Journal of COMPUTER AND FORENSIC SCIENCES

CFS

ISSN 2956-087X
e-ISSN 2956-0799

Volume 1 • Issue 1 • 2022

UNIVERSITY OF CRIMINAL INVESTIGATION AND POLICE STUDIES
Belgrade

UNIVERSITY OF CRIMINAL INVESTIGATION AND POLICE STUDIES, BELGRADE – THE REPUBLIC OF SERBIA
KRIMINALISTIČKO-POLICIJSKI UNIVERZITET, BEOGRAD – REPUBLIKA SRBIJA

CFS

CFS. JOURNAL OF COMPUTER AND FORENSIC SCIENCES

Belgrade, 2022

PUBLISHER

University of Criminal Investigation and Police Studies, Cara Dušana 196, 11080 Belgrade, Serbia

Editor-in-Chief

Prof. Milan Gnjatović, PhD, University of Criminal Investigation and Police Studies, Belgrade, Serbia

Editor for Computer Sciences

Prof. Brankica Popović, PhD, University of Criminal Investigation and Police Studies, Belgrade, Serbia

Editor for Forensic Sciences

Prof. Smilja Teodorović, PhD, University of Criminal Investigation and Police Studies, Belgrade, Serbia

English Language Editor and Proofreader

Jelena Pandža, University of Criminal Investigation and Police Studies, Belgrade, Serbia

Journal Manager

Jelena Pandža, University of Criminal Investigation and Police Studies, Belgrade, Serbia

Text Layout

Jovan Pavlović, University of Criminal Investigation and Police Studies, Belgrade, Serbia

Printed by

Službeni glasnik, Belgrade

Impression

100 copies

TABLE OF CONTENTS

EDITORIAL

CFS 2022, Vol. 1, Issue 1, pp. 3–4
Editorial

EDITORIAL

At the Foot of the Mountain

Milan Gnjatović
Editor-in-Chief
University of Criminal Investigation and Police Studies, Belgrade;
milan.gnjatovic@kpu.edu.rs

Published: 2022-12-30

Some people say that starting a newspaper is a Herculean task. I am not sure that it holds for scientific journals. To me, it is much more of a Sisyphean task, and a scientific editor is a somewhat absurd Sisyphus-like figure rolling a rock to the top of a mountain in an indifferent universe. And here we are, standing at the foot of the mountain. So how to proceed? The words of Albert Camus seem appropriate: "The struggle itself toward the heights is enough to fill a man's heart. One must imagine Sisyphus happy." [1]

So let us maintain that assumption for now.

It is indeed a pleasure to launch the Journal of Computer and Forensic Sciences and to welcome you to this inaugural issue. However, the launch of a scientific journal is not a one-man show. With this in mind, I wish to thank the Editorial Board members, the editorial team members, and the Rector's team at the University of Criminal Investigation and Police Studies in Belgrade for their support in launching this journal. But before and above all, I thank the contributing authors and reviewers for their extraordinary effort and dedication in preparing this inaugural issue. Their involvement is gratefully acknowledged and highly appreciated.

The Journal of Computer and Forensic Sciences is an open access, peer-reviewed scientific journal published by the University of Criminal Investigation and Police Studies in Belgrade, covering advanced and innovative research across the fields of computer and forensic sciences. The aim of the journal is to provide a platform through which authors can communicate their viewpoints on diverse but often related aspects of computer and forensic sciences and a source of information to support advancing research, education, and practice in these fields. I invite you to learn more about the journal at cfs.kpu.edu.rs.

Inaugural editorials typically share some elaborate thoughts about the envisioned path for the journal. I will express my vision for the Journal of Computer and Forensic Sciences in a rather brief manner: I hope that you will find it a useful platform for presenting your research and a useful source of inspiration for expanding your research.

With this in mind, I would like to draw the reader's attention to the content of this inaugural issue. It contains five original research articles in the field of computer science:

The first article [2] introduces an approach to evaluation of spoken words based on lip movements without accompanying sound signals based on convolutional neural networks.

The second article [3] introduces a method for detecting non-technical losses, commercial losses, and electricity theft based on big data analysis for distribution networks.

The third article [4] reports on a comparative performance evaluation of three relevant types of suboptimal binary search trees, i.e., Adelson-Velsky and Landis trees, red-black trees, and splay trees.

The fourth article [5] discusses a novel hybrid computer architecture based on an asymmetric multicore processor, combining the control-flow and data-flow concepts on the same chip die.

The last article [6] reports on an approach to mathematical modeling of the file system performance in a hypervisor-based virtual environment, with special focus on the file system pair interactions.

Thank you for reading the Journal of Computer and Forensic Sciences. Our editorial team look forward to receiving your contributions.

REFERENCES

[1] A. Camus, *The Myth of Sisyphus and Other Essays,* New York: Alfred A. Knopf, 1955.

[2] D. Perić, N. Maček, and M. Bogdanoski, "Application of Convolutional Neural Networks to Spoken Words Evaluation Based on Lip Movements without Accompanying Sound Signal," *Journal of Computer and Forensic Sciences*, vol. 1, no. 1, pp. 7–16, 2022.

[3] M. Žarković and G. Dobrić, "Non-Technical Losses Detection in Power System," *Journal of Computer and Forensic Sciences*, vol. 1, no. 1, pp.17–28, 2022.

[4] S. Štrbac-Savić, M. Tomašević, N. Maček, and Z. Minchev, "Comparative Performance Evaluation of Suboptimal Binary Search Trees," *Journal of Computer and Forensic Sciences*, vol. 1, no. 1, pp. 29–45, 2022.

[5] N. Korolija and K. Milfeld, "Towards Hybrid Supercomputing Architectures," *Journal of Computer and Forensic Sciences*, vol. 1, no. 1, pp. 47–54, 2022.

[6] B. Đorđević, V. Timčenko, N. Maček, and M. Bogdanoski, "Performance Modeling of File System Pairs in Hypervisor-Based Virtual Environment Applied on KVM Hypervisor Case Study," *Journal of Computer and Forensic Sciences*, vol. 1, no. 1, pp. 55–76, 2022.

ORIGINAL RESEARCH PAPERS

CFS 2022, Vol. 1, Issue 1, pp. 7–16
https://doi.org/10.5937/1-42696
Original research paper

Application of Convolutional Neural Networks to Spoken Words Evaluation Based on Lip Movements without Accompanying Sound Signal

Dušan Perić[1], Nemanja Maček,[2*] and Mitko Bogdanoski[3]

[1] DP SOFTWARE, Belgrade, Serbia; dusan.peric98@gmail.com

[2] Academy of Technical and Art Applied Studies, School of Electrical and Computer Engineering, Belgrade, Serbia & University Business Academy in Novi Sad, Serbia & SECIT Security Consulting, Serbia; macek.nemanja@gmail.com

[3] Military Academy General Mihailo Apostolski, Skopje, NR Macedonia; mitko.bogdanoski@ugd.edu.mk

* Corresponding author: macek.nemanja@gmail.com

Received: 2022-11-14 • Accepted: 2022-12-05 • Published: 2022-12-30

Abstract: This paper proposes an approach to evaluate spoken words based on lip movements without accompanying sound signals using convolutional neural networks. The main goal of this research is to prove the efficiency of neural networks in the field, where all data is received from an array of images. The modeling and the hypotheses are validated based on the results obtained for a specific case study. Our study reports on speech recognition from only a sequence of images provided, where all crucial data and features are extracted, processed, and used in a model to create artificial consciousness.

Keywords: machine learning; convolutional neural networks; lip reading.

1. INTRODUCTION

For centuries, people have been searching for a solution to the problems that everyday life brings them. Today, human civilization increasingly relies on artificial intelligence systems. It has become our present and has great potential to change the world, improve people's lives in various areas, and respond to numerous social needs in education, medicine, agriculture, economy. A couple of decades ago, artificial intelligence was reserved only for sci-fi movies, and today there are great technological achievements that we employ every day, such as smartphones with applications that help us in our daily activities, smart homes and home appliances, electronic autonomous cars with various integrated intelligent systems, as well as numerous applications on the Internet. As the application of artificial intelligence grew, the challenges it solved became more and more complex. One such challenge is the software's ability to read people's lips and thus bring this rare skill closer to a large number of users.

Citation: D. Perić, N. Maček, and M. Bogdanoski, "Application of Convolutional Neural Networks to Spoken Words Evaluation Based on Lip Movements without Accompanying Sound Signal," *Journal of Computer and Forensic Sciences*, 1(1), https://doi.org/10.5937/1-42696. Copyright: © 2022 by the University of Criminal Investigation and Police Studies in Belgrade. All rights reserved.

People with impaired hearing, as well as people who work in a very noisy environment, most often communicate visually, using sign language or lip-reading techniques. For both methods of communication, the key role is played by the sense of sight, which accepts information and the movements of hands or lips and converts them into a series of images at a one-time interval.

Speech reading is a complex psychophysiological process in which three moments are important: the visual perception of oral movements, the kinesthetic memorization of speech movements, and the psychological act of recognizing a word in order to better understand it [1, 2].

Lip reading can best be described as the visual segmentation of words into time sequences nested within a time period. The application of lip reading can also be seen as an inevitable skill in all spy movies, as well as in some renowned television programs such as BBC News, where, with the help of lip reading techniques, artificial intelligence recognizes the speech from the lips and prints the subtitle of the speaker.

In this paper, we will demonstrate the creation of software that will solve the problem of lip reading with great precision and make this technique available to more users.

2. RELATED WORK

Aggravating circumstances when reading lips are a very common problem in people, and these are: limited movements of the articulator (jaw and lips), fast or slow speech, poor mimicry, specific head movements, inadequate distance, and poor lighting [1, 2]. Many researchers are trying to create the perfect system for lip reading. A lot of related scientific papers can be found addressing this topic. Several approaches related to lip reading are briefly addressed in this paper.

The authors [3] presented the method of detecting lips and using the cropped images as a dataset for the training set for Convolutional Neural Networks. Also, they discussed different methods of evaluation that can be used. In [4], an interesting approach to lip detection by skin color and Kalman image filtering used to eliminate noises and unwanted information from the picture is presented. In Artificial Intelligent systems, effectiveness is a crucial aspect because of the more frequent use of those systems in more and more responsive jobs that were reserved for people only until now. Zhao et al. [5] presented how mutual information maximization facilitates effective lip reading. Real-time speech detection is a hardware-consuming task that can be done with the OpenCV library, which is widespread in movement tracking software. WenJuan et al. [6] presented how lip movement can be detected and used for lip reading. Rahmani and Almasganj [7] showed us a hybrid system for lip reading that uses a combination of image-based and model-based features. A lot of researchers thought that lip reading was only possible with the en face pictures of the face. Saitoh and Konishi [8] presented the solution for training the lip reading system from profile pictures. A detailed explanation of backpropagation, which neural networks are using to learn lip reading, is given by Rathee in [9]. One of the hot topics in the software industry today is mobile applications development. The work of Matsunaga and Matsui [10] depicts how lip reading can be implemented on mobile devices.

3. MATERIALS AND METHODS

Neural networks are well suited for learning from large amounts of raw data, such as images or sequences of images. An image can be represented as a series of pixels, and the task of the neural network is to create useful information from that series of pixels by extracting the features with the greatest weight of information, such as the shape or the edges of the object in the image.

Algorithms based on neural networks have several advantages compared to standard algorithms, but they are dependent on human expertise in the field for which the system is being developed. Of course, all algorithms that use neural networks, due to their nature of using a large amount of information for training the system, have one major drawback, which is the need for very powerful and expensive hardware.

Neurons are the main component of deep learning systems. In them, calculations are performed on the input data that the neuron receives from the previous neuron, and the output parameters are forwarded as a result. Those output parameters are further supplied to the next neuron, except in the case when the neuron is in the first hidden layer. In that case, the input data is raw data on which only pre-processing has been performed.

"In a biological neuron, an electrical signal is given as an output when it receives a more influential input. To capture that functionality in a mathematical model of neuron, we need to have a function that operates on the sum of the input multiplied by the appropriate weights and responds with the appropriate value based on the input. If an input with a higher impact is received, the output should be higher, and if an input with a lower impact is received, the output should be lower, which can be represented as a switch. An activation function is a function that takes a combination of inputs, applies a function over them, and passes an output value." In this way, it tries to imitate the activation/deactivation function. Thus, the activation function determines the state of the neuron by computing the activation function on the combined input [11]. The three activation functions that are far commonly used than the others are "Sigmoid", "Tanh" and "ReLU", which are depicted in Figure 1.

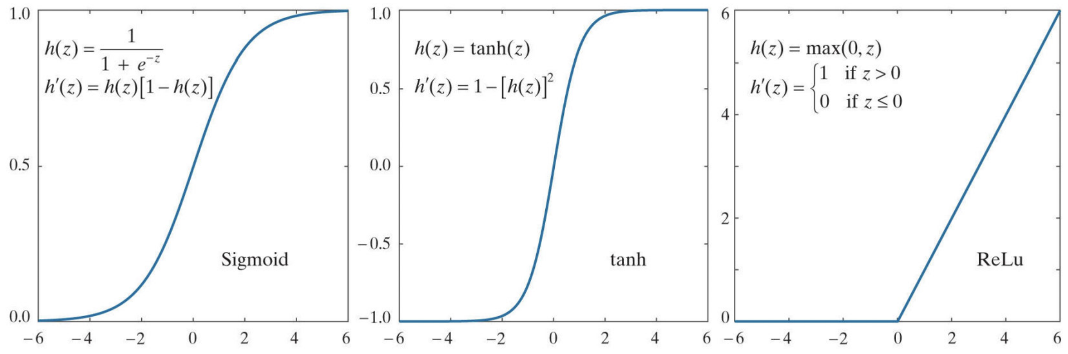


Figure 1. *Activation functions (Sigmoid,Tanh and ReLU). Adopted from [10].*

A neural network consists of one input and one output layer, as well as one or more hidden layers. Depending on the complexity of the task for which the neural network needs to create awareness, the hidden layers can vary.

3.1. Convolutional Neural Networks

Convolutional neural networks, often represented by the abbreviation CNN, are a special type of multilayer neural networks designed to recognize visual patterns directly from images with minimal use of computer resources. There are several ways to connect neurons to create a convolutional neural network. One of the ways is the feed-forward network method [12]. The method is based on the fact that neurons from each layer feed the next layer with their output values. This process is repeated until we get the final resulting values.

Each convolutional neural network consists of the three most important layers for its functioning, namely the convolutional layer, the pooling layer, and the fully connected layer. The task of the convolutional layer is to extract features from the image that is used as input information, so we can consider it the most important layer because it does most of the calculations in the convolutional neural network. The activation function that performs exceptionally well for CNN is ReLU because of its features that perfectly fit the description of the input data for CNN, which are images. The pooling layer selects an area from the image and then selects one representative value using a maximum or average pooling technique. The architecture of a convolutional neural network can be represented as several convolutional layers connected in a cascade style. In each cascade, there is a convolutional layer with a ReLU activation function, then a pooling layer, thus simulating cascades several times. Finally comes the fully connected layer. The output from each convolutional layer is a set of objects called feature maps, generated by a single convolutional filter. Feature maps can be used to define new input data for the next layer. Thus, convolutional neural networks work by extracting features from the image that is the input information by dragging a filter over it. The result of multiplying the values of the filter matrix and the image area of the same size gives the output values for the next convolutional layer. In this way, in each subsequent step, we get an increasingly precise contour of the object we are looking for.

4. RESULTS

The testing platform consisted of an "upgraded" Dell Precision laptop based on an Intel Ii7 processor with 64 GB RAM, 3.5 terabytes in solid state drives, and an external NVidia 3080 RTX graphic card. We have used common testing platforms like Python and various libraries for visualization and deep learning (like Tensorflow).

Figure 2 depicts a sequence of images that form one word, viewed from left to right and spoken by one person. The optimal image size here is not of any interest. The functions embedded will, according to our algorithm, crop the surface around lips, hence the size is inconsistent. Yet, a bit of preprocessing to crop it to a proper fixed-size rectangular image may provide some help (i.e., an improvement in accuracy, etc.) in certain scenarios.

