# Cloud Performance Variability Prediction

Yuxuan Zhao
Leiden University
y.zhao.2@umail.leidenuniv.nl

Dmitry Duplyakin
University of Utah
dmdu@cs.utah.edu

Robert Ricci
University of Utah
ricci@cs.utah.edu

Alexandru Uta*
Leiden University &
Amazon
a.uta@liacs.leidenuniv.nl

## ABSTRACT

Cloud computing plays an essential role in our society nowadays. Many important services are highly dependant on the stable performance of the cloud. However, as prior work has shown, clouds exhibit large degrees of performance variability. Next to the stochastic variation induced by noisy neighbors, an important facet of cloud performance variability is given by *changepoints*—the instances where the non-stationary performance metrics exhibit persisting changes, which often last until subsequent changepoints occur. Such undesirable artifacts of the unstable application performance lead to problems with application performance evaluation and prediction efforts. Thus, characterization and understanding of performance changepoints become important elements of studying application performance in the cloud. In this paper, we showcase and tune two different changepoint detection methods, as well as demonstrate how the timing of the changepoints they identify can be predicted. We present a gradient-boosting-based prediction method, show that it can achieve good prediction accuracy, and give advice to practitioners on how to use our results.

**ACM Reference Format:**
Yuxuan Zhao, Dmitry Duplyakin, Robert Ricci, and Alexandru Uta. 2021. Cloud Performance Variability Prediction. In *Companion of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21 Companion), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3447545.3451182

## 1 INTRODUCTION

Cloud computing powers many important application domains, ranging from business, government, to science. The stable performance of the cloud servers is essential to providing good user experience [4, 16] for applications in these domains. Nonetheless, the performance of cloud computing infrastructure fluctuates even when the same code runs on the same hardware at different points in time. This phenomenon is known as performance variability [2, 25] of clouds. Even though cloud variation might be seemingly small, application performance suffers significantly [6, 18, 25], with complex implications in performance evaluation, predictability, as well as operation of time-sensitive applications. In the current work, we investigate whether certain artifacts of performance variability can be predicted.

Previous work identified multiple classes of performance variability: (1) stochastic variability from noisy neighbors [16, 25]; (2) poor interaction with cloud provider policies [25]; (3) cloud performance changepoints [6]. In the context of the latter, performance distributions do not remain stationary but rather show frequent changes, sources of which are difficult to identify in many cases. These performance distributions "shift" when *changepoints* occur. Using *changepoint detection* [6, 12, 14, 15, 22, 23] methods, one can identify when such changes occur. Understanding of performance changepoints might be necessary in the work of cloud practitioners and operators [6]. Performance metrics can be affected by hardware aging, firmware and software updates, security patches[1], and even changes in physical environments [9]. Some of these changes are planned, while many are not.

A changepoint refers to a moment in time when statistical properties before and after it differ considerably [6, 12]. Changepoints can have an impact on the user experience [12]. For instance, in clouds, changepoints in time series data with system metrics, potentially caused by hardware issues, can have an impact on the availability and performance of a service. Duplyakin et al. [6] describe changepoint characteristics and detect changepoints in the performance of CloudLab [5], a large-scale testbed for cloud computing.

Different changepoint detection methods use different analysis techniques, from Bayesian to time series analysis techniques [7, 12, 14, 15, 22]. In this paper, we consider two efficient and recently developed methods—*robseg* [7] and *breakout detection* [12]—yet the prediction methodology we describe and evaluate is applicable to other methods.

The possibility to predict performance changepoints in advance enables proactive measures in the management of resource provisioning, which can help stabilize key performance metrics. From the user perspective, changepoint prediction can enable more informed experiment planning, contributing to repeatable and reproducible experimental evaluations. In this paper, we describe our analysis of three gradient boosting prediction methods and discuss their trade-offs.

This paper describes how we run two changepoint detection methods on the dataset with 6.9M datapoints performance measurements collected on the CloudLab testbed [6]. For *robseg*, we tune its single and intuitive hyperparameter in the analysis process, but for *breakout detection*, which has four hyperparameters, we put effort into tuning it to best match the *robseg*'s results and show the agreement between the two methods.

