–the ADFs built their own expectations of the features' behaviours so long as the fitness tests continued to pass. This approach, combined with the expiration of old data, was intended to allow for the systems to account for changes in behaviour and incorporate some native elasticity. It would be interesting to see further exploration into this area of study–particularly on the dynamic development of fitness tests through self–provisioning (*i.e.*, self-configuring) systems.

Lastly, although we were limited to predicting a single point within the feature behaviour, it may be possible to observe sequences of behaviour for more predictive monitoring. This would require using a different learning methodology such as the Viterbi algorithm [32], but it should be compatible with the approach described in this paper.

Acknowledgments

We would like to thank Saleem Bhatti and César Roberto de Souza for their help. Funding for this research was provided by the Scottish Informatics and Computer Science Alliance.

7. REFERENCES

[1] C. Schneider, A. Barker, and S. Dobson, "A survey of self-healing systems frameworks," in *Software Practice and Experience*. Wiley, 2013.

[2] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey, "Fulfilling the vision of autonomic computing," *IEEE Computer*, vol. 43, no. 1, pp. 35–41, January 2010.

[3] H. Psaier and S. Dustdar, "A survey on self-healing systems: approaches and systems," *Computing*, vol. 91, Issue: 1, pp. 43–73, 2010.

[4] J. McCann and M. Huebscher, "Evaluation issues in autonomic computing," in *Grid and Cooperatve Computing - GCC 2004 Workshops*. Springer Berlin, 2004, vol. 3252, pp. 597–608.

[5] D. J. Dean, H. Nguyen, and X. Gu, "Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems," in *Proceedings of the 9th international conference on Autonomic computing*, ser. ICAC '12. New York, NY, USA: ACM, 2012, pp. 181–190. [Online]. Available: http://doi.acm.org/10.1145/2371536.2371571

[6] D. Miorandi, D. Lowe, and L. Yamamoto, "Embryonic models for self–healing distributed services," in *Bioinspired Models of Network, Information, and Computing Systems*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, 2010, vol. 39, pp. 152–166.

[7] A. J. Ramirez, D. B. Knoester, B. H. Cheng, and P. K. Mckinley, "Plato: a genetic algorithm approach to run-time reconfiguration in autonomic computing systems," *Cluster Computing*, vol. 14, no. 3, pp. 229–244, Sep. 2011.

[8] O. Shehory, *A Self-healing Approach to Designing and Deploying Complex, Distributed and Concurrent Software Systems*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2007, vol. 4411, pp. 3–13.

[9] M. Aldinucci, M. Danelutto, G. Zoppi, and P. Kilpatrick, "Advances in autonomic components and services," in *From Grids to Service and Pervasive Computing*, T. Priol and M. Vanneschi, Eds. Springer US, 2008, pp. 3–17.

[10] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. Lo Presti, and R. Mirandola, "Moses: A framework for qos driven runtime adaptation of service-oriented systems," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–23, 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5963694

[11] S.-W. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives," in *ICSE 2006 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Shanghai, China: ACM, 2006, pp. 2–8, new York, NY.

[12] G. Li, L. Liao, D. Song, J. Wang, F. Sun, and G. Liang, "A self-healing framework for qos-aware web service composition via case-based reasoning," in *Web Technologies and Applications*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 7808, pp. 654–661.

[13] D. Menasce, H. Gomaa, S. Malek, and J. Sousa, "Sassy: A framework for self-architecting service-oriented systems," *Software, IEEE*, vol. 28, no. 6, pp. 78–85, 2011.

[14] H. Naccache, G. Gannod, and K. Gary, "A self-healing web server using differentiated services," in *Service-Oriented Computing – ICSOC 2006*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, vol. 4294, pp. 203–214.

[15] L. Rilling, "Vigne: Towards a self-healing grid operating system," in *Euro-Par 2006 Parallel Processing*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, vol. 4128, pp. 437–447.

[16] C. Schuler, R. Weber, H. Schuldt, and H. j. Schek, "Scalable peer-to-peer process management - the osiris approach," in *In: Proceedings of the 2 nd International Conference on Web Services (ICWS'2004)*. San Diego, CA: IEEE Computer Society, 2004, pp. 26–34, washington DC, USA.

[17] N. Stojnic and H. Schuldt, "Osiris-sr: A safety ring for self-healing distributed composite service execution," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*. Zürich, Switzerland: ACM, 2012, pp. 21–26, new York, NY.

[18] B. Garvin, M. Cohen, and M. Dwyer, "Failure avoidance in configurable systems through feature locality," vol. 7740, pp. 266–296, 2013. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36249-10

[19] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[20] "Windows management instrumentation," Microsoft Corporation, http://msdn.microsoft.com/en-us/library/aa384642(v=vs.85).aspx, Tech. Rep., 10 2013.

[21] D. M. Chess, V. Kumar, A. Segal, and I. Whalley, "Work in progress: Availability-aware self-configuration in autonomic systems," in *Utility Computing*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, vol. 3278, pp. 257–258.

[22] J. O. Kephart, "Autonomic computing: The first decade," in *International Conference on Autonomic Computing*. Karlsruhe, Germany: ACM SIGARCH/USENIX, 2011, pp. 1–56, new York, NY.

[23] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, Issue: 1, pp. 41–50, 2003.

[24] J. O. Kephart and W. E. Walsh, "An artificial intelligence perspective on autonomic computing policies." Yorktown Heights, NY, USA: IEEE Computer Society, June 2004, pp. 3–12, washington, DC, USA.

[25] L. Baum and T. Petrie, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–71, 1970.

[26] ——, "An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology," *Bulletin of the*

American Mathematical Society, vol. 73, no. 3, pp. 360–3, 1967.

[27] ——, "Statistical inference for probabilistic functions of finite state markov chains," *The Annals of Mathematical Statistics*, vol. 37, no. 6, pp. 1554–63, 1966.

[28] A. Kirillov, "Aforge.net framework," http://www.aforgenet.com/framework/members.html, 2013.

[29] C. R. Souza, "Accord.net framework," 2013, http://accord-framework.net/.

[30] G. Kirby, A. Dearle, A. Macdonald, and A. Fernandes, "An approach to ad hoc cloud computing," *ArXiv.org*, 2010, http://arxiv.org/pdf/1002.4738.pdf.

[31] W. Jiandong, T. Chen, and B. Huang, "Fir modelling for errors-in-variables/closed-loop systems by exploiting cyclo-stationarity," in *International Journal of Adaptive Control and Signal Processing*. John Wiley Sons, Ltd., 2007, vol. 27, no. 7, pp. 603–622, http://dx.doi.org/10.1002/acs.948.

[32] V. A. J., "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," in *IEEE Transactions on Information Theory 13*. IEEE Information Theory Society, 1967, vol. 13, pp. 260–269.