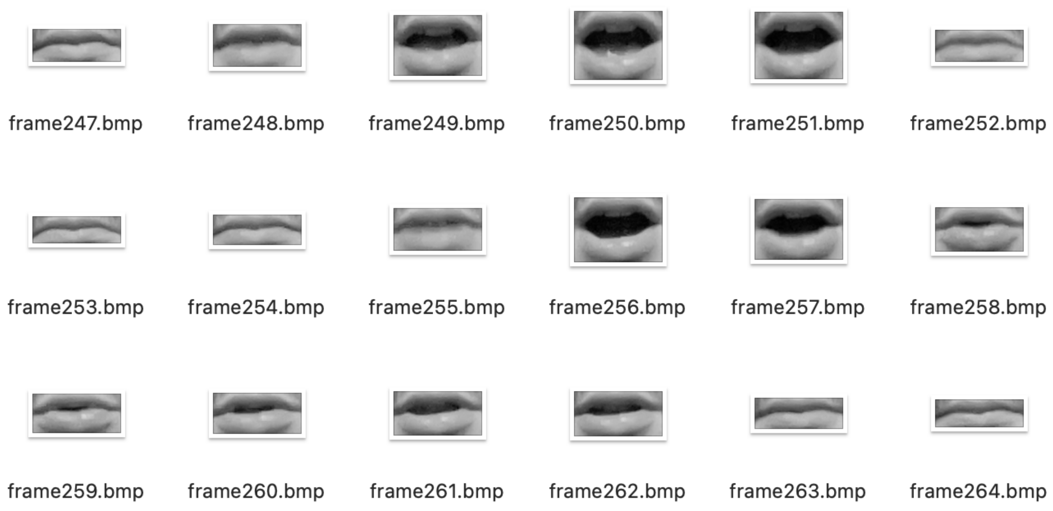
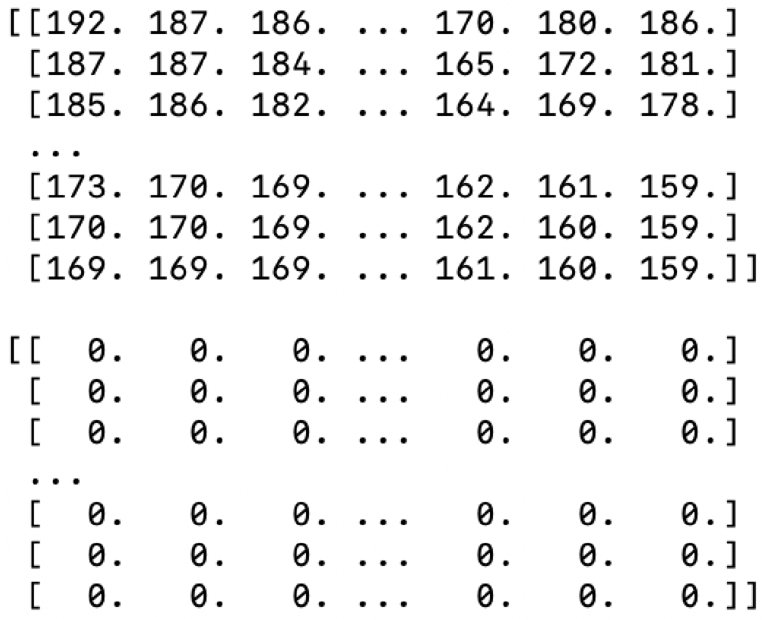


Figure 2. *Sequence of images used to represent one word. Images presented in Figure 2 and all others used in our experiments are generated from several databases [13–15].*

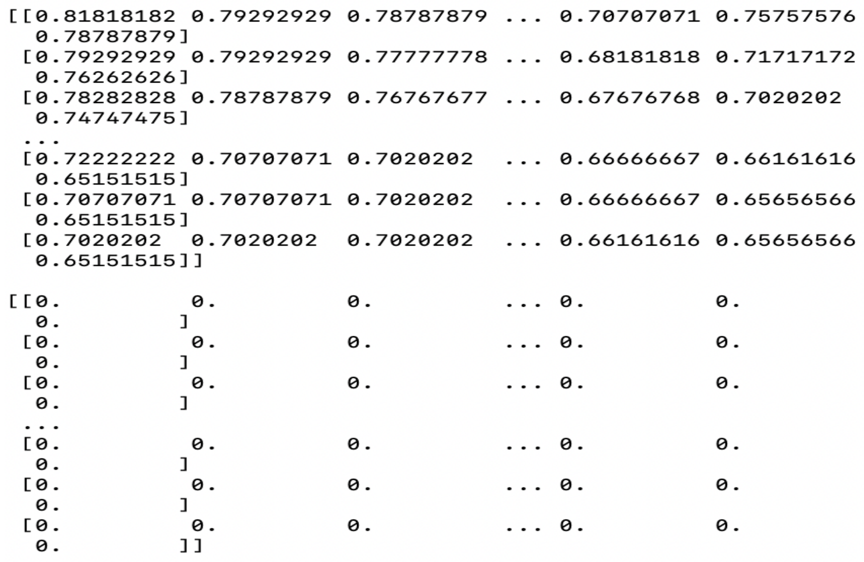
The final step in data preprocessing is data normalization, which we can do by nesting the range of pixel values between 0 and 1. This type of optimization is called “colormap normalization” and is used when pixel values need to be represented by values between 0 and 1.



```
[[192. 187. 186. ... 170. 180. 186.]
 [187. 187. 184. ... 165. 172. 181.]
 [185. 186. 182. ... 164. 169. 178.]
 ...
 [173. 170. 169. ... 162. 161. 159.]
 [170. 170. 169. ... 162. 160. 159.]
 [169. 169. 169. ... 161. 160. 159.]]

[[  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]
 ...
 [  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]]
```

Figure 3. *Data before normalization (data screenshot).*



```
[[0.81818182 0.79292929 0.78787879 ... 0.70707071 0.75757576
  0.78787879]
 [0.79292929 0.79292929 0.77777778 ... 0.68181818 0.71717172
  0.76262626]
 [0.78282828 0.78787879 0.76767677 ... 0.67676768 0.7020202
  0.74747475]
 ...
 [0.72222222 0.70707071 0.7020202  ... 0.66666667 0.66161616
  0.65151515]
 [0.70707071 0.70707071 0.7020202  ... 0.66666667 0.65656566
  0.65151515]
 [0.7020202  0.7020202  0.7020202  ... 0.66161616 0.65656566
  0.65151515]]

[[0.         0.         0.         ... 0.         0.
  0.        ]
 [0.         0.         0.         ... 0.         0.
  0.        ]
 [0.         0.         0.         ... 0.         0.
  0.        ]
 ...
 [0.         0.         0.         ... 0.         0.
  0.        ]
 [0.         0.         0.         ... 0.         0.
  0.        ]
 [0.         0.         0.         ... 0.         0.
  0.        ]]
```

Figure 4. *Data after normalization (data screenshot).*

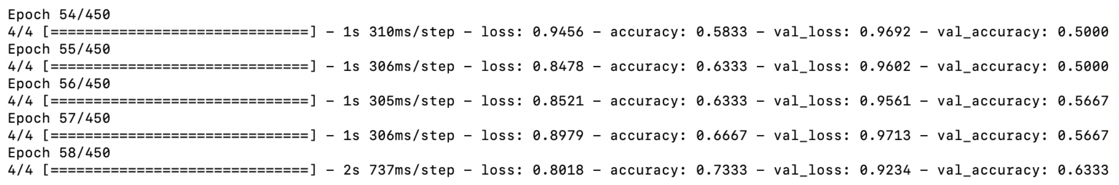
After we have loaded, processed, and normalized our data, it is time to define a learning model that will create awareness from that data. For the purposes of this algorithm, a sequential model composed of layers that can be viewed as groups of neurons was used. The sequential model allows adding different types of layers on top of each other, creating a single stack.

The optimizer is the most important part of our model. Its functionality is to calculate and change the weights of the neurons so that the losses are as small as possible. The back-propagation process we mentioned earlier is actually an optimization algorithm. Two of the most well-known and widely used deep learning optimizers are Stochastic Gradient Descent (SGD) and Adam (Adaptive Moment Estimation).

For outcomes with categories, the prediction will be the name of the class, i.e., the category. In this algorithm, the results are the words that the system should recognize based on the movement of the lips, which indicates that the regressive standard will not be used but a categorical one.

In this research, categorical cross-entropy is used to calculate the losses because there are more than two possible prediction outcomes.

The results of model training through epochs are depicted in Figure 5.



```
Epoch 54/450
4/4 [==============================] - 1s 310ms/step - loss: 0.9456 - accuracy: 0.5833 - val_loss: 0.9692 - val_accuracy: 0.5000
Epoch 55/450
4/4 [==============================] - 1s 306ms/step - loss: 0.8478 - accuracy: 0.6333 - val_loss: 0.9602 - val_accuracy: 0.5000
Epoch 56/450
4/4 [==============================] - 1s 305ms/step - loss: 0.8521 - accuracy: 0.6333 - val_loss: 0.9561 - val_accuracy: 0.5667
Epoch 57/450
4/4 [==============================] - 1s 306ms/step - loss: 0.8979 - accuracy: 0.6667 - val_loss: 0.9713 - val_accuracy: 0.5667
Epoch 58/450
4/4 [==============================] - 2s 737ms/step - loss: 0.8018 - accuracy: 0.7333 - val_loss: 0.9234 - val_accuracy: 0.6333
```

Figure 5. *Model training through epochs (results screenshot).*

We can present the performance of our model visually through two graphs. Figure 6 depicts the accuracy of the model across epochs, while Figure 7 depicts the losses.

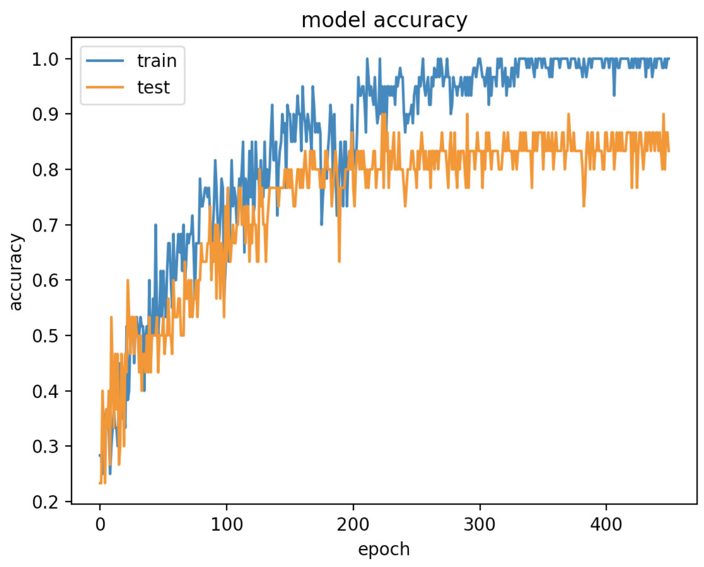


Figure 6. *Model accuracy.*

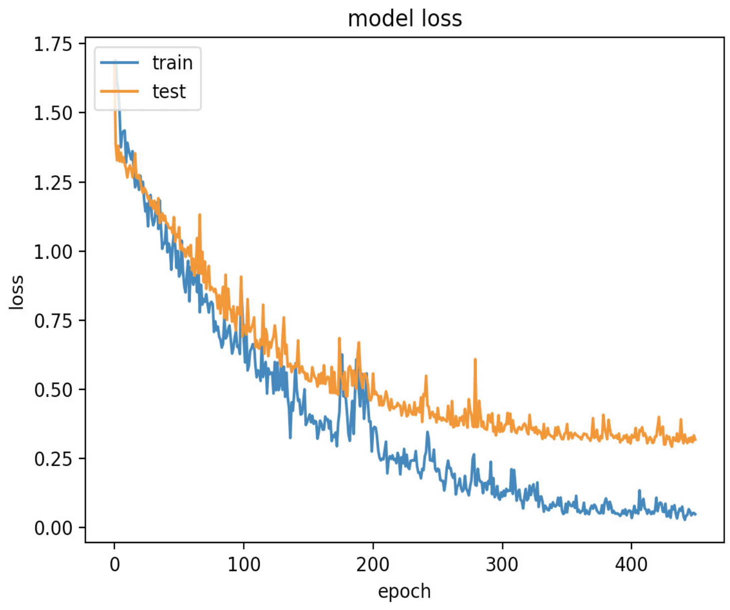
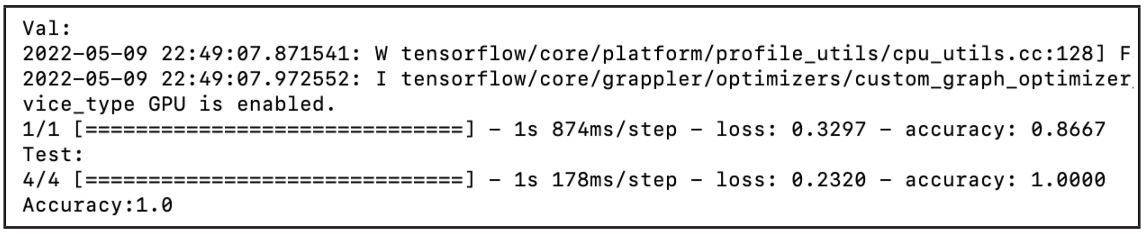


Figure 7. *Model losses.*

The effectiveness of our model is finally presented in the following screenshot:



```
Val:
2022-05-09 22:49:07.871541: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] F
2022-05-09 22:49:07.972552: I tensorflow/core/grappler/optimizers/custom_graph_optimizer
vice_type GPU is enabled.
1/1 [==============================] - 1s 874ms/step - loss: 0.3297 - accuracy: 0.8667
Test:
4/4 [==============================] - 1s 178ms/step - loss: 0.2320 - accuracy: 1.0000
Accuracy:1.0
```

Figure 8. *Model evaluation (results screenshot).*

5. DISCUSSION

Since there is almost no field of human activity where artificial intelligence is not used, modern man is required to master new knowledge and technologies. The purpose of the emergence of artificial intelligence is to facilitate the daily activities of people, which most often require mental fatigue, and to increase the performance of work in jobs that require the processing of a large amount of information. A human is capable of doing that kind of work for a few hours, but a machine, on the other hand, can do it non-stop for days or years.

The software we presented in this paper simulates human performance, which means that it performs the task at the same speed as a human. The main advantage is that the person has to take a break after some time and divide the work into several days, and the software will work non-stop until the task is completed. In this way, the skill of lip reading can be used much more effectively in long videos, as well as in cases where the video does not have a sound recording or the ambient noise is too great to record the sound with sufficient quality. The downside of the software is the same as with all complex neural networks, and that is the large training time of the model. As the number of categories that the model needs to predict increases, the training time also multiplies.

Readers may also consider the following articles for further information on the topic [16–19].

FUNDING:

This research received no external funding. Data used to generate lip images in Figure 2 is generated from publicly available datasets licensed under CC BY-NC 4.0 license.

INSTITUTIONAL REVIEW BOARD STATEMENT:

Not applicable.

INFORMED CONSENT STATEMENT:

Not applicable.

CONFLICTS OF INTEREST:

The authors declare no conflict of interest.

REFERENCES

[1] S. Pažin, L. Isaković, S. Slavnić, and M. Srzić, “Specifičnost čitanja govora sa usana kod gluvih i nagluvih učenika različitog uzrasta,” In Specificity of hearing impairment – new trends, 2020, pp. 219–233.

[2] A. Živanović, I. Sokolovac, M. Marković, S. Suzić, and V. Delić, “Prepoznavanje reči u govornoj audiometriji,” In Specificity of hearing impairment – new trends, 2020, pp. 97–111.

[3] Y. Xiao, L. Teng, A. Zhu, X. Liu, and P. Tian, “Lip Reading in Cantonese,” *IEEE Access*, Vol. 10, 2022, pp. 95020–95029. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9878070

[4] T. Thein and K. M. San, “Lip localization technique towards an automatic lip reading approach for Myanmar consonants recognition,” In 2018 International Conference on Information and Computer Technologies (ICICT), 2018, pp. 123–127.