In this paper, we make the following contributions:

*Alexandru Uta holds concurrent appointments at Leiden University and as an Amazon Visiting Academic. This paper describes work performed at Leiden University and is not associated with Amazon.

[1] https://databricks.com/blog/2018/01/13/meltdown-and-spectre-performance-impact-on-big-data-workloads-in-the-cloud.html


*ICPE '21 Companion, April 19–23, 2021, Virtual Event, France*

ACM ISBN 978-1-4503-8331-8/21/04…$15.00
https://doi.org/10.1145/3447545.3451182

(1) We give a refresher on gradient boosting and changepoint detection (Section 2).
(2) We tune *breakout detection* to achieve comparable results to changepoint detection with *robseg* (Section 3).
(3) We present a method for predicting performance variability given by changepoints. We use the gradient boosting model to predict the time when a changepoint would occur. We show that the selected methods perform well and our confidence level in predicting changepoints is high. We make all developed code artifacts publicly available[2] (Section 4).

## 2 BACKGROUND

In this section, we introduce the concepts needed to achieve efficient prediction mechanisms for cloud systems performance changepoints.

**Gradient boosting.** Gradient boosting algorithms were developed by Friedman [8] in 1999. In machine learning, regression and classification problems are solved using grading boosting—techniques which build predictive models. Gradient boosting produces a strong prediction model by assembling weak prediction models. Gradient descent methods are used in gradient boosting to minimize the loss in the procedure of adding trees. In this paper, we apply three popular gradient boosting frameworks, which are XGBoost [3], LightGBM [13] and CatBoost [27]. XGBoost benefits from parallelized tree boosting, which provides efficient implementation. LightGBM can produce more accurate results with lower memory usage. CatBoost reduces time spent on parameter tuning due to its great results with default parameter settings and allows users to utilize non-numeric factors, which is important in some prediction scenarios.

**Changepoint Detection.** The changepoint is the moment of time when a given dataset is split in two subsets that exhibit different statistical properties. The method of finding changepoints is called changepoint detection [23].

Fearnhead and Rigaill proposed in 2016 a changepoint detection technique that is robust to the existence of outliers[7]. This technique replaces the $L_2$ (square error) loss function with an alternative loss function in order to make it less sensitive to outliers. To determine the changepoints and their locations, their implementation, *robseg*, focuses on the minimum penalized cost method. After defining the loss function, *robseg* presents a dynamic programming algorithm to minimize the cost efficiently. The algorithm, named R-FPOP, is an extension of the pruned DP [20] and the FPOP [17] algorithms.

*Breakout detection* is proposed by James et al. [12], as a statistical technique for detecting breakouts in cloud data. In statistics, breakouts refers to changepoints. A study on *breakout detection* does not make explicit claims about robustness to outliers but describes high efficacy of the method in practical scenarios. The method involves the concept of E-Divisive with Medians (EDM) [24], which is proposed as a novel statistical technique for automatic detection of changepoints. EDM is a non-parametric method that uses E-statistics to detect the divergence. E-statistics relies on the estimates of energy distance between two given variables for measuring the divergence in the means. Thus, the method determines whether changepoints are statistically significant or not and returns estimates of changepoint locations.

[2]https://github.com/ZhaoNeil/cloud-performance-variability-prediction

**Table 1: Changepoints collected by robseg.**

| Parameter K | CPU: CP # | Mem: CP # | Disk: CP # |
|---|---|---|---|
| **0.3** | 49 | 492 | 42 |
| **0.4** | 147 | 925 | 76 |
| **0.5** | 193 | 1,113 | 108 |
| **0.6** | 235 | 1,263 | 134 |
| **0.7** | 283 | 1,406 | 160 |
| **0.8** | 325 | 1,529 | 191 |
| **0.9** | 363 | 1,630 | 215 |
| **1.0** | 410 | 1,784 | 245 |
| **Total** | 2,005 | 10,142 | 1,171 |

## 3 CHANGEPOINT COLLECTION

We use a 6.9M datapoints performance measurements dataset collected on the CloudLab testbed by Duplyakin et al. [6]. They measure CPU performance of CloudLab hardware such as CPU, memory, disks, networks. To evaluate the memory performance, 1038 memory configuration measurements are collected. Similarly, 152 disk configurations in the performance measurements dataset are included. There are eight test measurements per device: read and write load, random and sequential tests, low and high 'iodepth' settings [6].

