Can we Predict Performance Events with Time Series Data from Monitoring Multiple Systems?

Work-In-Progress Paper

Andreas Schörgenhumer
Christian Doppler Laboratory MEVSS
Johannes Kepler University Linz, Austria
andreas.schoergenhumer@jku.at

Mario Kahlhofer
Christian Doppler Laboratory MEVSS
Johannes Kepler University Linz, Austria
mario.kahlhofer@jku.at

Paul Grünbacher
Institute for Software Systems Engineering
Johannes Kepler University Linz, Austria
paul.gruenbacher@jku.at

Hanspeter Mössenböck
Institute for System Software
Johannes Kepler University Linz, Austria
hanspeter.moessenboeck@jku.at

ABSTRACT

Predicting performance-related events is an important part of proactive fault management. As a result, many approaches exist for the context of single systems. Surprisingly, despite its potential benefits, multi-system event prediction, i.e., using data from multiple, independent systems, has received less attention. We present ongoing work towards an approach for multi-system event prediction that works with limited data and can predict events for new systems. We present initial results showing the feasibility of our approach. Our preliminary evaluation is based on 20 days of continuous, preprocessed monitoring time series data of 90 independent systems. We created five multi-system machine learning models and compared them to the performance of single-system machine learning models. The results show promising prediction capabilities with accuracies and F1-scores over 90% and false-positive-rates below 10%.

KEYWORDS

Event Prediction, Multivariate Timeseries, Infrastructure Monitoring Data, Supervised Machine Learning

ACM Reference Format:
Andreas Schörgenhumer, Mario Kahlhofer, Paul Grünbacher, and Hanspeter Mössenböck. 2019. Can we Predict Performance Events with Time Series Data from Monitoring Multiple Systems?. In *Tenth ACM/SPEC International Conference on Performance Engineering Companion (ICPE '19 Companion), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3302541.3313101

1 INTRODUCTION

Predicting performance-related events is essential for proactive fault management [16] and anomaly prediction [21] in software systems. Advance warnings about slowdowns, crashes or other major performance degradations allow administrators to take preventative actions, thereby avoiding deviations of the systems' expected quality properties. Many approaches exist on predicting anomalies, failures and events, often based on machine learning (ML) techniques. For example, researchers have used log files [5, 6, 14, 17, 22] and time series monitoring data [1, 4, 15, 20, 21, 23] as input for a variety of ML algorithms. Despite the potential insights, there is still a lack of event prediction approaches for multi-system environments [18, 19], i.e., utilizing and combining the data of multiple, independent systems. First, multi-system event prediction could mitigate the issue of insufficient data, i.e., rare events or limited dynamic data for training ML models. Second, it allows cross-system event prediction for new systems, similar to the idea of cross-project (or company) defect prediction [8, 13], where data (e.g., source code metrics) from one set of projects is used for training, and data from other projects is used for testing.

This work-in-progress paper thus investigates the potential benefits of multi-system event prediction and presents first insights based on an industrial data set. Specifically, we work with events and low-level infrastructure measurements provided by an industry partner. Our data set includes time series for CPU, memory, disk and network metrics of multiple, independent systems. Using these metrics, we predict *service slowdown* performance events, which are often caused by effects observable in infrastructure metrics such as high CPU loads or suspicious disk behavior. Our paper first discusses the structure of our industrial data set. We then present our choices for preprocessing the data from multiple systems. Afterwards, we present a preliminary ML approach and an evaluation, which compares our multi-system prediction approach with two baselines: single-system prediction and naïve multi-system prediction. We conclude our paper with an outlook to future work.

2 STRUCTURE OF THE DATA SET

We briefly explain the structure of our data set, including infrastructure time series data and events, which are input to our multi-system event prediction approach. All data including its structure is provided by our industry partner.

A *system* (e.g., a travel booking platform) is a collection of interoperating hardware and software components. It is is operated by

ICPE '19 Companion, April 7–11, 2019, Mumbai, India

ACM ISBN 978-1-4503-6286-3/19/04…$15.00
https://doi.org/10.1145/3302541.3313101

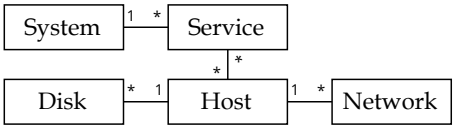

Figure 1: ERD of a monitored system.

a service provider and monitored at the level of components. Our data set contains multiple systems, which are completely independent of each other, without physical or logical interactions. The key elements of systems are represented in the entity-relationship diagram (ERD) shown in Figure 1. Services are abstract entities and represent the business logic of the system. Each service is also associated with attributes, such as its service type, indicating its operation or task (e.g., database access, web request).