[5] X. Zhao, S. Yang, S. Shan, and X. Chen, “Mutual information maximization for effective lip reading,” In 2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020), 2020, pp. 420–427.

[6] Y. WenJuan, L. YaLing, and D. MingHui, “A real-time lip localization and tacking for lip reading,” In 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), 2010, Vol. 6, pp. V6–363.

[7] M. H. Rahmani and F. Almasganj, “Lip-reading via a DNN-HMM hybrid system using combination of the image-based and model-based features,” In 2017 3rd International Conference on Pattern Recognition and Image Analysis (IPRIA), 2017, pp. 195–199.

[8] T. Saitoh and R. Konishi, “Profile lip reading for vowel and word recognition,” In 2010 20th International conference on pattern recognition, 2010, pp. 1356–1359.

[9] N. Rathee, “Investigating back propagation neural network for lip reading,” In 2016 International Conference on Computing, Communication and Automation (ICCCA), 2016, pp. 373–376.

[10] Y. Matsunaga and K. Matsui, “Mobile Device-based Speech Enhancement System Using Lip-reading,” In 2018 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET), 2018, pp. 1–4.

[11] J. Moolayil, *Learn Keras for Deep Neural Networks: A Dust-Track Approach to Modern Deep Learning with Python*, 1st edition. Apress, 2018.

[12] M. Sewak, R. Karim, and P. Pujari, *Practical Convolutional Neural Networks: Implement advanced deep learning models using Python*. Packt Publishing, 2018.

[13] J. S. Chung and A. Zisserman, “Lip Reading in the Wild,” In Asian Conference on Computer Vision, 2016. Available: https://www.robots.ox.ac.uk/~vgg/publications/2016/Chung16/chung16.pdf

[14] J. S. Chung, A. Senior, O. Vinyals, and A. Zisserman, “Lip Reading Sentences in the Wild,” In IEEE Conference on Computer Vision and Pattern Recognition, 2017. Available:

https://openaccess.thecvf.com/content_cvpr_2017/papers/Chung_Lip_Reading_Sentences_CVPR_2017_paper.pdf

[15] IPSJ SIG-SLP Noisy Speech Recognition Evaluation WG (2011): Audio-Visual Speech Recognition Evaluation Environment (CENSREC-1-AV). Speech Resources Consortium, National Institute of Informatics. (dataset). https://doi.org/10.32130/src.CENSREC-1-AV

[16] S. Ren, Y. Du, J. Lv, G. Han, and S. He, "Learning from the master: Distilling cross-modal advanced knowledge for lip reading," In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 13325–13333.

[17] F. Eguchi, K. Matsui, Y. Nakatoh, Y. O. Kato, A. Rivas, and J. M. Corchado, "Development of Mobile Device-Based Speech Enhancement System Using Lip-Reading," In International Symposium on Distributed Computing and Artificial Intelligence, 2021, pp. 210–220.

[18] P. Ma, Y. Wang, J. Shen, S. Petridis, and M. Pantic, "Lip-reading with densely connected temporal convolutional networks," In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2021, pp. 2857–2866.

[19] K. R. Prajwal, T. Afouras, and A. Zisserman, "Sub-word level lip reading with visual attention," In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 5162–5172.

Journal of COMPUTER AND FORENSIC SCIENCES

CFS 2022, Vol. 1, Issue 1, pp. 17–28
https://doi.org/10.5937/1-42707
Original research paper

Non-Technical Losses Detection in Power System

Mileta Žarković[1]* and Goran Dobrić[1]

[1] University of Belgrade, School of Electrical Engineering, mileta@etf.rs, dobric@etf.rs

* Corresponding author: mileta@etf.rs

Received: 2022-09-07 • Accepted: 2022-11-22 • Published: 2022-12-30

Abstract: The digitization of distribution networks enables the collection of big data from which it is necessary to draw conclusions and detect anomalies among electricity consumers. This paper explains methodologies to detect non-technical losses, commercial losses, and electricity theft. Based on monthly electricity consumption measurements, possible and prevalent cases of anomalies and theft among consumers are identified. Indicators that can detect anomalies have been proposed for such types of load diagrams. The sensitivity of the indicators to different types of consumers was analyzed. The applicability of this methodology was examined for a set of real measurements, and its advantages were pointed out. This concept represents a good recommendation, as it is possible to observe and detect irregularities in electricity consumption.

Keywords: non-technical losses (NTL), smart meter, energy theft detection, power system.

INTRODUCTION

The goal of every energy system is the safe transmission and distribution of energy with minimal losses. Losses in the distribution electricity network can be divided into technical and non-technical losses. Technical losses are calculated using well-known methods, while non-technical losses can't be easily and clearly detected and evaluated. Non-technical or commercial losses are the result of measurement inaccuracy, incomplete readings of metering devices, non-simultaneous readings, improper control of metering points, irregular meter calibration, untimely detection of unauthorized consumption, insufficient technical equipment of teams to work on customer control, insufficient training of readers and controllers of metering devices, insufficient support and assistance (of the law) after the detection of unauthorized consumption, unauthorized use of electricity on various grounds of unregistered consumption (electricity theft from existing customers and "wild" connections of new customers), error in the operation of measuring devices (delay in balancing customers' meters, malfunctions of meters and measuring transformers), and error in the reading and calculation of electricity. [1]

The review papers [2–5] presented the possibilities of applying a large number of measurements at Distribution System Operators (DSO) and smart grid in order to detect non-technical losses, commercial losses, and electricity theft. Reference [6] presents an intelligent energy meter that provides solution for maintaining power quality, provides superior metering and billing system, and also controls power theft. The research paper

[7] presents prevention of power theft in distribution system using smart hardware device. Three latest gradient boosting classifiers are used for smart grid energy theft identification in the paper [8]. A data analytic approach with false data injection is present in paper [9]. An IoT solution for electricity theft prevention is presented in [10]. Two methodologies based on artificial intelligence present stacked sparse denoising autoencoder [11] and support vector machine [12] for electricity theft detection.

The detection of non-technical losses, commercial losses, and electricity theft is not defined by international standards or internal standards and recommendations of the DSO. In all DSOs within the framework of Industry 4.0, mass digitization is launched, and as many data as possible is collected. In research papers, possible methods of detection and examples of detection of energy losses and electricity theft are presented. However, a comprehensive method and a clear algorithm for detection have not been defined.

This paper provides explanations related to the measurement of electricity consumption. The main load diagrams, energy consumption, as well as main parameters are explained, and multi-day load charts are analyzed. Based on that, the parameters that are important for the detection of anomalies in electricity consumption were observed: coefficient of variation, ratio between peak and valley load, load rate, valley coefficient, daily load variance, and equivalent time. The mentioned parameters are calculated for a normal energy consumption state and states with different types of anomalies: peak random anomaly, on-off anomaly, time random anomaly, and anomaly of decreasing peak. For these types of anomalies, the parameter bounding values are calculated. The algorithm is proposed to detect anomalies based on the analysis of the given parameters.

The paper is organized as follows. Section 2 presents the concept of smart grid and smart meters. In section 3, a load diagram is presented and an overview of the most significant parameters is given. Section 4 shows the results of electricity measurement for consumers where there are no anomalies and for specific consumers where there are special cases of anomalies. Section 5 provides the algorithm for detection and threshold values for parameters with results. Conclusions of the research are derived in Section 6.

SMART GRID AND SMART METERS

A smart grid is a power grid that uses analog and digital information and communication technologies in order to increase the reliability of electricity supply. One of the main components of the smart grid is definitely the communication network and accompanying sensors and measuring devices. Two-way communication between a utility and its consumers is a prerequisite for the smart grid. This communication enables advanced metering and control options, and it is known as the Advanced Metering Infrastructure (AMI). In addition to AMI, data management is very important. Namely, when big data is collected, it has to be properly stored and utilized. Data can be used for many purposes, and detection of electricity theft (non-technical losses) is one of them. Currently, AMI provides physical and wireless connections, bidirectional metering and billing, data storage and management, detection and diagnostics of system faults, and end-to-end communication.

Some of the most important goals of AMI implementation are:

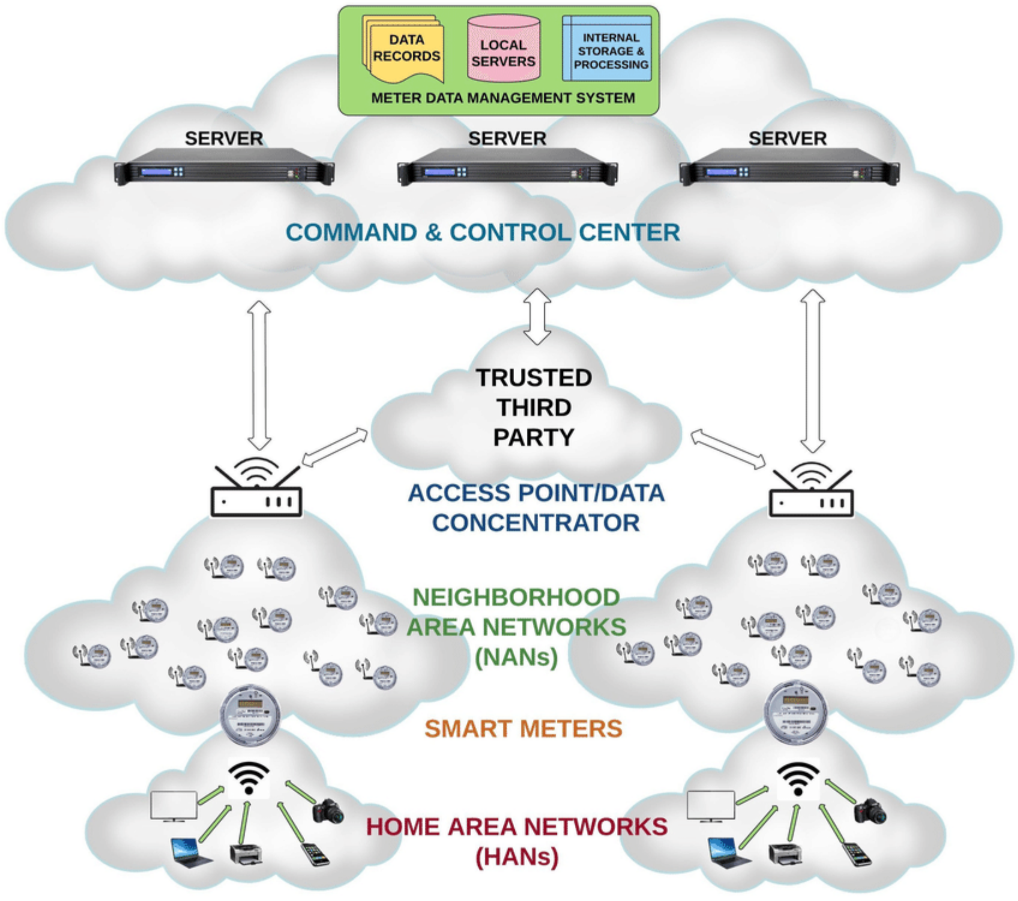

DATA RECORDS
LOCAL SERVERS
INTERNAL STORAGE & PROCESSING
METER DATA MANAGEMENT SYSTEM
SERVER
SERVER
SERVER
COMMAND & CONTROL CENTER
TRUSTED THIRD PARTY
ACCESS POINT/DATA CONCENTRATOR
NEIGHBORHOOD AREA NETWORKS (NANs)
SMART METERS
HOME AREA NETWORKS (HANs)

Figure 1. *Architecture of AMI.*

- Reduction of meter reading expenses;
- Increasing the accuracy of measurements;
- Expediting billing;
- Enabling centralized control of customers;
- Reducing non-technical losses; and
- Increasing network reliability.

The main parts of AMI are smart meters, data concentrators, and data management centers. Figure 1 shows the simple architecture of the AMI system.

POWER LOAD DIAGRAM AND INDICATORS

The daily load diagram presents dependence between power and time. Talking about time, load diagrams appear as daily, weekly, monthly, and yearly charts. The basis of all these diagrams is the daily load diagram, whose shape depends on several factors: the nature of the consumer area, the share of individual consumers in a certain consumer group area, the season (summer, winter), and other factors [1]. The daily load diagram can be estimated as the average measured value within 15, 30, or 60 minutes. It is characterized by three basic indicators: maximum daily load (*Pmax* [kW]), minimum daily load (*Pmin* [kW]), and total daily consumed energy (W [kWh]). Other characteristic indicators are defined from the basic indicators:

$$P_{mean} = \frac{W}{24} \tag{1}$$

$$m = \frac{W}{24P_{max}} \tag{2}$$

$$T = \frac{W}{P_{max}} \tag{3}$$

$$n = \frac{P_{min}}{P_{max}} \tag{4}$$

where P_{mean} is the daily mean load, *m* is the daily load factor , *T* is the maximum power utilization time, and *n* is the ratio of daily minimum and maximum.

Measurement data was taken from more than a thousand smart meters in a part of a distribution network that comprises mostly households. According to the collected data, seven-day hourly diagrams are given in Fig. 2.

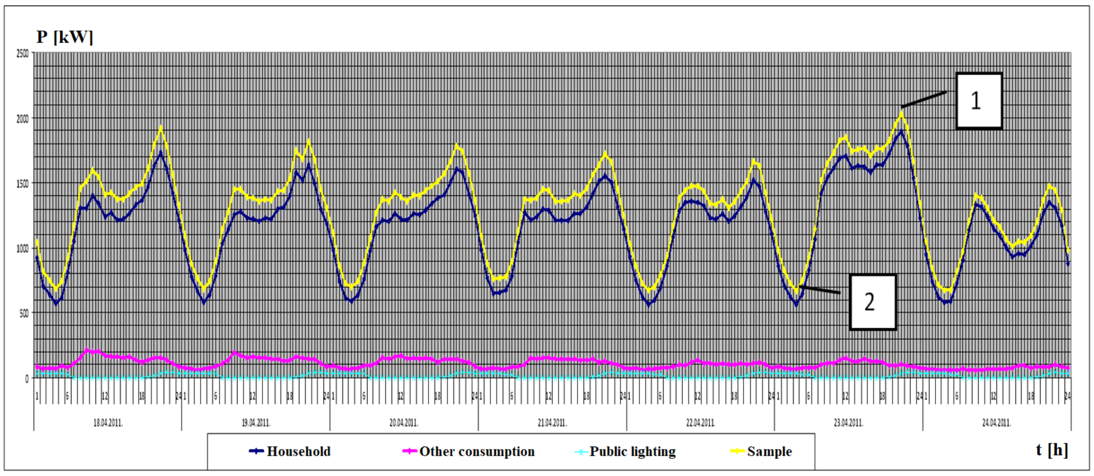


Figure 2. *Weakly load diagrams.*

For the sampled data, the highest daily maximum load for consumption at the sample level is 2027.64 kW (point 1), and the lowest daily maximum is 1474.89 kW, with a relative ratio of 0.73. The mean power value at the sample level is 1295.75 kW, while the relative ratio of minimum and maximum power is 0.33.

The problem with DSO is that the digitization of the network is not complete, and a large number of consumers do not have smart meters. Because of that, data is available only with monthly consumption readings. It is necessary to detect anomalies in the data collected in this way, and this is a specific problem. Figure 3 shows the monthly energy consumption over a period of seven years. In that case, it is necessary to observe the annual load diagram shown in Fig. 4. Such an annual load diagram can be compared with previous annual diagrams and diagrams of neighboring consumers in the same category. The diagram in Fig. 4 refers to three arbitrary users, and it is concluded that the diagram must be normalized with the maximum energy or mean energy over the observed time interval. Data processing was performed in Matlab [13].