***robseg* Method.** An implementation of changepoint detection which is implemented in the *robseg* R package [21] is used to collect changepoint in the performance measurements dataset. This implementation involves a threshold K, related to magnitudes of potential changepoints and the ratio of the signal to its standard deviation. Duplyakin et al. experiment with [0.3,1.0] range for K values [6]. Finally, 13,318 changepoints are collected, consisting of 2,005 CPU changepoints, 10,142 memory changepoints, and 1,171 disk changepoints. The distribution of changepoints is depicted in Table 1.

***Breakout Detection* Method.** In this section, we discuss the changepoint collection procedure with *breakout detection* method. In *breakout detection*, there are several parameters related to our experiment. Figure 1 is an example displaying the comparison between *robseg* Changepoint Detection and *breakout detection*. Figure 1 shows that these two methods can get similar changepoints. The locations of changepoints found by two methods are approximate when the change is remarkable, which are represented by the green vertical lines and blue vertical lines. But when the trend of points is stable, it shows there is a noticeable difference between the results of the two methods. Changepoints collected by *breakout detection* are more fine-grained than changepoints collected by *robseg*. We consider the following parameters to be tuned for *breakout detection* [3]:

- `min_size` is the minimum number of observations between changepoints.
- `degree` can take the values 0, 1 or 2, which represents the degree of the penalization polynomial.

[3]https://github.com/indeedeng/anomaly-detection/blob/master/breakout.py

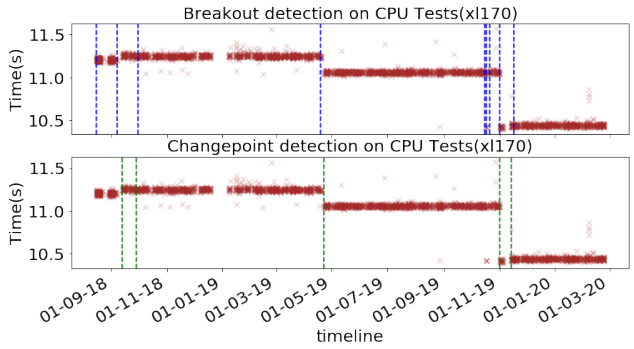


**Figure 1: The comparison between Changepoint Detection and Breakout Detection on CPU Tests(xl170). Threshold in Changepoint Detection is set to 2.5. Min_size in Breakout Detection is set to 10, beta is set to 0.0001 and degree is set to 1. Green line indicates the changepoints from Changepoint Detection, while blue lines indicate changepoints from Breakout Detection.**

**Table 2: The distribution of changepoints collected by Breakout Detection.**

| min_size | CPU: CP # | Mem: CP # | Disk: CP # |
|---|---|---|---|
| **10** | 3,433 | 9,870 | 2,496 |
| **13** | 2,557 | 7,145 | 1,710 |
| **16** | 1,746 | 5,477 | 1,349 |
| **19** | 1,386 | 4,210 | 1,122 |
| **22** | 1,141 | 3,225 | 964 |
| **25** | 956 | 2,938 | 860 |
| **Total** | 11,219 | 32,865 | 8,501 |

- `beta` is a a parameter used to further control the amount of penalization, which is the default form of penalization.
- `percent` is a parameter used to further control the amount of penalization, which represents the minimum percentage change of the goodness-of-fit statistics to consider adding other changepoints.