Slowdown events occur at the level of services. They are detected by threshold-based heuristics, which determine if the average response times of the services exceed their baselines. Services are executed on hosts, which are physical or virtual computing entities. Services can also be executed on multiple hosts, e.g., in case of load balancing.

Table 1 lists all 34 metrics of our data set, which are available in one-minute resolution. Each minute is an aggregated average of ten-second samples. The hosts provide 11 different time series, including CPU, memory and operating system metrics. Multiple disks and network interfaces may be connected to each host. There are 13 time series for disk metrics and 10 time series for network metrics.

Hosts	Disks	Network Interfaces
CPU idle %	Space available	Bytes received
CPU system %	Space used	Bytes sent
CPU load %	Space free %	Recv. packets
CPU user %	Read bytes	Sent packets
CPU IO wait %	Write bytes	Recv. dropped
Page Faults / s	Read operations	Sent dropped
MEM available %	Write operations	Recv. errors
MEM available	Read time	Sent errors
MEM used	Write time	Recv. utilization
SWAP available	Util time	Sent utilization
SWAP used	Queue length	
	Inodes available %	
	Inodes total	

Table 1: Infrastructure metrics for entity types. Corresponding units are percentages, bytes, milliseconds and counts.

3 DATA SELECTION AND PREPROCESSING

Our data set contains 20 days of continuous monitoring data of about 250 independent systems and about 9,000 service slowdown events. Because some systems have only very few events, we discarded systems with fewer than 50 events, resulting in 90 systems and 7,500 events in total, though the event distribution still differs significantly (median: 111, mean: 166, max.: 1,125, std.: 157).

Before training our ML models, we preprocessed the data, which included retrieving the time series values for all events over all systems and dealing with missing data. To do so, we used the framework presented in [18], which yields comma-separated value (CSV) files that contain both positive feature vectors (samples where an event occurred) and negative feature vectors (samples where no event occurred).

In the following, we summarize our most important configuration options for working with the framework:

Sampling. For each system, we created positive samples at every event timestamp, and equally many negative samples at random timestamps where no event occurred, yielding 15,000 samples.

Observation window. For retrieving the time series data, we defined a 30-minutes observation window. At every timestamp (event for positive samples, random for negative samples), the preceding 30 minutes of each time series metric were then collected.

Aggregation functions. We then aggregated these 30-minute observation windows (chunks of raw data) using 11 functions representing the data entries in the final feature vectors (FVs): minimum, maximum, mean, standard deviation, 25% percentile, median, 75% percentile, skewness, kurtosis, slope and Pearson correlation.

Overall, these preprocessing steps yielded a single CSV file containing 15,000 observations (7,500 positive FVs and 7,500 negative FVs) with 374 features each (34 metrics × 11 aggregation functions), extracted from all 90 systems.

The second step involved the normalization of these FVs. We accomplished this by a service-based scaling approach. We first created FV groups of equal services, with respect to their unique identifiers. For each group, the data was scaled by subtracting the mean of the corresponding feature and dividing the result by the standard deviation (zero mean, unit variance). This way, we ensured that the values were equally normalized among all services, in contrast to scaling the entire data set at once. Finally, we removed duplicate FVs and balanced the data again to keep an equal amount of positive and negative FVs.

4 MACHINE LEARNING APPROACH AND PRELIMINARY RESULTS

We created five multi-system ML models and compared them with the performance of single-system ML models. Specifically, we used two baselines: the lower baseline given by the naïve multi-system model, and the upper baseline defined by the single-system models.

In particular, we used the default random forest classifier of scikit learn [12] to create the following models for our 90 systems:

(i) *Single-system models* were trained and tested individually for each of the systems.

(ii) The *naïve multi-system model* was trained on *all* systems, with a uniform number of observations per system in the training set (balanced systems).

(iii) *Clustered multi-system models* were evaluated like in (ii), after clustering data [21] into four different groups.