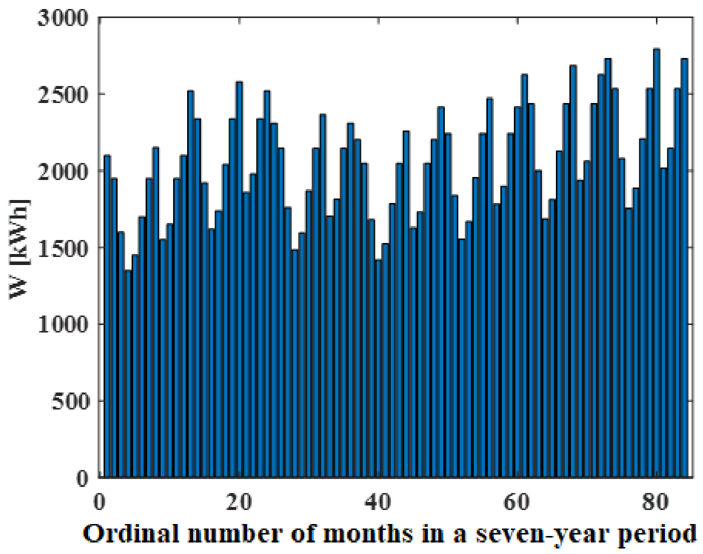


Figure 3. *Monthly load diagrams.*

In Fig. 4 (the normalized diagram), different load failures and different user behaviors can be observed. For these reasons, quantifiers were calculated for a set of 237 users. In order to observe the monthly consumption of different consumers, new quantifiers, shown in Table 1, will be introduced. The behaviors and values of the quantifiers depend on the load diagram. Excessive quantifier deviation for some users from other users will indicate the existence of an anomaly. This analysis is present in Section 5.

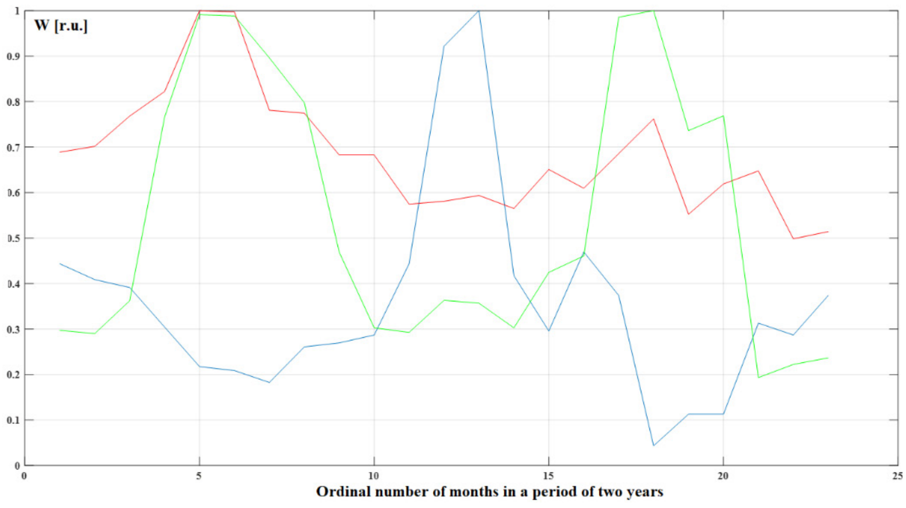


Figure 4. *Normalized monthly load diagrams for different users in the same part of the distribution network.*

Table 1. *Statistical indicators for anomaly detection.*

Indicator	Description code
Coefficient of variation	a1=Pmax./(Pmin+eps)
Ratio between peak and valley load	a2=Pmin./Pmax
Ratio between peak and average load	a3=Pmax./Psr
Valley coefficient	a4=max(P(i+1)-P(i))
Load variance	a5=sum(ΔP(i))
Time of maximum power utilization	T= sum(P’)./Pmax

TYPES OF POWER CONSUMPTION ANOMALIES

Non-technical losses, commercial losses, and electricity theft detection can be done by processing the collected measurement data. The easiest way is to compare the characteristic indicators of load diagrams. If there are load diagrams of customers with and without electricity theft, it is possible to create indicative critical values for indicators. If there are no measurements from consumers with regard to theft, then there is the possibility of creating different diagrams with certain anomalies. The types of electrical energy theft occurring in DSO can be presented as follows:

1. Multiplying all samples by the same randomly chosen value (lower than one)
2. “On-off” attack in which the consumption is reported as zero during some intervals
3. Multiplying consumption by a random value that varies over time
4. The combination of the second and the third type
5. Multiplying only the peak loads by the same randomly selected value (lower than one).

Computer simulation is used to form all five anomalies from a set of real measurement data. The obtained annual diagrams are given in Fig. 5.

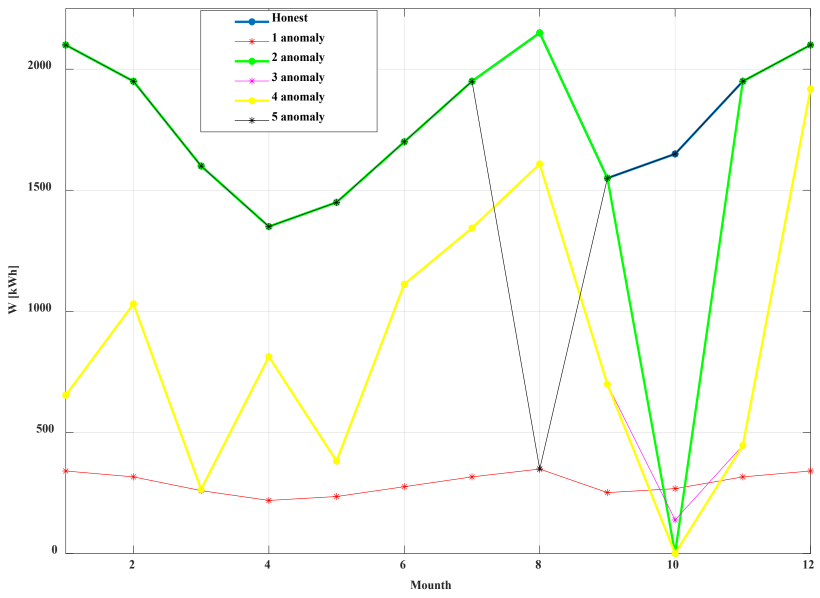


Figure 5. *Comparing the diagrams of the honest consumer and the created anomalies.*

For all six cases from Fig. 5, the indicators from Table 1 are calculated. Fig. 6 presents the values of these parameters.

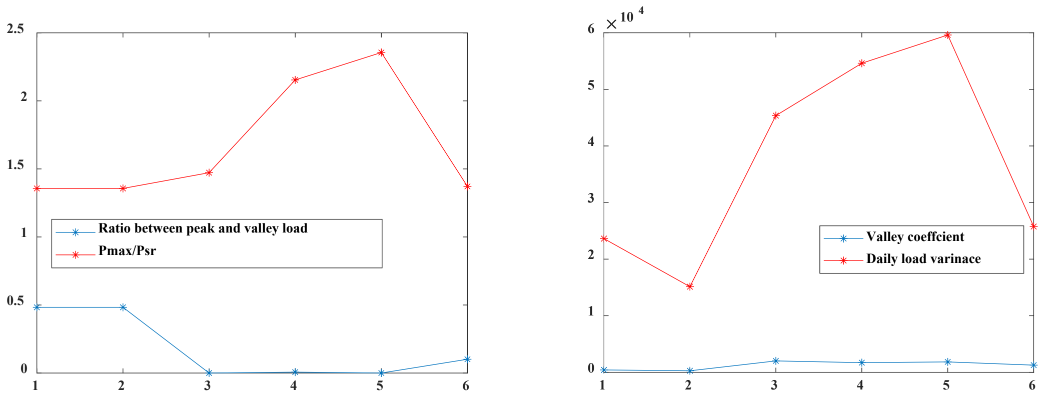


Figure 6. *Comparing the indicators a_2, a_3 (left) and a_4, a_5 (right) in six cases.*

In order to analyze the sensitivity for two different consumers, for two different consumption diagrams (7 a) and b)) the parameters were calculated and plotted. By looking at Fig 7 c), d), e) and f), the behavior of the parameters is very similar for different types of anomalies. Based on that, it can be concluded that it is necessary to further determine their limit values, which would represent triggers for finding anomalies. Table 2 presents exact values, while the bold numbers indicate the limit values of the operating parameters.

Table 2. *Exact vales for statistical indicators for anomaly detection.*

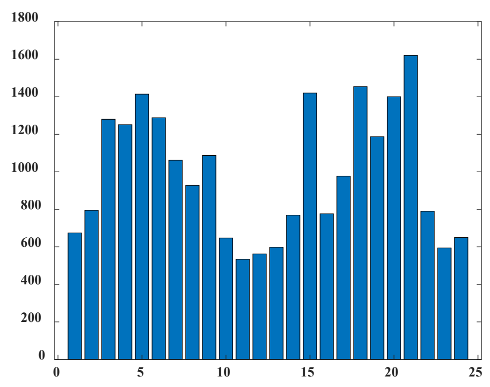
Indicators	Base case	1. anomal.	2. anomal.	3. anomal.	4. anomal.	5. anomal.
a_1	1.2723	1.2723	4.50E+15	22.8547	4.21E+15	**16.5368**
a_2	0.7859	0.7859	0	0.0437	0	**0.0604**
a_3	1.1038	1.1038	1.2034	2.1825	2.3660	**1.1637**
a_4	0.2140	**0.1525**	0.9584	**0.6903**	0.7815	0.8404
a_5	1.5846	**1.1293**	5.0990	7.3465	7.1403	**3.2666**
T	21.7411	21.7412	19.9424	10.9962	10.1435	**20.6222**

RESULTS FOR REAL DATASET

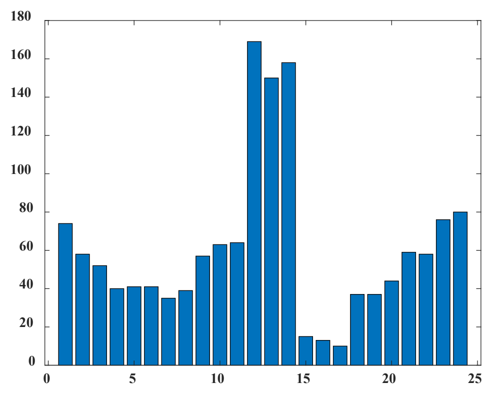
One way to create the limit values for indicator is shown in the previous section. This method involves creating artificial anomalies and monitoring whether consumers will fall into these behaviors and deviate from their daily electricity consumption habits. The second way is to analyze each real load diagram for each consumer from one part of smart grid. This method requires the existence of measurements from all smart meters and data storage. For each consumer and his diagram, the indicators from Table 1 are calculated. Then, for each of the 237 consumers that were considered, the values of those coefficients are observed, and deviations are detected. For that deviation, the limit values are specified and shown in Fig. 8. Table 3 presents the number of users that meet the conditions for each indicator. The cross section of all conditions picks out 5 consumers whose load diagram satisfies all 6 conditions in terms of indicators. So, those five consumers definitely have an anomaly and need to be checked by DSO inspection.

Table 3. *Number of customers with detected anomalies.*

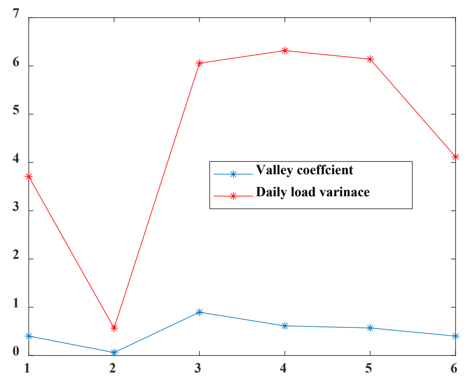
Condition	$a_1 > 20$	$a_2<0.1$	$a_3>2$	$a_4>0.6$	$a_5>4$	$a_6<10$	Cross
Detected customers	15	24	31	20	19	21	**5**



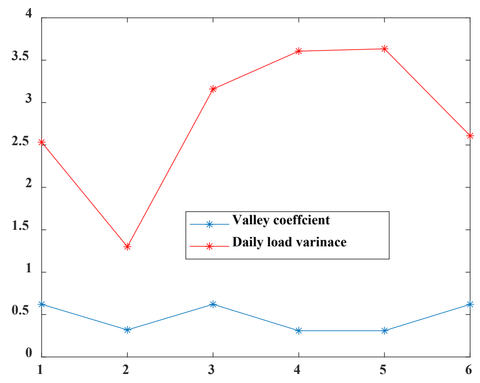
(a)



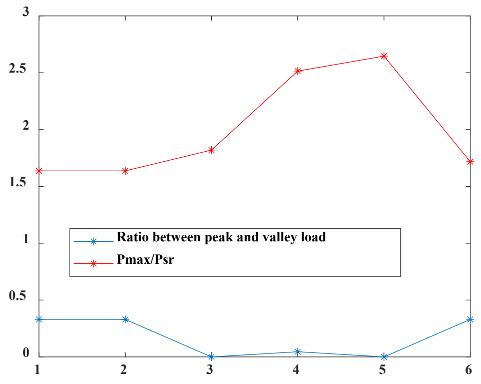
(b)



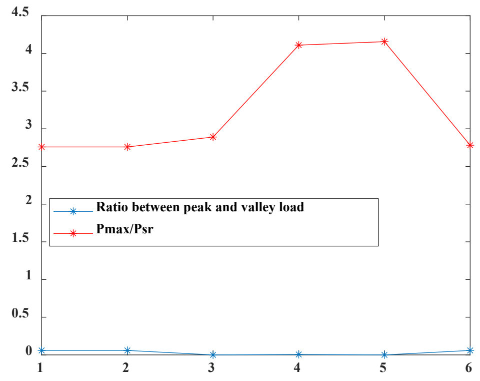
(c)



(d)



(e)



(f)

Figure 7. Statistical indicators calculated based on Table 1: (a) Load diagram for user 1; (b) Load diagram for user 2 (c) a_2 and a_3 for user 1; (d) a_2 and a_3 for user 2 (e) a_4 and a_5 for user 1 (f) a_4 and a_5 for user 2.

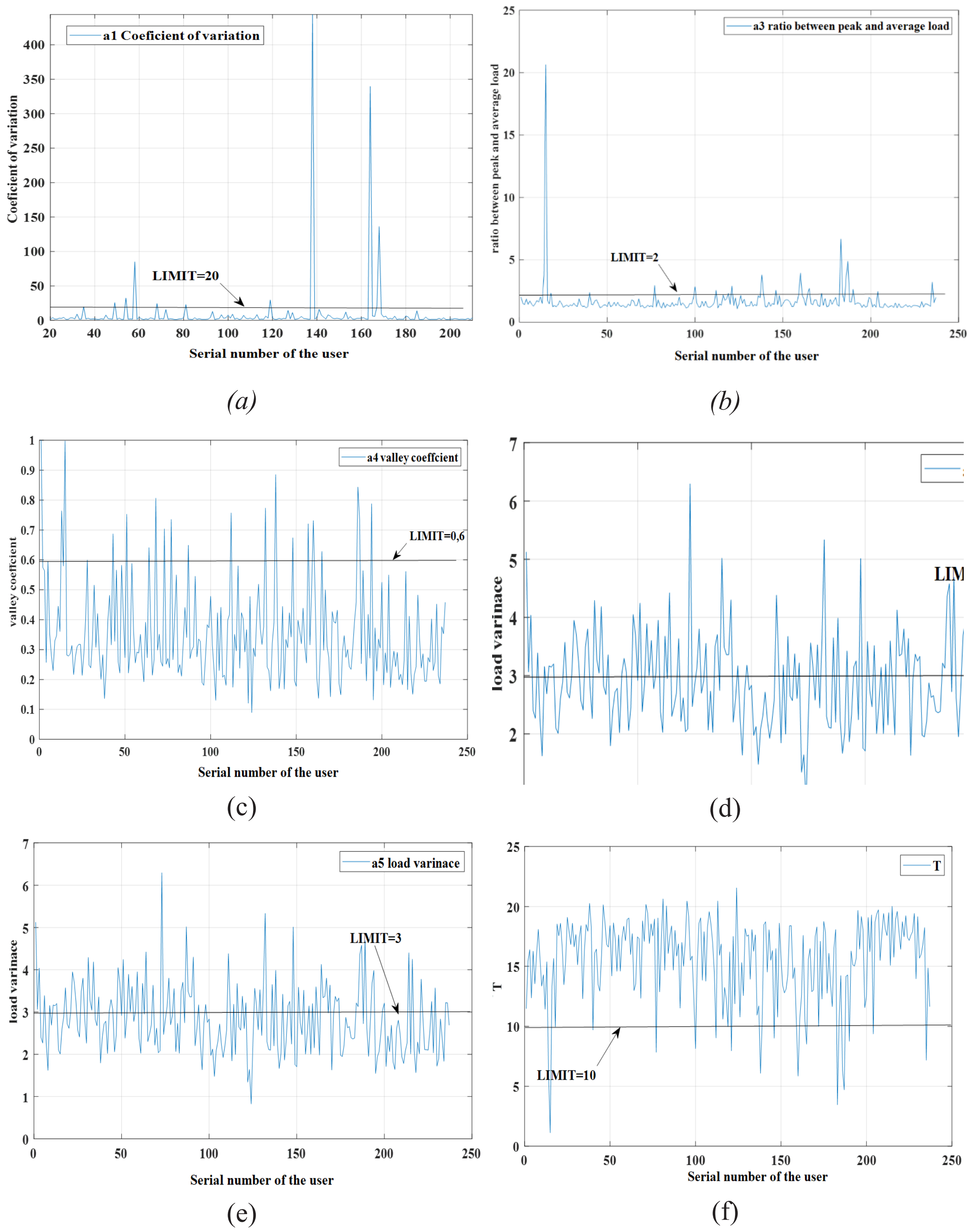


Figure 8. *Statistical indicators with specified limits for 237 consumers: (a) a_1; (b) a_3 (c) a_4; (d) a_2 (e) a_5 (f) T.*

CONCLUSION

This paper presents a method of detecting non-technical losses based on indicators for anomaly detection in load diagrams. Based on the collected measurements, the paper explains the possible methodologies for observing and detecting anomalies at the measurement points to detect non-technical losses, commercial losses, and electricity theft. The paper points to the possibility of simulating the creation of anomalies based on a realistically recorded load diagram. Also, the paper indicates the possibility of comparing all recorded load diagrams from the same part of the smart grid. In both cases, indicators are calculated to detect the existence of an anomaly using their limit values. Future work will be focused on using artificial intelligence and big data from the smart grid to create an algorithm for the same purpose.