We collect the changpoints using *breakout detection*. For *robseg*, we tune its single and intuitive hyperparameter in the analysis process, but for *breakout detection*, which has four hyperparameters, we put effort into tuning it to best match the *robseg*'s results and show the agreement between the two methods. Thus, we manually set beta to 0.008 and min_size from 10 to 25. In the end, we collected 52,585 changepoints, consisting of 11,219 CPU changepoints, 32,865 memory changepoints, and 8,501 disk changepoints. The distribution of changepoints is depicted in Table 2. Analyzing these results, one can immediately conclude that *breakout detection* results in much more fine-grained changepoints, which are likely more difficult to predict.

# 4 PERFORMANCE VARIABILITY PREDICTION

In this section, we use gradient boosting methods to predict when changepoints occur based on their systematic information. Systematic information is a mixture of continuous and categorical features as input variables, such as hardware type, number of threads used, compiler version, etc. What we predict is the timestamp of a changepoint, a real number representing the UNIX epoch value of the time the test occurred. We apply root mean square error (RMSE) in our experiments as an estimator score method to qualify the quality of predictions. Several sets of features are tried in this experiment in order to get a minimum RMSE and figure out the feature importance of these datasets, namely, which feature is informative in the process of prediction. Moreover, in order to attain a more accurate prediction outcome, we involve a hyperparameter optimization method, random search, in our experiment. To avoid overfitting, we use 5-fold cross validation.

**Method.** We regard this prediction problem as a regression problem. Furthermore, this problem can be regarded as a regression tree more specifically. In this paper, we use three kinds of gradient boosting models, which are XGBoost, LightGBM, and CatBoost. The main idea behind the gradient boosting method is to ensemble weak learners as strong learners. Here, regression trees are used to output real values. Trees are added subsequently and correct the predictions in a greedy manner to minimize the loss function. In this paper, we apply RMSE as our loss function. The time when changepoints appear is what we aim to predict, which can be considered as a real number when it is in the form of a timestamp. Three gradient boosting methods mentioned above are implemented on the changepoint dataset collected before. This dataset contains the main performance indicators for changepoints on CPU, memory, and disk. We use the Scikit-learn [19] package in Python as our experiment tool.

**Training and Prediction.** The dataset is divided into two parts: the former for training and the latter for testing. We use Scikit-learn [19] to split the dataset randomly. The ratio is set to 0.8, which means 80% of the whole dataset is set as the training set and 20% of the whole dataset is set as the test set. The feature vector is our training data and the corresponding timestamp is our target values. We fit them into the model and get the model trained. When we input a feature vector of a changepoint in the test set into the trained model, a timestamp will be predicted, which is the predicted time of when that changepoint could appear. It will be compared with the true value of the timestamp.

**Chosen Predictors and Initial Performance.** We have experimented with several features of the changepoints to understand what variables influence the occurrence of a changepoint. Due to space constraints, we only present our best-achieved results and the prediction variables that produced them. Naturally, changepoints for CPU and memory occur for different reasons than e.g., disk changepoints. Our method reveals this information. Empirically, we find evidence that changes in the kernel version give rise to performance variability. Similar to the changes of the kernel version, OS version changes give rise to performance changepoints as well. We also find that predictors like *hardware type* and *test*