The four clusters for (iii) are determined by hand-crafted rules, which were inferred from observing statistical distributions and two-dimensional t-SNE visualizations [10] of system's features. t-SNE is a technique to reduce high-dimensional data to fewer dimensions. We used the entity counts and the number of service types as characterizing features of a system. As Figure 2 shows, we clustered systems primarily by their number of services s and

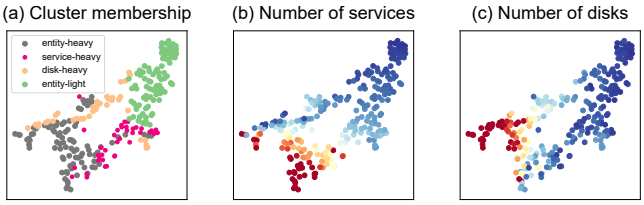

Figure 2: Visualization of system clusters, acquired with t-SNE, tinted by (a) their assigned cluster, (b) number of services and (c) number of disks per system, with blue tones indicating low and red tones indicating high entity counts.

disks d, where $Q_{0.5}(\mathbf{S})$ and $Q_{0.5}(\mathbf{D})$ denote the median number of services and disks, and the percentages the sizes of each cluster:

(1) entity-heavy	(~ 35%)	$s \geq Q_{0.5}(\mathbf{S})$	$\wedge$	$d \geq Q_{0.5}(\mathbf{D})$
(2) service-heavy	(~ 15%)	$s \geq Q_{0.5}(\mathbf{S})$	$\wedge$	$d < Q_{0.5}(\mathbf{D})$
(3) disk-heavy	(~ 15%)	$s < Q_{0.5}(\mathbf{S})$	$\wedge$	$d \geq Q_{0.5}(\mathbf{D})$
(4) entity-light	(~ 35%)	$s < Q_{0.5}(\mathbf{S})$	$\wedge$	$d < Q_{0.5}(\mathbf{D})$

Figure 3 shows our preliminary evaluation setup. We chose an 80:20 split for the training set (~ 12,000 observations) and test set (~ 3,000 observations), respectively, after randomly shuffling the observations. To ensure a uniform number of observations in all multi-system models (ii, iii), thus avoiding over-fitting to larger systems, we randomly discarded observations in larger systems, until all systems had the same number of observations. Since the event distribution among systems differs significantly, only about 25% of the training set data remained (3,000 observations).

Figure 4 shows the boxplots of three commonly used evaluation metrics [11]: accuracy, false-positive-rate (FPR) and F1-score. Each boxplot represents the 90 results we got for the systems in the test set data. Single-system models performed best, followed by our four clustered models and the naïve multi-system model. This result confirms our expectations. Single-system models generally perform best, as their training and test data is drawn from a very similar distribution. Still, the naïve multi-system model, which

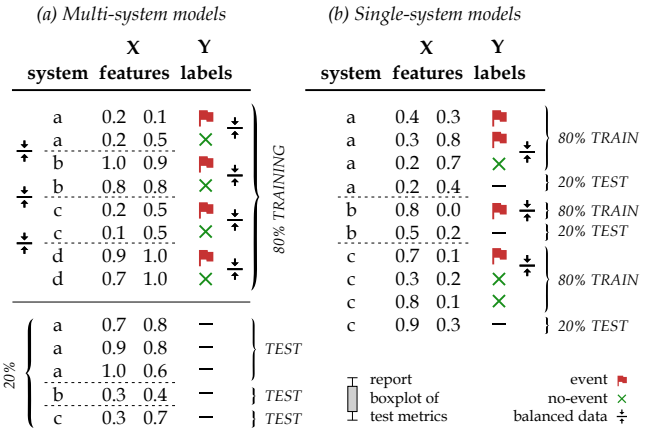

Figure 3: Multi- and single-system models evaluation setup.

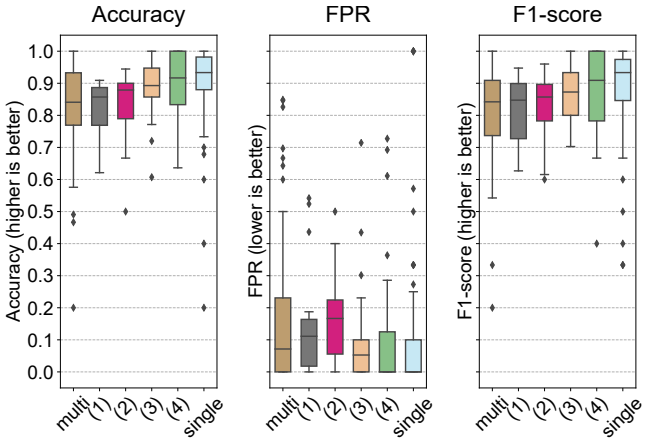

Figure 4: Test set accuracy, FPR and F1-score boxplots for single-system and naïve multi-system models, (1) entity-heavy, (2) service-heavy, (3) disk-heavy and (4) entity-light multi-system models.

has to generalize more, doesn't perform too bad with a median accuracy of 84% and a median FPR of 7%, though, with significant outliers. Promisingly, the median accuracy is between 86% and 92% and the FPR is between 11% and near 0% in the clustered models, indicating that multi-system event prediction can benefit from appropriate clustering. As expected, the clustered models place themselves below the single-system models, but above the naïve multi-system model. The difference of cluster (4) to the single-system models is quite small, even for the preliminary approach. Further improvements can be expected with a better clustering approach, making multi-system prediction a promising option.