ACKNOWLEDGMENT

Funding: The authors would like to thank the Ministry of Science and Technological Development of the Republic of Serbia, which funded research presented in this paper within the framework of Project III 42009 “Smart Grid”.

INSTITUTIONAL REVIEW BOARD STATEMENT:

Not applicable.

INFORMED CONSENT STATEMENT:

Not applicable.

CONFLICTS OF INTEREST:

The authors declare no conflict of interest.

REFERENCES

[1] N. Rajaković, D. Tasić, and G. Savanović, *Distributivne i industrijske mreže*. Beograd, Srbija: Akademska misao, 2004.

[2] G. Grigoras and B. C. Neagu, “Smart Meter Data-Based Three-Stage Algorithm to Calculate Power and Energy Losses in Low Voltage Distribution Networks,” *Energies*, Vol. 12, pp. 1996–1073, 2019.

[3] D. Carr, and M. Thomson, “Non-Technical Electricity Losses,” *Energies*, Vol. 15, No. 6: 2218, 2022.

[4] A. Fragkioudaki, P. Cruz-Romero, A. Gómez-Expósito, J. Biscarri, M. Jesús de Tellechea, and Á. Arcos, "Detection of non-technical losses in smart distribution networks: A review," In International Conference on Practical Applications of Agents and Multi-Agent Systems, 2016, pp. 43–54.

[5] J. L. Viegas, P. R. Esteves, R. Melício, V. M. F. Mendes, and S. M. Vieira, "Solutions for detection of non-technical losses in the electricity grid: A review," *Renewable and Sustainable Energy Reviews*, Vol. 80, pp. 1256–1268, 2017.

[6] N. V. Patil, D. R. Bondar, R. S. Kanase, and P. D. Bamane, "Intelligent Energy Meter with Advanced Billing System and Electricity Theft Detection," In 2017 International Conference on Data Management, Analytics and Innovation (ICDMAI), 2017, pp 36–41.

[7] Dr. S. Thangalakshmi, G. Sangeetha bharath, and S. Muthu, "Power Theft Prevention in Distribution System Using Smart Devices," *International Journal of Applied Engineering Research*, Vol. 10, No. 42, pp. 30841–30845, 2015.

[8] R. Punmiya and S. Choe, "Energy Theft Detection Using Gradient Boosting Theft Detector With Feature Engineering-Based Preprocessing," *IEEE Transactions on Smart Grid*, Vol. 10, No. 2, pp. 2326–2329, 2019, DOI 10.1109/TSG.2019.2892595.

[9] K. Zheng, Q. Chen, Y. Wang, C. Kang, and Q. Xia, "A Novel Combined Data-Driven Approach for Electricity Theft Detection," *IEEE Transactions on Industrial Informatics*, Vol. 15, No. 3, pp. 1809–1819, 2019, DOI 10.1109/TII.2018.2873814.

[10] R. E. Ogu and G. A. Chukwudebe, "Development of a cost-effective electricity theft detection and prevention system based on IoT technology," In 2017 IEEE 3rd International Conference on Electro-Technology for National Development (NIGERCON), 2017, pp. 756–760.

[11] Y. Huang and Q. Xu, "Electricity theft detection based on stacked sparse denoising autoencoder," *International Journal of Electrical Power & Energy Systems*, Vol. 125, 106448, 2021.

[12] X. Kong, X. Zhao, C. Liu, Q. Li, D. Dong, and Y. Li, "Electricity theft detection in low-voltage stations based on similarity measure and DT-KSVM," *International Journal of Electrical Power & Energy Systems*, Vol. 125, 106544, 2021.

[13] Z. Stojković, *Projektovanje pomoću računara u elektroenergetici – primena programskih alata*. Elektrotehnički Beograd, Srbija: Akademska misao, 2009, p. 521.

CFS 2022, Vol. 1, Issue 1, pp. 29–45
https://doi.org/10.5937/1-42709
Original research paper

Comparative Performance Evaluation of Suboptimal Binary Search Trees[1]

Svetlana Štrbac-Savić[1], Milo Tomašević[2], Nemanja Maček[3*], Zlatogor Minchev[4]

[1] Academy of Technical and Art Applied Studies, School of Electrical and Computer Engineering, Belgrade, Serbia; svetlana.strbac@viser.edu.rs

[2] School of Electrical Engineering, University of Belgrade, Belgrade, Serbia; mvt@etf.bg.ac.rs

[3] Academy of Technical and Art Applied Studies, School of Electrical and Computer Engineering, Belgrade, Serbia & University Business Academy in Novi Sad, Serbia & SECIT Security Consulting, Serbia; macek.nemanja@gmail.com

[4] Joint Training Simulation and Analysis Center, Institute of ICT, Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, zlatogor@bas.bg

* Corresponding author: macek.nemanja@gmail.com

Received: 2022-11-07 • Accepted: 2022-12-05 • Published: 2022-12-30

Abstract: Three relevant types of suboptimal binary search trees are comparatively evaluated in this paper: two well-known representatives of height-balanced approaches (the AVL and red-black trees) and a popular self-adjusting splay tree. After a brief theoretical background, an evaluation method was described that employs a suitable synthetic workload method capable of producing diverse desired workload characteristics (different distributions and ranges of key values, varying input sequence lengths, etc.). Evaluation analysis was conducted for search, insert, and delete operations separately for each particular type and in appropriate combinations. Experimental results for an average operation cost as well as for tree maintenance cost are comparatively presented and carefully discussed. Finally, the suggested favorable conditions for application of each tree type are summarized.

Keywords: binary search trees; AVL trees; red-black trees; splay-trees; self-adjusting trees.

1. INTRODUCTION

Binary search tree (BST) is a basic data structure that combines two benefits. It allows for fast binary searching of a sorted structure and also, like each dynamic structure, ensures efficient maintenance during the insertion and deletion of keys. It provides the $O(\log n)$ complexity of search, insert, and delete operations in the best and average cases, but for degenerated topologies in the worst case the performance of operations can be deteriorated to $O(n)$. In order to prevent such cases, the tree topology should be kept balanced. However, keeping the optimal balance after each insert or delete operation can impose a

1 This paper in an extension of [1].

significant maintenance overhead (up to $O(n)$ sometimes). A compromise is often found in suboptimal BSTs by somewhat relaxing optimal balancing criteria. This approach can significantly reduce the maintenance cost while still guaranteeing the $O(\log n)$ complexity of search even in the worst case with some small constant degradation factor compared to an optimally balanced tree. Different suboptimal strategies have been proposed, but height balancing is the most popular one. The main representatives are AVL trees and red-black trees, which are based on some local sub-tree balancing criteria rather than global ones. In both types of BST (but in different ways), the difference in heights of the leaves is practically restricted within a small constant factor, preventing the linear worst case.

The aforementioned balancing techniques are efficient if the keys in the tree are searched for with nearly uniform probabilities. However, the search probabilities for different keys are often non-uniform, especially when the level of temporal locality is increased. According to that, self-adjusting binary search trees have been proposed that are reorganized even after a search operation. A prominent representative of such an approach is the splay tree, where the successfully found key is moved up to the root in order to exploit the benefits of temporal locality. Since splay trees do not have some explicit balance criteria, the worst case can even go up to $O(n)$, which is acceptable only if it occurs vary rarely. However, the amortized analysis, which gives the time complexity of operations in a series, guarantees $O(\log n)$ complexity in the average case.

The main goal of this paper is to conduct a comparative performance evaluation of AVL, red-black, and splay trees as prominent representatives of suboptimal binary search trees. In order to analyze the performance of these trees under a wide spectrum of different conditions, an appropriate synthetic workload generator is used, which is capable of producing diverse desired workload characteristics. The performance indicators were chosen to be platform- and implementation-independent. The evaluation results should indicate the optimal suggested condition for the employment of these types of trees.

2. MATERIALS AND METHODS

This section provides a brief theoretical background on the AVL, RB, and splay trees, respectively, with their definitions and considerations on the time complexity of the operations.

2.1. AVL Trees

The AVL trees proposed by Adelson-Velski and Landis are height-balanced trees [2]. Let us define the balance of a node as the difference between the heights of its left and right sub-trees. Then, the AVL tree is defined as a binary search tree in which the absolute value of the balance for each node is one at most. The height of an empty tree is defined as 0.

In this way, the balance criterion is considerably relaxed. While the leaves in an optimally balanced tree can be deployed only in two lowest levels, in an extreme case the leaves of the AVL tree can span the range between levels h and $2h$. The worst topology of the AVL tree with maximum height for a given number of nodes is referred to as the Fibonacci tree.

In the AVL trees, the node balance can only be 1, 0, and -1. A node with a balance of 1 leans left, while a node with a balance of -1 leans right. Some insert or delete operations can disturb the balances of the node ancestors on their path to the root, but, as long as they are in the allowed range, there is no need to reorganize the tree. However, when at least one ancestor balance becomes 2 or -2, the specific tree adjusting operations (called single or double rotations) are carried out in order to return the balance of all nodes to the allowed range. The rotations are relatively inexpensive and infrequent, so the overhead of maintaining the AVL tree is quite acceptable [3].

In spite of the fact that the AVL tree is only "nearly" balanced, it was demonstrated in [4] that for an AVL tree with n nodes, its height h satisfies the condition

$$h < 1.4405 \log_2 (n+2) - 0.327 \tag{1}$$

Since the number of comparisons on the search path is determined by the tree height, its finding guarantees that the time complexity of the search is $O(\log n)$, where n denotes the number of nodes in the tree. Along with the same time complexity as in an optimally balanced tree, the degradation factor of the worst case search path is also quite acceptable (less than 45%). Since the eventual rotations in insert and delete operations impose some practically constant additional overhead, the length of the search path is also dominant factor which determines their $O(\log n)$ complexity.

2.2. Red-black Trees

Another nearly balanced topology principally based on height balancing is the red-black tree. While the AVL tree directly restricts the local dis-balance for each node, the red-black tree indirectly controls the length of the search paths by defining the color of each node. It uses an extra bit that denotes a node as red or black and imposes some coloring rules as follows.

The binary search tree is a red-black tree if it satisfies the following conditions [5]:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both his sons are black.
5. Every path from a given node to any of its descendant leaves contains the same number of black nodes.

In the rest of the paper, these trees will be referred to as RB trees.

The RB trees also represent an implementation of the 2-3-4 trees in a form of binary tree [6]. The 2-3-4 trees have optimally balanced topology since all leaves are at the same level. Besides the usual 2-nodes as in the binary trees, these trees may also have 3-nodes with two keys and three sub-trees, as well as 4-nodes with three keys and four sub-trees. The implementation of the 2-3-4 trees requires more memory, and insert and delete operations are more complex since, when necessary, one type of node is transformed into another. Therefore, it is very important that the 2-3-4 trees are B-trees of degree 4, being isomorphic with the RB tree, which means that for each 2-3-4 tree there exists at least one RB tree as its binary representation [7].

If the tree structure is modified by some insert or delete operation that impairs the requirements of the definition, rotations are performed in order to re-establish the correct structure and coloring. The basic operations of the RB trees have been described in [4, 7] and [8]. It is demonstrated in [5] that the height of the RB tree is bounded by

$$h < 2\log\left(n+1\right). \tag{2}$$

Therefore, just like in the AVL trees, the logarithmic performance of all operations if the RB trees is also guaranteed in the average and worst case.

2.3. Splay Trees

Splay trees were proposed by Sleator and Trajan [9]. Although they do not rely on some explicit balancing strategy, unlike AVL and RB trees, the splay trees are reorganized on each access, including even non-invasive search operations, by means of rotations. Each accessed or inserted key is moved to the root, as well as the predecessor/successor of a deleted or unsuccessfully searched key. The rationale behind this is to exploit temporal locality with increased probabilities of accessing recently used keys or range of values.

Two techniques can be employed for tree reorganization: top-down splaying and bottom-up splaying. In bottom-up splaying, the tree is searched for the key in the first step, saving some parent information in nodes on the search path, and then the node is lifted up to the root by consecutive zig, zig-zag, and zag-zag rotations, as described in [10]. In [9], the authors favor top-down playing since it performs in one step without need for an extra storage. Because of that, top-down splaying was used in this evaluation study.

Although splay trees do not guarantee $O(\log n)$ complexity in the worst case, it is demonstrated in [11] that the complexity of performing a series of m operations in splay tree with n keys is

$$O\left(m\left(1+\log n\right)+n\log n\right). \tag{3}$$

Consequently, the amortized cost of operations in splay trees is also logarithmic.

3. RELATED WORK

The comparison of different kinds of binary search trees was a goal of many studies. The study from [12] follows a similar approach to our study. Six types of binary search trees were compared: random BST, AVL tree, and four types of self-adjusting binary search trees (splay trees with top-down splaying and bottom-up splaying, and self-adjusting trees with MTR and Exchange techniques described in [13]). It was concluded that AVL trees are the most efficient ones when searching is the most frequent operation, while, among the self-adjusting structures, the splay trees with top-down splaying technique perform the best in a highly dynamic environment.

In [14], four types of binary search trees were compared: unbalanced BST, AVL tree, RB tree, and splay tree. For each of them, five different node representations were considered: plain, with parent pointers, threaded, right-threaded, and with an in-order linked list. In total, 20 BST variants were compared using three experiments in real-world scenarios with real and artificial workloads. The measured parameters were execution time and the number of comparisons. The results indicate that RB trees are preferred when random input with occasional runs in sorted order is expected. When insertions in sorted order are prevalent, the AVL trees outperform the others for later random access, whereas splay trees perform the best for later sequential or clustered access.

In [15], performance of height-balanced trees (HB[k]) is evaluated. Both analytical and experimental results that show the cost of maintaining HB[k] trees as a function of k are discussed. The AVL tree is treated as a special case for $k = 1$. For the AVL trees, it was concluded that only the search time is a function of the tree size, and in a general case, the maintenance does not depend on the tree size. In general, for HB trees for $k > 1$, the execution times of the procedures for maintaining the HB[k] trees are independent of the tree size, except for the average number of nodes revisited on a delete operation in order to restore the HB[k] property on its trace back. Also, the cost of maintaining HB[k] trees drops significantly as the allowed imbalance (k) increases.