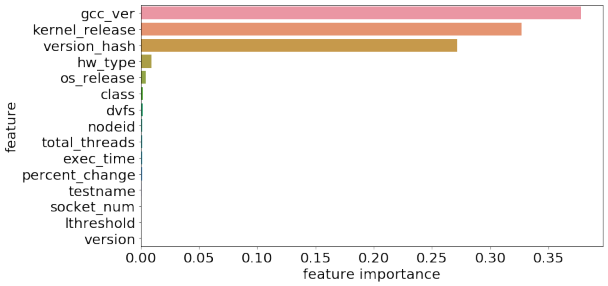


**Figure 2: The feature importance of CPU changepoints.**

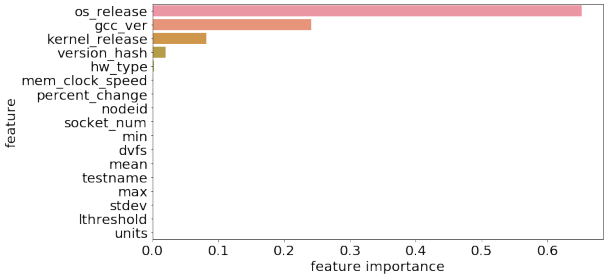


**Figure 3: The feature importance of memory changepoints.**

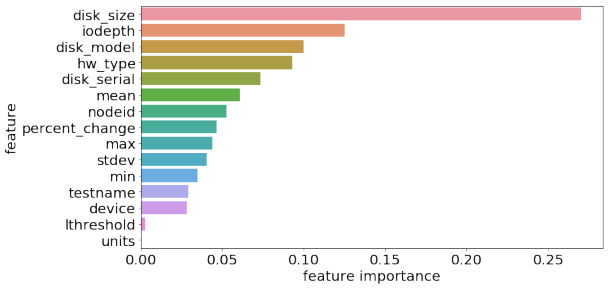


**Figure 4: The feature importance of disk changepoints.**

*name*, have a very low correlation to the occurrence time of performance changepoints. By contrast, entries in kernel version, OS version, and compiler version, etc. have a high correlation to the time of changepoint appears. These features relate to the changepoint occurrence time very closely, they will have a significant impact on changepoints instead of main performance indicators of CPU, memory, and disk.

Feature importance represents a score to indicate how the influence of every feature when constructing the boosted decision trees: the more a feature is used, the higher its relative importance. This score is computed by the number of performance indicators improved by each attribute split point and weighted by the number of observations the node is responsible for [10].

Figure 2 shows the impact of the features we consider for the prediction of changepoints on the CPU changepoint dataset. Figures 3 and 4 show the feature importance for the memory and disk changepoints, respectively. Using as predictor features these

**Table 3: The performance of our prediction method for changepoints CPU, memory, and disk performance.**

| RMSE (days) | XgBoost | LightGBM | CatBoost |
|---|---|---|---|
| Memory | **5.20** | 5.79 | 6.75 |
| CPU | **5.52** | 5.88 | 6.03 |
| Disk | **42.47** | 47.65 | 47.06 |

**Table 4: The runtime of our prediction method for changepoints CPU, memory, and disk performance without random search and cross validation.**

| Runtime (s) | XgBoost | LightGBM | CatBoost |
|---|---|---|---|
| Memory | 5.98 | 0.76 | 3.16 |
| CPU | 1.50 | 0.41 | 1.42 |
| Disk | 1.33 | 0.38 | 1.43 |

**Table 5: The performance of our prediction method for changepoints CPU, memory, and disk performance using a 5-fold cross validation strategy and random search.**

| RMSE (days) | XgBoost | LightGBM | CatBoost |
|---|---|---|---|
| Memory | 4.72 | **4.51** | 5.30 |
| CPU | 5.75 | **5.54** | 5.79 |
| Disk | **52.52** | 55.73 | 52.85 |

data, our prediction method gets remarkable results in predicting changepoints for CPUs and memory. Table 3 shows our results. The only type of changepoints we are unable to predict accurately is disk changepoints. We do not understand at the moment what generates these and by considering Figure 4 we understand that many features contribute to the changepoints, such as *disk_size*, *iodepth*, *disk_model*, etc. We leave for future work uncovering the phenomenon behind disk changepoints.

Table 4 displays the runtime of our prediction methods for changepoints on CPU, memory, and disk performance without random search and cross validation. The prediction runtime is based on the experiment on a 2.9 GHz Dual-Core Intel Core i5 CPU with 8 GB memory. Our results indicate that it is possible to perform this analysis efficiently, as most of these runs take less than 1.5 seconds. Only CatBoost and XgBoost for the memory dataset take 3.16 seconds, and 5.98, respectively. Therefore, these kinds of analyses could be performed in a real-time pipeline.