5 RELATED WORK

While we are not aware on other research on multi-system event prediction, the area of single systems is already well researched. For anomaly, failure and event prediction, researchers focused on two main approaches based on dynamic system data: using log data and using monitoring data.

Log-data-based approaches. Fronza et al. [6], for example, parsed the performed operation, their severity (e.g., info, warning, error) and their timestamps from log files and used support vector machines for classification. In [14], the authors processed event logs with manual labels provided by a system administrator and then used sequences of log messages as input to check for patterns. Given a specified prediction window, they tested various ML models and achieved a high classification quality. Salfner and Tschirpke [17] also inspected event logs and extracted event sequences. They improved their prediction results by focusing on grouping and filtering. Yu et al. [22] gathered log data from an HPC (high performance computing) system, where they tested an event-driven approach for short-term (minutes) prediction and a period-based approach for long-term prediction (hours/days). They extracted several features (error code, severity, event time, etc.) from the logs and trained a Bayesian-based predictor. Das et al. [5] investigated different classes of failures also in HPC. They collected raw log data and performed a three-phase learning scheme (extract chains of events, augment

chains with expected lead times to failure, predict lead times) with a long short-term memory neural network.

Monitoring-data-based approaches. Alonso et al. [1] tested a variety of ML algorithms to predict system states based on feature vectors from a set of system metrics, such as CPU workload or disk used. Bodik et al. [4] predicted performance crises of data center servers based on so-called fingerprints. Fingerprints are aggregations of the collected performance metrics and can be interpreted as states of a data center. Pitakrat et al. [15] additionally considered the system architecture for failure prediction, where they define failures as violations of certain quality-of-service properties (e.g., response time below threshold). Using metrics such as CPU, memory and others at component level, they combine component failure predictors with a failure Bayesian-based propagation model for better results than monolithic approaches. In [20], the authors investigated the detection of problems in cloud environments, where they collected virtual machine metrics (page faults, context switches, latency, throughput, etc.) as well as host metrics (CPU, memory, cache). Zhang et al. [23] used fine-grained metrics such as process IDs or CPU consumption of threads to detect performance anomalies. In combination with anomaly detection, Tan et al. [21] used between 20 and 66 metrics to create alerts to signal upcoming performance anomalies. Using decision trees, they trained clusters of common execution contexts to improve prediction results.

Software reliability prediction. Long-term prediction of failures, is also an interesting area of research [2, 3, 7, 9]. Instead of dynamic system data, previous instances and recordings of failures are used for prediction. It would be worth investigating whether long-term reliability prediction is also possible in a multi-system scenario.

6 CONCLUSION

In this paper, we presented first steps towards an approach for multi-system event prediction that operates on infrastructure metrics to predict performance-related events. We preprocessed 20 days of monitoring data of 90 independent systems and trained various ML models to predict service slowdowns in each of them. In our evaluation, we compared a naïve multi-system model, which simply uses all systems for training, to four multi-system models that cluster systems based on their disk and service counts, as well as single-system models. Results show that multi-system models achieve high accuracies, F1-scores and low FPRs, especially when finding appropriate clusters.

Our preliminary results are encouraging, and there are interesting opportunities for future work. The reasons for the differences in our clustered results need to be investigated, and we will also work on an improved clustering approach that takes additional system characteristics (e.g., service metadata) into account. We plan to improve our approach by considering further preprocessing options and additional ML algorithms to get even better results. We will also address the problem of changing systems, which requires online model retraining. Moreover, we are eager to determine prediction performance for systems with limited data and yet unseen systems.

ACKNOWLEDGMENTS

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

REFERENCES

[1] Javier Alonso, Luis Antonio Belanche Muñoz, and Dimiter Avresky. 2011. Predicting software anomalies using machine learning techniques. In *Proceedings of the International Symposium on Network Computing and Applications*. IEEE Computer Society Publications, 163–170.

[2] Ayman Amin, Lars Grunske, and Alan Colman. 2013. An approach to software reliability prediction based on time series modeling. *Journal of Systems and Software* 86, 7 (2013), 1923–1932.

[3] Momotaz Begum and Tadashi Dohi. 2017. A neuro-based software fault prediction with Box-Cox power transformation. *Journal of Software Engineering and Applications* 10, 03 (2017), 288.