Bear and Schwab in [16] empirically compare the height-balanced trees with the weight-balanced trees by means of simulation with a synthetic workload. In the conclusion, they give preference to the AVL trees.

In [17], a novel limit-splaying heuristic called periodic-rotation is described. It performs splaying after n insertions or accesses in order to reduce the maintenance cost while preserving the performance. They experimentally compared seven data structures: the simple BST, the RB trees, splay trees both with top-down and bottom-up splaying techniques, randomized trees, and their heuristic splay tree. It was presented that such heuristic splay tree where splaying is done periodically rather than on each access is around 27% faster on average than efficient bottom-up splaying. Over five separate text collections that were chosen for workload, several somewhat unexpected conclusions were highlighted: first, top-down splaying is slower than bottom-up splaying in practice; second, bottom-up splaying is about as fast as a self-adjusting randomized tree, but in general is around 25% slower than a BST; and, finally, the most efficient heuristic splaying scheme is only 3% faster than a BST, which performed even better than RB trees.

The study in [18] provides a comparative analysis of a number of different binary search trees: un-optimized BST, AVL tree, several types of the weight-balanced trees (described in [19]), the trees where the searched node moves by one level towards the root [13], as well as the tree with appropriate combinations of some algorithms. The evaluation is based on measured execution times for different types of input sequences. The operations considered were insertion and searching. Although the basic search operation in an ordinary binary tree is quite efficient in many cases, it was concluded that the tree that combines the principles of the AVL tree and an ordinary BST is the most efficient generally.

Although the studies from [12] and [14] are similar to the topic of our study in terms of analyzed trees, the comparisons are carried out from different perspectives. While some previous studies (e.g., [14]) observed a specific environment in which the trees were applied, our study presents a more general performance study independent of the implementation of the algorithm, operating system, the specific applications, and the machine on which the tests are conducted.

4. EVALUATION METHOD

Although the real workloads are preferred in many studies focused on a specific area of applications, they are unable to reflect a wide spectrum of different workload characteristics. Since an appropriate synthetic workload generator is very convenient for producing diverse desired workload characteristics, our comparative performance evaluation employs the simulation method with specific synthetic workload described in [20]. The main parameters of the workload are: the number of keys, the range of the key values, the distribution of the key values, time locality, the relative frequency of search, insert, and delete operations, the probability of successful and unsuccessful search, etc. The performance indicators have been chosen to be independent of the algorithm implementation and platform on which the measurements are performed (tree height, number of rotations, etc.).

The various key sequences were generated in order to obtain a more complete insight into the chosen trees performance. The intervals from which the key values are taken, the number of elements in the sequence, and the frequency of the key values were varied. A special care was taken to simulate the time locality of the keys in some cases.

The lengths of the key sequences are chosen to be between 10 and 1,000,000 elements in multiples of 10. The number of elements in the sequences was varied in order to establish how the performance depends on the tree size.

The values in the same key sequence may be repeated. They are taken from intervals whose lower bounds are set to zero and whose upper bounds vary from case to case. Four groups of the key sequences used in this evaluation differ according to the way of key generation. The sequences without key repetitions are used for building an initial tree.

The first group of key sequences is sorted in increasing order. They contain unique key values without repeating.

The second group of key sequences is similar to the first, but the order of the key values is random. All values appear exactly once in the sequence, and the length of the key sequence corresponds to the interval upper bound.

In the third group, the keys are also generated randomly. However, the elements are chosen out of a certain interval, whose upper bound also varies, as well as the sequence length. The consequence of such a key choice is that some values from the interval can be repeated, while other values do not appear in the sequences.

The fourth group has the key values that can also be repeated in a sequence, while the sequence lengths and the interval upper bounds are varied like in the third group. However, instead of using random, uniformly distributed key values, the goal was to obtain the key sequences with a non-uniform distribution and to enforce the temporal locality of chosen values, which is sometimes quite pronounced in tree accesses. The function

$$y = \frac{1-x}{1+ax} \tag{4}$$

was used to enforce different levels of temporal locality in the following way. First, an initial sequence with *n* keys without repeated values is formed (let it denote by array *key*). Then, *x* is randomly chosen from the interval $x \in [0,1]$. With such an *x*, *y* is calculated according to equation (3). Since $y \in [0,1]$ for *a* > 0, index *i* is then calculated as $i = n \cdot y$. Finally, element of the *key* sequence with index *i*(*key*[*i*]) is entered into the resulted key sequence. This procedure is repeated until the resulting sequence of the required length is generated.

By varying the parameter *a*, the shape of the curve can be adjusted, as shown in Figure 1. Values 1, 10, 50, and 100 were taken for parameter *a*, and the results for *a*=100 were analyzed. For higher values of *a*, a uniform distribution of *x*, values of *y* are lower. Consequently, lower indices of the key array are much more probable, which increases the time locality in the resulting sequence of key values closer to the start of the array.

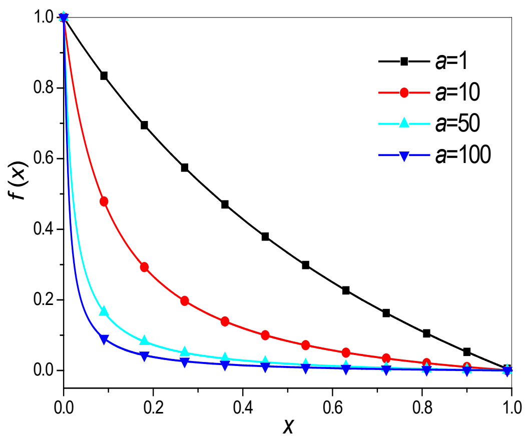


Figure 1. *Function (4) in the interval x ∈ [0, 1] for the different values of parameter a.*

Search, insertion, and deletion operations are evaluated both independently in separate series and jointly in mixed series. In a mixed series, percentages of particular operations are varied, and operations appear randomly according to the adopted frequency.

The following performance indicators were collected during experiments:

- Average height during search operation – average height where the element was found during a successful search or the height of the node where the search was finished in the case of unsuccessful search. It indicates the number of comparisons on the search path.
- Tree height – this parameter refers to the maximum height of the initial tree on which the search series was performed.
- Average number of rotations per operation. It indicates the maintenance cost.
- Tree height at the end of a series of insert operations.
- Average height of the splay tree in an entire series of operations.

5. RESULTS

The experimental results collected from running the series of insert, search, and delete operations are presented and discussed in the first three subsections, respectively. In the last subsection, the results from a series of appropriate combinations of all three operations are shown and analyzed.

5.1. Insert Operation

Since only different values can be inserted in a binary search tree, only the sequences with unique keys are applicable for evaluation of insert operation. The insertions were started with initially empty trees.

Table 1. *Series of insert operations for input key sequences with sorted values in increasing order.*

Number of inserted keys (sorted)	Average height per operation			Average number of rotations per insert operation		Height of resulted tree	
	AVL	RB	Splay	AVL	RB	AVL	RB
100	5.730	8.090	0.990000	0.93000	0.89000	6	10
1,000	8.977	14.481	0.999000	0.99000	0.98300	9	16
10,000	12.362	21.138	0.999900	0.99860	0.99760	13	23
100,000	15.689	27.723	0.999990	0.99983	0.99969	16	30
1,000,000	18.951	34.379	0.999999	0.99998	0.99996	19	36

The results for sorted input key sequences of variable lengths are shown in Table 1. Since the current inserted key is always the highest one, splay trees need no rotations, but after the series of insert operations, the resulting tree has degenerated topology and is far from optimal for most operations that can follow in practice. The AVL trees have a considerably lower average height of the current inserted node, but the RB trees have slightly less average rotations per operation. As a comparison and a rotation are the operations with similar cost, so the AVL tree is more efficient at inserting a sorting sequence. In addition, its final tree height in case of the RB tree is almost twice as tall. Their efficiency is greater considering the final height after the series of insert operations. Since this tree can be the initial tree for some other operations, it can have a serious impact on the cost of the operations that follow.

The results for inserting key sequences generated randomly are given in Table 2. The heights of resulted tree are shown for the AVL and RB tress only since it is very relevant for subsequent search operations, while for splay trees it changes with each operation. Splay trees perform the worst by far in this case, as expected. As for both height indicators, the AVL and RB trees show similar results, while the RB trees have fewer rotations per operation.

Table 2. *Series of insert operations for input key sequences with random values.*

Number of inserted keys (random)	Average height per operation			Average number of rotations per insert operation			Height of resulted tree	
	AVL	RB	Splay	AVL	RB	Splay	AVL	RB
100	5.390	5.410	7.390	0.680	0.550	2.450	7	7
1,000	8.699	8.777	13.929	0.646	0.555	4.954	11	11
10,000	12.056	12.074	20.442	0.645	0.536	7.219	15	15
100,000	15.450	15.523	27.112	0.644	0.534	9.574	19	20
1,000,000	18.815	18.829	33.769	0.640	0.530	11.927	23	23

The results from both Tables 1 and 2 for the same type of tree indicate that the average number of rotations per operation is practically constant over all varied tree sizes for both random and sorted key sequences, except for inserting keys from random sequence in case of splay trees, where this indicator steadily increases with tree size. In the case of most unfavorable sorted input, both AVL and RB trees experience practically a rotation on every insert, while for random input the more efficient RB tree requires a rotation in almost every other insertion. The situation for splay trees is quite opposite, since the maintenance cost for random input is much higher than for a sorted one. Although splay trees most efficiently handle insertion of sorted key sequence, the final tree height has degenerated topology equivalent to a linked list inappropriate for later searching. The fact that the AVL trees have a more restrictive balance criterion contributes to more efficient handling in inserting keys of a sorted sequence, reflected in a considerably smaller average height per operation and final tree height than in the case of the RB trees.

5.2. Delete Operation

A series of delete operations are conducted on initial trees generated with a series of insert operations of random key values in order to be large enough. The heights of the initial trees were 23 for both the AVL and RB trees and 69 for the splay tree. As in the case of insert operations, key sequences with no repeated values were chosen in order to avoid unsuccessful delete operations.

The results presented in Table 3 confirm that splay trees perform the best in cases of deletions of keys from sorted sequences. Each delete operation raises the right subtree in which the next key in sequence is found, making its subsequent deletion more efficient. In the case of splay trees, as the number of operations in a series increases, both the average height of the deleted node and the average number of rotations decrease. Except for a very small percentage of nodes deleted from the initial tree, the RB trees perform better than AVL trees but are still much worse than splay trees.

For the deletion of keys in random order (Table 4), the AVL and RB trees perform very similarly, and their performance indicators only slightly change with the varying number of deleted keys. On the other side, splay trees are again noticeably less efficient, and their performance deteriorates with an increasing number of deleted keys in random order.

Table 3. *Series of delete operations on key sequences with sorted values in increasing order.*

Number of deleted keys (sorted)	Average height of deleted node per operation			Average number of rotations per delete operation		
	AVL	RB	Splay	AVL	RB	Splay
100	15.520	15.980	2.210	0.600	0.700	0.890
1,000	14.710	15.793	1.500	0.572	0.680	0.546
10,000	14.730	14.717	1.392	0.578	0.662	0.497
100,000	14.150	14.083	1.361	0.578	0.659	0.482
1,000,000	13.708	12.534	1.347	0.575	0.658	0.475

Comparing the results for the sorted and random order of deleted keys in splay trees, two opposite trends can be noticed. Longer sorted key sequences during deletions are favorable, while longer random ones are unfavorable for both the average height of the deleted node and the average number of rotations. The AVL and RB trees have a larger average height and a smaller average number of rotations for the same number of delete operations in the case of a random key sequence. Also, unlike splay trees, these performance indicators for AVL and RB trees are relatively insensitive to the number of deleted keys.

Table 4. *Series of delete operations on key sequences in random order.*

Number of deleted keys (random)	Average height of deleted node per operation			Average number of rotations per delete operation		
	AVL	RB	Splay	AVL	RB	Splay
100	15.060	16.000	8.550	0.330	0.320	3.160
1,000	15.835	16.321	14.648	0.384	0.357	5.458
10,000	15.402	16.541	21.745	0.351	0.359	8.166
100,000	16.703	16.865	28.671	0.357	0.361	10.747
1,000,000	16.465	16.437	35.687	0.378	0.387	13.359

5.3. Search Operation

The search operation is especially important because it is usually the most frequent operation and also because it is the first part of insert and delete operations. This is the reason why the performance of this operation is analyzed in more detail.

Search operations in the AVL and RB trees do not modify the tree topology, and no rotations are required. Therefore, the average length of the search path is the only relevant performance indicator. However, in splay trees, every search operation is followed by an adjustment of the topology, and the average number of rotations is meaningful as well. All the results presented in Table 5 are obtained by searching for a sorted sequence of keys in increasing order. The heights of the initial trees were 15 for the AVL and RB trees and 43 for the splay tree. In the first three cases (up to 10,000 keys searched), all searches were successful, while in the other two cases, there were 90% and 99% unsuccessful searches.

Table 5. *Series of search operations on key sequences with sorted values in increasing order.*

Number of searched keys (sorted)	Average search path length per operation			Average number of rotations (splay tree)
	AVL	RB	Splay	
100	12.260	11.160	2.790	0.670
1,000	11.697	11.472	2.460	0.515
10,000	11.578	11.564	2.348	0.473
100,000	14.656	12.858	1.135	0.047
1,000,000	14.966	12.986	1.013	0.048

Again, the splay trees are obviously the most efficient ones when the key sequence is sorted. With longer search sequences, their performance is steadily improving. After the first operation in a series radically rearranges the tree topology, each subsequent search slightly adjusts it to make the subsequent operation more efficient. Finally, when unsuccessful search operations for key values higher than the maximum key in the tree prevail, they execute very fast since no further tree adjustments are needed. However, the average cost of search operations in the AVL and RB trees depends greatly on the initial tree size since there are no adjustments during the series of search operations. Unlike splay trees, in case of AVL and RB trees performance is deteriorated when the number of keys in the sorting sequence grows due to prevailing number of unsuccessful search operations (their search paths are ended in leaves of the tree). The performance of the AVL tree is especially affected in this case.

Table 6. *Series of 100,000 search operations on key sequences with random distribution.*

Range of searched key values (random)	Average search path length per operation			Average number of rotations (splay tree)
	AVL	RB	Splay	
0..99	13.741	14.682	6.254	2.059
0..999	13.299	14.256	10.868	3.574
0..9,999	13.756	14.813	15.662	5.111
0..999,999	16.743	15.853	3.421	0.884

Table 6 presents the results for sequences of 100,000 search operations with random distribution of key values performed on initial trees which were built with series of insert operations of random keys values between 0 and 99,999. Table 7 shows the results obtained under the same conditions but with a non-uniform distribution and enforced temporal locality of the searched key values. The heights of the initial trees were 19 for the AVL tree, 20 for the RB tree, and 56 for the splay tree. In cases when range of key values searched was 0..999,999, there were approximately 90% unsuccessful search operations, while in other sequences all searches were successful.

Splay trees clearly outperform the others when non-uniform sequences of key values are searched since they can take advantage of increased temporal locality (much better indicators for splay trees in Table 7 compared to those in Table 6) by proper adjustment of the topology, while the AVL and RB trees are insensitive to this phenomenon. Their performance rather depends on a set of key values and their place in the initial tree. For both distributions, the average cost of a successful search operation in the AVL trees is slightly better than in the RB tree. The large percentage of unsuccessful operations affected the performance of the analyzed trees in the same manner as in the previous case.

Table 7. *Series of 100,000 search operations on key sequences with non-uniform distribution.*

Range of searched key values (non-uniform)	Average search path length per operation			Average number of rotations (splay tree)
	AVL	RB	Splay	
0..99	14.043	13.283	1.916	0.593
0..999	13.307	14.183	5.829	1.919
0..9,999	13.290	14.374	10.296	3.406
0..999,999	16.712	15.820	2.763	0.666

Figure 2 depicts how the distribution of key values in the search sequences affects the average length of the search path. It more explicitly demonstrates much better handling of increased temporal locality of search sequences in splay trees than in the AVL and RB trees.