**Cross Validation.** Learning the parameters of a prediction model and testing on the same data usually gets a satisfactory result but fails to get good results on different test sets—due to overfitting. To avoid overfitting, we apply cross validation, which is a resampling procedure to learn parameters on a limited data sample. K-fold cross validation divides the whole dataset into K folds and makes sure that every fold will be used as a test set at one certain iteration. The model is trained on the remaining K-1 folds per iteration and there are K iterations in total. The final metric result provided by K-fold cross validation is the average of the metrics results of every iteration. In this experiment, we use 5-fold cross validation to avoid overfitting.

**Table 6: The performance of our prediction method for breakout detection changepoints on CPU, memory, and disk performance using a 5-fold cross validation strategy and random search.**

| RMSE (days) | XgBoost | LightGBM | CatBoost |
|---|---|---|---|
| Memory | **11.67** | 13.51 | 15.15 |
| CPU | **13.67** | 14.97 | 15.64 |
| Disk | **83.63** | 95.22 | 98.33 |

**Random Search.** To achieve better predictions, namely a lower RMSE in our experiment, we decide to objectively search different values for model hyperparameters. The aim of this optimization process is to find a configuration resulting in the best performance, minimum RMSE in our experiment specifically. Random search is one of the simplest and most common methods. In random search, we define a bounded domain of hyperparameter values as a search space and then search sample points in that domain randomly. At the expense of certain accuracy, random search can reduce the search time while ensuring model accuracy. Table 5 shows our results using both the 5-fold cross validation and the random search. Using these techniques improves the performance of predicting changepoints, as can be observed by noticing the differences between Tables 3 and 5.

**Predicting *Breakout Detection.*** Table 6 shows the RMSE of three changepoint collected by *breakout detection* method using all features as predictors with random search and 5-fold cross validation. Both memory and CPU datasets get RMSE around 10 days. The RMSE of the disk dataset is inferior to the results of memory and CPU datasets. Furthermore, the performance of prediction on changepoints collected by *breakout detection* is not as good as the performance of prediction on changepoints collected by *robseg*. The main reason is that the changepoint collected by *breakout detection* method is more fine-grained than the changepoint collected by *robseg*.

**Discussion and advice for practitioners.** In evaluating our method for changepoint prediction, we made the following observations. In summary, we find that:

(1) **Changepoints on different cloud resources depend on different factors.** We have found that CPU and memory changepoints depend heavily on OS, kernel, and compiler versions, while disk-based changepoints depend on factors related to hardware. Practitioners should pay much attention to such factors when performing, designing, and planning experiments.
(2) **Changepoints for CPU and memory can be successfully detected.** Because CPU and memory changepoints have fewer factors they depend on it is likely that they are easier to predict. With disks, where a multitude of factors concur, such as *disk_size*, *iodepth*, and *disk_model*, it is more difficult to predict changepoints. It is therefore important to dig deeper into understanding I/O variability and leave this for future work. However, we encourage practitioners to collect as many metrics as possible [26] to collect sufficient data for future tools to comb through for performance phenomena.
(3) **Prediction accuracy is good for CPU and memory and is comparable for all the methods we tested.** We achieved very good accuracy for CPU and memory changepoints, and the performance was comparable for all three frameworks we used—XgBoost, LightGBM, CatBoost. Therefore, experimenters and performance analysts can use any of these to predict performance.
(4) **Changepoints can be predicted less accurately for *breakout detection.*** Because *breakout detection* is finer-grained in detecting performance changepoints, prediction methods achieve a lower accuracy in detecting these. However, CPU and memory accuracy is reasonably good for *breakout detection*. Therefore, practitioners who need a more sensitive changepoint detection method can still make use of our prediction method without much loss in prediction performance.