[4] Peter Bodik et al. 2010. Fingerprinting the Datacenter: Automated Classification of Performance Crises. In *Proceedings of the 5th European Conference on Computer Systems (EuroSys '10)*. ACM, 111–124.

[5] Anwesha Das, Frank Mueller, Charles Siegel, and Abhinav Vishnu. 2018. Desh: Deep Learning for System Health Prediction of Lead Times to Failure in HPC. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '18)*. ACM, New York, NY, USA, 40–51.

[6] Ilenia Fronza, Alberto Sillitti, Giancarlo Succi, Mikko Terho, and Jelena Vlasenko. 2013. Failure prediction based on log files using Random Indexing and Support Vector Machines. *Journal of Systems and Software* 86, 1 (2013), 2–11.

[7] Song Fu and Cheng-Zhong Xu. 2007. Exploring Event Correlation for Failure Prediction in Coalitions of Clusters. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC '07)*. ACM, New York, NY, USA, Article 41, 12 pages.

[8] Seyedrebvar Hosseini, Burak Turhan, and Dimuthu Gunarathna. 2018. A Systematic Literature Review and Meta-Analysis on Cross Project Defect Prediction. *IEEE Transactions on Software Engineering* (2018).

[9] Arunima Jaiswal and Ruchika Malhotra. 2016. Software Reliability Prediction Using Machine Learning Techniques. In *Proceedings of Fifth International Conference on Soft Computing for Problem Solving*. Springer Singapore, 141–163.

[10] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, Nov (2008), 2579–2605.

[11] Cathy O'Neil and Rachel Schutt. 2014. *Doing Data Science* (3 ed.). O'Reilly Media Inc., Sebastopol, CA.

[12] Fabian Pedregosa et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[13] Fayola Peters, Tim Menzies, and Andrian Marcus. 2013. Better Cross Company Defect Prediction. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. IEEE Press, Piscataway, NJ, USA, 409–418.

[14] Teerat Pitakrat, Jonas Grunert, Oliver Kabierschke, Fabian Keller, and André van Hoorn. 2014. A Framework for System Event Classification and Prediction by Means of Machine Learning. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS '14)*. ICST, 173–180.

[15] Teerat Pitakrat, Dušan Okanović, André van Hoorn, and Lars Grunske. 2018. Hora: Architecture-aware online failure prediction. *Journal of Systems and Software* 137 (2018), 669–685.

[16] Felix Salfner, Maren Lenk, and Miroslaw Malek. 2010. A Survey of Online Failure Prediction Methods. *ACM Comput. Surv.* 42, 3 (2010), 10:1–10:42.

[17] Felix Salfner and Steffen Tschirpke. 2008. Error Log Processing for Accurate Failure Prediction. In *Proceedings 1st USENIX WS on the Analysis of System Logs*.

[18] Andreas Schörgenhumer, Mario Kahlhofer Peter Chalupar, Hanspeter Mössenböck, and Paul Grünbacher. 2019. A Framework for Preprocessing Multivariate, Topology-Aware Time Series and Event Data in a Multi-System Environment. In *Proceedings 19th International Symposium on High Assurance Systems Engineering (HASE)*. IEEE, Hangzhou, China.

[19] Andreas Schörgenhumer, Mario Kahlhofer, Hanspeter Mössenböck, and Paul Grünbacher. 2018. Using Crash Frequency Analysis to Identify Error-Prone Software Technologies in Multi-System Monitoring. In *The 18th IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 183–190.

[20] Bikash Sharma, Praveen Jayachandran, Akshat Verma, and Chita R. Das. 2013. CloudPD: Problem determination and diagnosis in shared dynamic clouds. In *Proceedings 43rd International Conference on Dependable Systems and Networks*. 1–12.

[21] Yongmin Tan, Xiaohui Gu, and Haixun Wang. 2010. Adaptive System Anomaly Prediction for Large-scale Hosting Infrastructures. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*. 173–182.

[22] Li Yu, Ziming Zheng, Zhiling Lan, and Susan Coghlan. 2011. Practical online failure prediction for Blue Gene/P: Period-based vs event-driven. In *Proceedings 41st International Conference on Dependable Systems and Networks Workshops*. 259–264.

[23] Xiao Zhang, Fanjing Meng, Pengfei Chen, and Jingmin Xu. 2016. TaskInsight: A Fine-Grained Performance Anomaly Detection and Problem Locating System. In *Proceedings of the 9th International Conference on Cloud Computing*. 917–920.