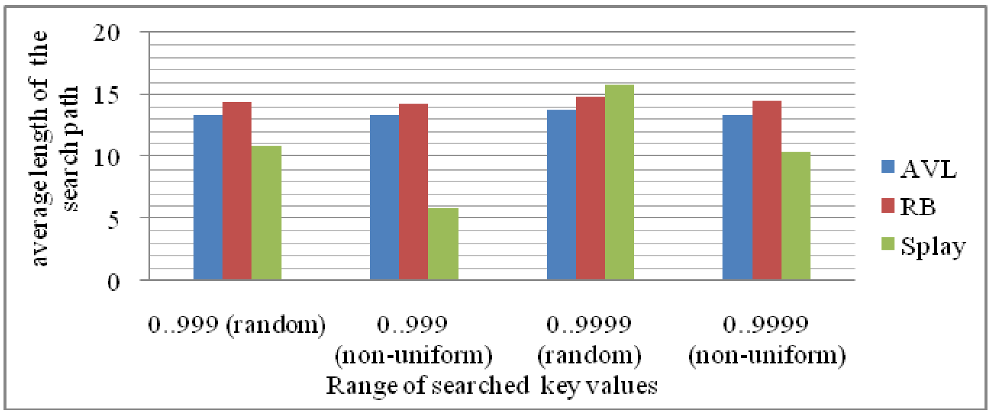


Figure 2. *Average lengths of search path for series of search operations for random and non-uniform key sequences.*

In previous experiments, unsuccessful search operations for key values in the interval 100,000..999,999 traversed the right-most search part, which is not quite typical. Therefore, a more realistic situation with unsuccessful searches dispersed across an entire tree should be simulated. To this end, about 20,000 randomly chosen key values were deleted from an initial tree randomly built with 100,000 keys. The obtained tree was used for the evaluation of search sequences of 100,000 keys. Four ranges of key values (0..29,999, 0..49,999, 0..79,999, 0..99,999) and two distributions (random and non-uniform) are varied to produce six new sequences.

The results from such input sequences are given in Table 8 for random distribution and in Table 9 for non-uniform distribution. The efficiency of search operations in the AVL and RB trees practically does not depend on the range of search keys or type of distribution. A number of unsuccessful search operations increases their average search path to some extent. The AVL tree is again slightly better than the RB tree, but both types of trees outperform the splay tree, especially for randomly distributed searched key values. Although enforced temporal locality in non-uniform distribution evidently improves the performance of search operations and the maintenance cost of the splay tree, it is not sufficient to make it better than the AVL and RB trees in conditions when unsuccessful search operations are spread over a larger range of key values.

Table 8. *Series of 100,000 search operations in different ranges of searched key values (random distribution).*

Range of searched key values (random)	Average search path length per operation			Average number of rotations (splay tree)
	AVL	RB	Splay	
0..99,999	14.991	15.062	22.208	7.188
0..79,999	14.903	15.224	20.503	6.672
0..49,999	14.810	15.573	19.569	6.370
0..29,999	14.557	15.566	18.249	5.940

Table 9. *Series of 100,000 search operations in different ranges of searched key values (non-uniform distribution).*

Range of searched key values (non-uniform)	Average search path length per operation			Average number of rotations (splay tree)
	AVL	RB	Splay	
0..99,999	14.448	14.426	16.829	5.502
0..79,999	15.152	15.479	16.172	5.290
0..49,999	15.365	16.075	16.536	5.406
0..29,999	14.675	15.656	15.797	5.027

5. DISCUSSION

Finally, after insert, delete, and search operations are analyzed separately, a more realistic situation when different operations are interspersed is in place. Different workload characteristics are also simulated by varying the relative frequencies of these three types in a sequence. Two different series of operations are analyzed. It was assumed that the search operation is the most frequent one, while delete and insert operations are equally represented in this evaluation. The initial tree was built from the values in the interval [0, 99999] inserted in random order. The cost of building the initial tree is not accounted for in the evaluation of the mixed series, which has 100,000 operations.

Table 10. *Series of 80% search, 10% insert, and 10% delete operations with random distribution of key values.*

Range of key values (random)	Average height per operation			Average number of rotations per operation		
	AVL	RB	Splay	AVL	RB	Splay
0..9	14,189	15,897	1,290	0,021	0,015	0,143
0..99	13,325	15,813	2,157	0,023	0,018	0,463
0..999	11,623	13,209	7,120	0,023	0,020	2,319
0..9,999	14,356	15,359	16,556	0,029	0,028	5,499
0..99,999	14,711	14,782	21,957	0,033	0,030	7,309
0..999,999	17,925	18,017	19,020	0,066	0,055	6,275

The results for the first mixed series made of 80% search, 10% insert, and 10% delete operations with the key values from different ranges in random order are presented in

Table 10. Splay trees are mostly sensitive to the range of key values. For smaller ranges, they have a smaller average height per operation than both AVL and RB trees at the expense of rotations constantly used to adjust the tree. However, as the growing range of values decreases, the temporal locality and its average performance become significantly worse. When compared to the RB trees, the AVL trees, as a more restrictive topology, require slightly more rotations, but it pays off in a smaller average height per operation due to the prevalent number of search operations. For smaller ranges of key values, both the AVL and RB trees are less sensitive to this parameter. However, a very large range of key values impairs their performance because of the increased incidence of unsuccessful search operations ending in the leaves.

Table 11 shows the experimental results for the same frequencies of operations as before, but the key values in mixed sequences follow the non-uniform distribution. It is evident again that splay trees perform noticeably better for key values with a non-uniform distribution of exploiting enforced temporal locality. On the other side, there is no consistent effect on the performance of the height-balanced representatives for this distribution. The AVL trees are still slightly better.

Table 11. *Series of 80% search, 10% insert, and 10% delete operations with non-uniform distribution of key values.*

Range of key values (non-uniform)	Average height per operation			Average number of rotations per operation		
	AVL	RB	Splay	AVL	RB	Splay
0..9	15,057	15,127	0,987	0,023	0,019	0,031
0..99	15,025	15,176	2,809	0,018	0,010	0,630
0..999	14,841	15,899	8,336	0,009	0,006	2,775
0..9,999	15,395	16,447	13,440	0,014	0,013	4,482
0..99,999	15,265	15,292	18,195	0,023	0,021	6,071
0..999,999	16,961	17,168	17,345	0,060	0,050	5,748

Table 12. *Series of 50% search, 25% insert, and 25% delete operations with random distribution of key values.*

Range of key values (random)	Average height per operation			Average number of rotations per operation		
	AVL	RB	Splay	AVL	RB	Splay
0..9	14,459	15,729	1,724	0,052	0,040	0,357
0..99	14,231	15,594	4,058	0,056	0,046	0,120
0..999	12,716	13,618	7,402	0,058	0,048	2,395
0..9,999	14,174	15,168	16,869	0,066	0,059	5,694
0..99,999	14,795	14,864	23,330	0,083	0,073	7,970
0..999,999	17,101	17,660	21,158	0,165	0,136	7,071

Table 13. *Series of 50% search, 25% insert, and 25% delete operations with non-uniform distribution of key values.*

Range of key values (non-uniform)	Average height per operation			Average number of rotations per operation		
	AVL	RB	Splay	AVL	RB	Splay
0..9	15,139	15,289	0,948	0,058	0,047	0,065
0..99	14,682	15,046	2,924	0,042	0,035	0,779
0..999	14,686	15,794	8,193	0,018	0,014	2,773
0..9,999	15,323	16,432	13,526	0,031	0,028	4,584
0..99,999	15,382	15,409	18,881	0,042	0,037	6,423
0..999,999	16,416	16,836	18,810	0,130	0,109	6,353

After that, the relative frequencies for the second mixed series were set to 50% for search, 25% for insert, and 25% for delete operations to see how a higher percentage of input and delete operations affected the average cost per operation. The results for sequences with key values from different ranges are presented in Table 12 for random distributions and in Table 13 for non-uniform distribution. In case of the AVL and RB trees, the average number of rotations is growing almost linearly with the increased percentage of insert and delete operations. The cost of tree maintenance in splay trees is not much affected since they adjust the tree topology on each access. It seems that different relative frequencies of three operations different do not have significant impact on an average height per operation in all trees for both random and non-uniform distributions. Suitability of non-uniform distribution and smaller ranges of key values for splay trees is evidenced once again.

6. CONCLUSIONS

It can be concluded that the AVL and red-black trees perform quite similarly. However, in a number of cases, the AVL trees are slightly more efficient than their red-black counterparts in terms of average height per operation, especially in sequences of search operations and in sequences of mixed operations, as a consequence of their more stringent topology requirements. It comes at the expense of somewhat increased maintenance costs expressed in an average number of rotations. The red-black trees are more efficient when the key values in sequences are unique and random. Both types of trees are rather insensitive to the order of key values and temporal locality. On the other hand, the splay trees prefer situations where key values come in sorted order. They especially outperform the others when the temporal locality of accesses is increased and when searching for a rather narrow range of the key values. In these cases, the high cost of constant tree maintenance is amortized; otherwise, it can be intolerable. This conclusion indicates that a modified splay tree could be proposed that dynamically tracks the temporal locality of key values (e.g., access counters) and adjusts the tree topology only when it is justified.

For more information on AVL trees, readers may consult [15]. Details on the AVL-based settlement algorithm and reservation system for smart parking systems in IoT-based smart cities are presented in [16]. More details on red-black trees can be found in [17]. To read more on splay trees, readers may cf. [18]. To learn more about augmented binary search trees, cf. [19].

FUNDING:

This research received no external funding.

CONFLICTS OF INTEREST:

The authors declare no conflict of interest.

REFERENCES

[1] S. Štrbac-Savić and M. Tomašević, "Comparative performance evaluation of the AVL and red-black trees," In Proceedings of the Fifth Balkan Conference in Informatics, September (BCI '12), 2012, pp. 14–19.

[2] G. M. Adelson-Velskii and E. M. Landis, "An Algorithm for the Organization of Information," *Soviet Mathematics Doklady*, vol. 3, pp. 1259–1263, 1962.

[3] M. Tomašević, *Algorithms and Data Structures*. Belgrade, Serbia: Academic Mind, (in Serbian), 2011.

[4] D.E. Knuth, The Art of Computer Programming, Volume 3: Sorting and Searching, Reading, Massachusetts: Addison-Wesley, 1998.

[5] T. Cormen, Ch. Leiserson, and R. Rivest, *Introduction to Algorithms*. The MIT Press, McGraw-Hill, 2009.

[6] A. Aho, J. Hopcroft, and J. Ullman, *Data Structures and Algorithms*. Addison-Wesley, 1983.

[7] B. Flaming, *Practical Data Structures in C++*. New York: John Wiley and Sons, 1993.

[8] C. Okasaki, "Red-black trees in a functional setting," *Journal of Functional Programming*, Vol. 9, No. 4, pp. 471–477, 1999.

[9] D. Sleator and R. E. Trajan, "Self-Adjusting Binary Search Trees," *Journal of the Association for Computing Machinery*, Vol. 32, No. 3, pp. 652–686, 1985.

[10] M. A. Weiss, *Data Structures and Algorithm Analysis in C*. Reading, MA: Addison-Wesley, 1997.

[11] R. Cole, "On the Dynamic Finger Conjecture for Splay Trees, Part II: The Proof," *SIAM Journal on Computing*, Vol. 30, pp. 44–85, 2000.

[12] J. Bell and G. Gupta, "An Evaluation of Self-Adjusting Binary Search Tree Techniques," *Software – Practice and Experience*, Vol. 23, pp. 369–382, 1993.

[13] B. Allen and I. Munro, "Self-Organising Binary Search Trees," *JACM*, Vol. 25, pp. 526–535, 1978.

[14] B. Pfaff, "Performance Analysis of BSTs in System Software," *ACM SIGMETRICS,* Vol. 32, Issue 1, pp. 410–411, 2004.

[15] R. Wiener, "AVL Trees," In *Generic Data Structures and Algorithms in Go*, Berkeley, CA: Apress, 2022, pp. 315–347.

[16] H. Canli and S. Toklu, "AVL Based Settlement Algorithm and Reservation System for Smart Parking Systems in IoT-based Smart Cities," *The International Arab Journal of Information Technology*, Vol. 19, No. 5, pp. 793–801, 2022.

[17] R. Wiener, "Red-Black Trees," In *Generic Data Structures and Algorithms in Go*, Berkeley, CA: Apress, 2022, pp. 363–385.

[18] B. A. Berendsohn and L. Kozma, "Splay trees on trees," In Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2022, pp. 1875–1900.

[19] T. Luo, "Learning Augmented Binary Search Trees," Doctoral thesis, Carnegie Mellon University, Pittsburgh, PA, 2022.

CFS 2022, Vol. 1, Issue 1, pp. 47–54
https://doi.org/10.5937/1-42710
Original research paper

Towards Hybrid Supercomputing Architectures

Nenad Korolija[1*], Kent Milfeld[2]

[1] School of Electrical Engineering, University of Belgrade, 73 Bulevar Kralja Aleksandra, 11020 Belgrade, Serbia; nenadko@etf.bg.ac.rs

[2] The University of Texas, Texas Advanced Computing Center, J.J. Pickle Research Campus, 10100 Burnet Rd, Austin TX 78758; milfeld@tacc.utexas.edu

* Corresponding author: nenadko@etf.bg.ac.rs

Received: 2022-09-05 • Accepted: 2022-09-24 • Published: 2022-12-30

Abstract: In light of recent work on combining control-flow and dataflow architectures on the same chip die, a new architecture based on an asymmetric multicore processor is proposed.

The control-flow architectures are described as a most commonly used computer architecture today. Both multicore and manycore architectures are explained, as they are based on the same principles. A dataflow computing model assumes that data input flows through hardware as either a software or hardware dataflow implementation. In software dataflow, processors based on the control-flow paradigm process tasks based on their availability from the same queue (if there are any). In hardware dataflow architectures, the hardware is configured for a particular algorithm, and data input is streamed into the hardware, and the output is streamed back to the multicore processor for further processing. Hardware dataflow architectures are usually implemented with FPGAs.

Hybrid architectures employ asymmetric multicore and manycore computer architectures that are based on the control-flow and hardware dataflow architecture, all combined on the same chip die. Advantages include faster processing time, lower power consumption (and heating), and less space needed for the hardware.

Keywords: high performance computing; dataflow programming; manycore architectures; asymmetric cores.

1. INTRODUCTION

An exploration of using hardware for control-flow and hardware dataflow programming paradigms on the same chip die is presented.

In the following chapters, control-flow architecture principles, and computer architectures that are based on them are explained. Then, a description of dataflow architectures follows. Finally, a hybrid architecture that combines them is proposed. The communication between the control-flow hardware and the dataflow hardware on the same chip die might be achieved either using the cache memories that are shared between these hardware, or using the dedicated internal bus. As it will be explained, both approaches have their benefits and drawbacks.

2. MATERIALS AND METHODS

One of the main problems that computer vendors face in producing modern processors is the limited speed at which the hardware can operate. For decades, the speed has approximately doubled every two years. However, it appears that a 5GHz wall has been reached, and many commercial processors operate at 3GHz or less. Though the increase in speed is further possible, it would impose relatively high power consumption and cooling requirements. The alternative to improving the speed (or instruction rate) has been to increase the core count.

Hardware dataflow computing is based on the data traveling through the hardware. This solves many problems that computers based on control-flow face. However, the dataflow computing paradigm is suitable solely for certain types of high performance computing algorithms. Even algorithms that can be accelerated using the dataflow paradigm [1–5] include preparing data for parallel execution and collecting results, storing them in files, and auxiliary processing. As a result, dataflow hardware is usually connected to a multicore processor that controls it. It is the speed of the communication between the dataflow hardware and the control-flow hardware that limits the usability of this approach.

Another problem that arises from the combined control-flow and dataflow hardware is job scheduling. While certain jobs can be executed using both paradigms, other jobs can be executed solely on control-flow architectures.

Control-flow architectures are based on the von Neumann principle. Computer programs are written by programmers and compiled or interpreted. Both the compiler and the interpreter generate machine code understandable by the processor. Once the processor is ready to execute the instructions staged in memory, they are read and inserted into instruction registers. The instructions flow from the memory address register into the instruction register over the internal bus. Based on the instruction type, the processor might need to fetch an operand from memory as well. This involves the internal bus as well. Upon execution, the internal bus may be engaged once again, bringing the result into the register or memory. It should be noted that the internal bus might become the bottleneck. This limits the speed of the processing.