## 5 RELATED WORK

We discuss related work and contrast them with the approach we take in this paper. Hirade and Yoshizumi devise an algorithm [11] for changepoint prediction based on the method of ensemble learning. They assume the occurs of changepoints can be characterized by the time intervals between the changepoints, and their symptoms though the causes for changepoints are different. Then they generate weak classifiers to extract the property of the reasons for changepoints and ensemble these weak classifiers into a novel classifier [11]. Unlike metric in our experiment, they evaluate the performance of their algorithm with F-measure on a driving simulator dataset. Agudelo-Espana et al. propose a method from a Bayesian perspective and which is an extension of the Bayesian Online Change Point Detection (BOCPD) algorithm [1] to predict when the next changepoint will occur. The main idea of BOCPD is a probability distribution over the run length. Run length is a non-negative discrete metric that denotes the number of time steps passed at time t since the last changepoint. They evaluate their algorithm on a mice sleep staging dataset. Xu et al. focus on the prediction of high performance computing system's IO variability [28]. They investigate response surface models, Gaussian process based methods, inverse distance weighting methods, and nonparametric regression models and evaluate the performance of these approaches by the prediction accuracy. In this paper, we use gradient boosting models, which are ensemble learning models as well, to predict performance changepoints for cloud computing resources. We used large sets of features, characterize which features are important for individual resources, and make a comparison between multiple prediction models.

## 6 CONCLUSION

Cloud performance variability is a significant problem that affects modern clouds. As opposed to variability given by noisy neighbors or interaction with cloud providers which have quality of service policies, variability given by changepoints can be predicted. The ability to predict such performance changepoints can be leveraged by practitioners for tuning and updating their benchmark pipelines. In this article, we describe how we apply two methods for changepoint detection—*robseg* and *breakout detection*—in the analysis of a large-scale dataset of measurements collected on the CloudLab testbed. As we study many performance time series, we tune *breakout detection* in order to achieve comparable results to the *robseg*'s

results. We then present a method for predicting performance variability given by changepoints. We use gradient boosting models to predict the time when a changepoint occurs, and our methods offer good prediction quality. We show what features are most important in predicting CPU, memory, and disk performance changepoints. Finally, we offer advice for practitioners on how to leverage our results.

## ACKNOWLEDGMENTS

The work in this article was funded via NWO Veni VI.202.195. This research is also supported by the National Science Foundation, Grant Number 1743363.

## REFERENCES

[1] Diego Agudelo-España, Sebastian Gomez-Gonzalez, Stefan Bauer, Bernhard Schölkopf, and Jan Peters. 2020. Bayesian Online Prediction of Change Points. In *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI) (Proceedings of Machine Learning Research, Vol. 124)*, Jonas Peters and David Sontag (Eds.). PMLR, 320–329. http://proceedings.mlr.press/v124/agudelo-espana20a.html

[2] Zhen Cao, Vasily Tarasov, Hari Prasath Raman, Dean Hildebrand, and Erez Zadok. 2017. On the Performance Variation in Modern Storage Stacks. In *Proceedings of the 15th Usenix Conference on File and Storage Technologies* (Santa clara, CA, USA) *(FAST'17)*. USENIX Association, USA, 329–343.

[3] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[4] Jeffrey Dean and Luiz André Barroso. 2013. The Tail at Scale. *Commun. ACM* 56 (2013), 74–80. http://cacm.acm.org/magazines/2013/2/160173-the-tail-at-scale/fulltext

[5] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 1–14. https://www.usenix.org/conference/atc19/presentation/duplyakin

[6] D. Duplyakin, A. Uta, A. Maricq, and R. Ricci. 2020. In Datacenter Performance, The Only Constant Is Change. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. 370–379. https://doi.org/10.1109/CCGrid49817.2020.00-56

[7] Paul Fearnhead and Guillem Rigaill. 2016. Changepoint Detection in the Presence of Outliers. *J. Amer. Statist. Assoc.* (09 2016). https://doi.org/10.1080/01621459.2017.1385466

[8] Jerome H. Friedman. 2000. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* 29 (2000), 1189–1232.

[9] Haryadi Gunawi, Caitie McCaffrey, Deepthi Srinivasan, Biswaranjan Panda, Andrew Baptist, Gary Grider, Parks Fields, Kevin Harms, Robert Ross, Andree Jacobson, Robert Ricci, Riza Suminto, Kirk Webb, Peter Alvaro, Hakizumwami Runesha, Mingzhe Hao, Huaicheng Li, Russell Sears, Casey Golliher, and Nematollah Bidokhti. 2018. Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems. *ACM Transactions on Storage* 14 (10 2018), 1–26. https://doi.org/10.1145/3242086

[10] T. Hastie, R. Tibshirani, and J.H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer. https://books.google.nl/books?id=eBSgoAEACAAJ

[11] R. Hirade and T. Yoshizumi. 2012. Ensemble learning for change-point prediction. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. 1860–1863.

[12] N. A. James, A. Kejariwal, and D. S. Matteson. 2016. Leveraging cloud data to mitigate user experience from 'breaking bad'. In *2016 IEEE International Conference on Big Data (Big Data)*. 3499–3508. https://doi.org/10.1109/BigData.2016.7841013

[13] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 3146–3154. http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf

[14] Rebecca Killick and Idris Eckley. 2014. Changepoint: An R Package for Changepoint Analysis. *Journal of statistical software* 58 (06 2014). https://doi.org/10.18637/jss.v058.i03

[15] R. Killick, P. Fearnhead, and I. A. Eckley. 2012. Optimal Detection of Changepoints With a Linear Computational Cost. *J. Amer. Statist. Assoc.* 107, 500 (2012), 1590–1598. http://www.jstor.org/stable/23427357

[16] Philipp Leitner and Jürgen Cito. 2016. Patterns in the Chaos—A Study of Performance Variation and Predictability in Public IaaS Clouds. *ACM Trans. Internet Technol.* 16, 3, Article 15 (April 2016), 23 pages. https://doi.org/10.1145/2885497

[17] Robert Maidstone, Toby Hocking, Guillem Rigaill, and Paul Fearnhead. 2017. On Optimal Multiple Changepoint Algorithms for Large Data. *Statistics and Computing* 27, 2 (March 2017), 519–533. https://doi.org/10.1007/s11222-016-9636-3

[18] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. 2018. Taming Performance Variability. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA) *(OSDI'18)*. USENIX Association, USA, 409–425.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[20] Guillem Rigaill. 2015. A pruned dynamic programming algorithm to recover the best segmentations with 1 to $K_{max}$ change-points. arXiv:1004.0887 [stat.CO]

[21] Guillem Rigaill. 2019. Fpop implementation for robust losses. https://github.com/guillemr/robust-fpop.

[22] A. Scott and M. Knott. 1974. A Cluster Analysis Method for Grouping Means in the Analysis of Variance. *Biometrics* 30 (1974), 507.

[23] A. F. M. Smith. 1975. A Bayesian Approach to Inference about a Change-Point in a Sequence of Random Variables. *Biometrika* 62, 2 (1975), 407–416. http://www.jstor.org/stable/2335381

[24] Gabor Szekely and Maria Rizzo. 2005. Hierarchical Clustering via Joint Between-Within Distances: Extending Ward's Minimum Variance Method. *Journal of Classification* 22 (02 2005), 151–183. https://doi.org/10.1007/s00357-005-0012-9

[25] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. 2020. Is Big Data Performance Reproducible in Modern Cloud Networks?. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 513–527. https://www.usenix.org/conference/nsdi20/presentation/uta

[26] Alexandru Uta, Kristian Laursen, Alexandru Iosup, Paul Melis, Damian Podareanu, and Valeriu Codreanu. 2020. Beneath the SURFace: An MRI-like View into the Life of a 21st-Century Datacenter. *login Usenix Mag.* 45, 3 (2020).

[27] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. 2018. CatBoost: gradient boosting with categorical features support. *arXiv e-prints*, Article arXiv:1810.11363 (Oct. 2018), arXiv:1810.11363 pages. arXiv:1810.11363 [cs.LG]

[28] Li Xu, Thomas Lux, Tyler Chang, Bo Li, Yili Hong, Layne Watson, Ali Butt, Danfeng Yao, and Kirk Cameron. 2020. Prediction of High-Performance Computing Input/Output Variability and Its Application to Optimization for System Configurations. arXiv:2012.07915 [cs.DC]