Control-flow hardware frequencies have been increasing for decades, approximately doubling every two years. This trend has ended, since further increases in speed impose overwhelmingly higher energy consumption and logical unit segmentation that cannot be accommodated for the projected speed-up that it might bring. Instead, as the density of packing transistors has grown lately, the number of processors has increased, which has evolved into multicore processor architectures. Increasing the number of cores does not proportionally increase performance, as many algorithms are not scalable enough. A potential solution is to exploit the dataflow paradigm [6–8].

Certain types of fast and relatively uniform processing are needed in computer graphics. In order to cope with this problem, graphics cards host thousands of small processing elements based on control-flow on the same chip. These architectures are often appropriately referred to as manycore architectures (relative to the core count of CPUs).

There are software and hardware dataflow architectures. Software dataflow architectures are based on the control-flow paradigm. Multiple processing elements are executing tasks from their common input queue, storing the results in their common output queue. The reason this is called dataflow computing is that data flows through the hardware based on the availability of processing elements and queues. Examples and discussions of these well-known architectures are available elsewhere [9-10].

Hardware dataflow works on a completely different principle. The hardware is configured for a certain algorithm, and the data flows through the hardware.

The hardware is usually implemented using FPGAs, enabling it to be reconfigured for the next job that the hardware is to execute. Dataflow processors can enhance high performance computing [11-12], as evidenced by the many high performance applications that have been accelerated using the dataflow hardware [13-18].

Transforming applications from the control-flow paradigm into the dataflow paradigm can be automated [19-20]. However, even with automating transformations, exploiting hardware dataflow still requires significant programming effort [21]. Many high performance algorithms are already implemented for dataflow hardware and available online [22-23], which can help programmers who are new to the dataflow paradigm boost the development of dataflow algorithms.

Dataflow hardware jobs usually last relatively long compared to those of control-flow type of processors. The reason for this lies in the facts that the dataflow hardware is suitable only for high-performance computing algorithms and that the hardware has to be configured prior to the execution of the algorithm. In order to be able to execute the algorithm faster than the processor based on the control-flow paradigm, it has to perform both the configuration and the calculation faster than a processor based on the control-flow paradigm would perform solely the calculation. The trend is that computers should have both control-flow and dataflow components [24].

The problem becomes more complicated when there are multiple tasks and when some can be executed solely on the control-flow hardware. These require different scheduling techniques than those used for multicore processors and cluster computing, which are discussed in existing research [25].

3. RESULTS

With growing capabilities in terms of the number of transistors per chip and the limited scalability of algorithms that multicore architectures usually execute, the question of whether to combine the control-flow and the dataflow hardware on the same chip die is reasonable. Further, the processor might also include the manycore architecture, besides multicore, and the dataflow on the same chip die. The research suggests the implementation of hybrid architectures might bring certain benefits, including faster processing, lower power-consumption, and less space needed for the hardware [26–27]. This is especially important when it comes to cluster and cloud computing. Previous research also proposes including the Internet of Things on the same chip die.

In addition to what has already been proposed, a hybrid processor may include asymmetric processors. Although all cores have the same instruction set, their microarchitectural properties may differ, so that there may be one core that is the fastest and the rest slower but of equal speed. Researchers have shown that appropriate scheduling can lead to a decrease in the probability of conflicts and an improvement in performance for transactions migration to the proposed asymmetric multiprocessor, based on the history of execution [28]. Their proposed solution is fully implemented in hardware. The claim is that it is possible to improve the performance by up to 14%. This serves as a proof that the hybrid processor proposed in this article can improve the performance of a similar hybrid processor with symmetrical multicore architecture [29], albeit for certain types of algorithms.

Figure 1 depicts the architecture of the proposed hybrid architecture.

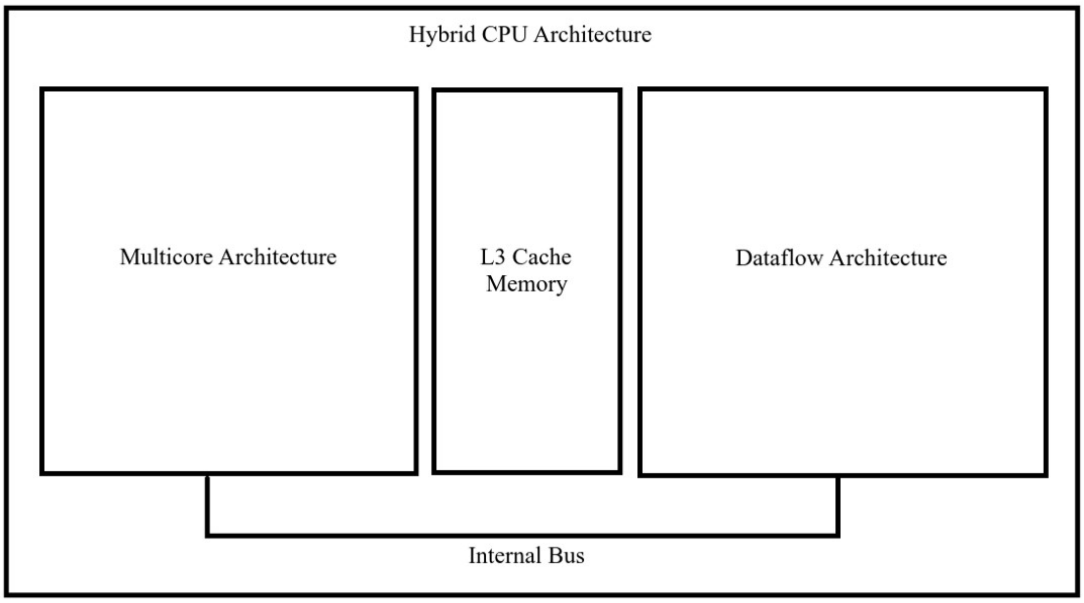


Figure 1. *Hybrid CPU architecture.*

4. DISCUSSION

The communication between the control-flow hardware (multicore) and dataflow hardware in the hybrid architecture might be achieved either using existing shared cache memories or using the internal bus. The first approach requires special care about the timing constraints, as two hardware in general do not operate at the same frequency. However, it doesn't require much extra hardware. The second approach requires additional control logic to cope with the specifics of the two hardware. This doesn't require introducing new logic but rather applying the existing mechanisms for communicating between the ordinary control-flow processor and considerably slower main memory. The communication in this case doesn't interfere with exploiting cache memories, but the drawback is that this approach requires more space on the chip die required for implementing the extra internal bus.

The proposed hybrid architecture can serve as a processor in a computer cluster or in a cloud [30], multiplying benefits with the number of processors working in parallel. Important domains of the combined control-flow and dataflow hybrid processor are the expected lifetime and counterfeit detection algorithms. The duration is expected to be the shortest of the three incorporated architectures, and possible counterfeit hybrid processor chips could be identified using existing statistical methods [31].

5. CONCLUSION

The improvement in computer architectures moved from incrementing the speed of processing toward increasing core counts. Additionally, computer graphics often requires much more processing per time, compared to the capacity of multicore processors, resulting in the introduction of so-called manycore architectures that include thousands of small processing elements based on the control-flow paradigm. Computer vendors that provide cloud infrastructure now offer dataflow processing that solves problems for processors based on the control-flow paradigm.

As the improvement in processor frequencies has dropped and the number of transistors per chip has increased, it is plausible for a single chip die to include multicore, manycore, and dataflow hardware. This requires special scheduling techniques to cope with the new needs. This work suggests that combining these computing architectures on the same chip is possible and has benefits, including: faster processing, lower power consumption, and less space needed for the hardware. Additional performance gains can be achieved with an asymetric multicore architecture. All these results can be especially important for cloud infrastructure and computer clusters.

Funding: N.K. is partially supported by the School of Electrical Engineering, University of Belgrade, Serbia, by the Institute of Physics Belgrade, contract no. 0801-1264/1, and by the Ministry of Education, Science, and Technological Development of the Republic of Serbia.

INSTITUTIONAL REVIEW BOARD STATEMENT:

Not applicable.

INFORMED CONSENT STATEMENT:

Not applicable.

CONFLICTS OF INTEREST:

The authors declare no conflict of interest.

REFERENCES

[1] A. Kos, V. Ranković, and S. Tomažič, "Sorting networks on Maxeler dataflow supercomputing systems," *Advances in Computers*, Vol. 96, pp. 139–186, 2015.

[2] V. Ranković, A. Kos, and V. Milutinović, "Bitonic merge sort implementation on the maxeler dataflow supercomputing system," *The IPSI BgD Transactions on Internet Research*, Vol. 9, No. 2, pp. 5-10, 2013.

[3] N. Korolija, V. Milutinovic, and S. Milosevic, "Accelerating conjugate gradient solver: temporal versus spatial data," *The IPSI BgD Transactions on Advanced Research*, Vol. 3, No. 1, pp. 21–25, 2007.

[4] V. Milutinovic, M. Kotlar, S. Stojanovic, I. Dundic, N. Trifunovic, and Z. Babovic, "Implementing Neural Networks by Using the DataFlow Paradigm," In *DataFlow Supercomputing Essentials*, New York: Springer Cham, 2017, pp. 3–44.

[5] V. Jelisavcic, I. Stojkovic, V. Milutinovic, and Z. Obradovic, "Fast learning of scale-free networks based on Cholesky factorization," *International Journal of Intelligent Systems*, Vol. 33, No. 6, pp. 1322-1339, 2018.

[6] M. J. Flynn, O. Mencer, V. Milutinovic, G. Rakocevic, P. Stenstrom, R. Trobec, and M. Valero, "Moving from petaflops to petadata," *Communications of the ACM*, Vol. 56, No. 5, pp. 39-42, 2013.

[7] A. Kos, S. Tomažič, J. Salom, N. Trifunovic, M. Valero, and V. Milutinovic, "New benchmarking methodology and programming model for big data processing," *International Journal of Distributed Sensor Networks*, Vol. 11, No. 8, 271752, 2015. Available: https://journals.sagepub.com/doi/10.1155/2015/271752

[8] N. Trifunovic, V. Milutinovic, J. Salom, and A. Kos, "Paradigm shift in big data supercomputing: dataflow vs. controlflow," *Journal of Big Data*, Vol. 2, No. 1, pp. 1-9, 2015.

[9] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, "Naiad: a timely dataflow system," In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, 2013, pp. 439-455.

[10] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, ... and U. Srivastava, "Building a high-level dataflow system on top of Map-Reduce: the Pig experience," *Proceedings of the VLDB Endowment*, Vol. 2, No. 2, pp. 1414-1425, 2009.

[11] V. Milutinović, J. Salom, N. Trifunović, and R. Giorgi, *Guide to dataflow supercomputing: Basic Concepts, Case Studies, and a Detailed Example*, Springer Cham, 2015. Available: https://link.springer.com/book/10.1007/978-3-319-16229-4

[12] A. R. Hurson and V. Milutinovic, *Dataflow Processing*. Cambridge, Massachusetts, United States: Academic Press, 2015.

[13] V. Milutinović, B. Furht, Z. Obradović, and N. Korolija, "Advances in high performance computing and related issues," *Mathematical Problems in Engineering*, Vol. 2016, 2016. Available: https://downloads.hindawi.com/journals/mpe/2016/2632306.pdf

[14] S. Stojanović, D. Bojić, and M. Bojović, "An overview of selected heterogeneous and reconfigurable architectures," *Advances in Computers*, Vol. 96, pp. 1-45, 2015.

[15] N. Korolija, T. Djukic, V. Milutinovic, and N. Filipovic, "Accelerating Lattice-Boltzmann method using Maxeler dataflow approach," *The IPSI BgD Transactions on Internet Research*, Vol. 9, No. 2, pp. 34–42, 2013. Available: http://tir.ipsitransactions.org/indexTIR_all.php

[16] S. Stojanović, D. Bojić, and V. Milutinović, "Solving Gross Pitaevskii equation using dataflow paradigm," *The IPSI BgD Transactions on Internet Research*, Vol. 17, 2013. Available: http://ipsitransactions.org/journals/papers/tir/2013july/p4.pdf

[17] I. Stanojević, M. Kovačević, and V. Šenk, "Application of maxeler dataflow supercomputing to spherical code design," In *Exploring the DataFlow Supercomputing Paradigm,* New York: Springer Cham, 2019, pp. 133-168.

[18] N. Bežanić, J. Popović-Božović, V. Milutinović, and I. Popović, "Implementation of the RSA Algorithm on a DataFlow Architecture," *IPSI BgD Transactions on Internet Research*, Vol. 9, No. 2, pp. 11-16, 2013.

[19] N. Korolija, J. Popović, M. Cvetanović, and M. Bojović, "Dataflow-based parallelization of control-flow algorithms," *Advances in Computers*, Vol. 104, pp. 73-124, 2017.

[20] V. Milutinovic, J. Salom, D. Veljovic, N. Korolija, D. Markovic, and L. Petrovic, "Transforming applications from the control flow to the dataflow paradigm," In *Dataflow Supercomputing Essentials,* New York: Springer Cham, 2017, pp. 107-129.

[21] J. Popovic, D. Bojic, and N. Korolija, "Analysis of task effort estimation accuracy based on use case point size," *IET Software*, Vol. 9, No. 6, pp. 166-173, 2015.

[22] N. Trifunovic, V. Milutinovic, N. Korolija, and G. Gaydadjiev, "An AppGallery for dataflow computing," *Journal of Big Data*, Vol. 3, No. 1, pp. 1-30, 2016.

[23] V. Milutinovic, J. Salom, D. Veljovic, N. Korolija, D. Markovic, and L. Petrovic, "Maxeler AppGallery Revisited," In *Dataflow Supercomputing Essentials,* New York: Springer Cham, 2017, pp. 3-18.

[24] A. Hurson, V. Milutinovic, "Special issue on dataflow supercomputing," *Advances in Computers*, Vol. 96, pp. 1–234, 2015.

[25] N. Korolija, D. Bojic, A. R. Hurson, and V. Milutinovic, "A runtime job scheduling algorithm for cluster architectures with dataflow accelerators," *Advances in Computers*, Vol. 126, pp. 201-245, 2022.

[26] V. Milutinović, E. S. Azer, K. Yoshimoto, G. Klimeck, M. Djordjevic, M. Kotlar, ... and I. Ratkovic, "The ultimate dataflow for ultimate supercomputers-on-a-chip, for scientific computing, geo physics, complex mathematics, and information processing," In 2021 10th Mediterranean Conference on Embedded Computing (MECO), 2021, pp. 1-6.

[27] V. Milutinović, N. Trifunović, N. Korolija, J. Popović, and D. Bojić, "Accelerating program execution using hybrid control flow and dataflow architectures," In 2017 25th Telecommunication Forum (TELFOR), 2017, pp. 1-4.

[28] Z. Sustran and J. Protic, "Migration in hardware transactional memory on asymmetric multiprocessor," *IEEE Access*, Vol. 9, pp. 69346-69364, 2021.

[29] J. Popović, V. Jelisavčić, and N. Korolija, "Hybrid Supercomputing Architectures for Artificial Intelligence: Analysis of Potentials," In 1st Serbian International Conference on Applied Artificial Intelligence (SICAAI), Kragujevac, Serbia, 2022.

[30] N. Korolija and A. Zamuda, "On Cloud-Supported Web-Based Integrated Development Environment for Programming DataFlow Architectures," In *Exploring the DataFlow Supercomputing Paradigm,* New York: Springer Cham, 2019, pp. 41-51

[31] K. Huang, Y. Liu, N. Korolija, J. M. Carulli, and Y. Makris, "Recycled IC detection based on statistical methods," *IEEE transactions on computer-aided design of integrated circuits and systems*, Vol. 34, No. 6, pp. 947-960, 2015.

Journal of COMPUTER AND FORENSIC SCIENCES

CFS 2022, Vol. 1, Issue 1, pp. 55–75
https://doi.org/10.5937/1-42712
Original research paper

Performance Modeling of File System Pairs in Hypervisor-Based Virtual Environment Applied on KVM Hypervisor Case Study

Borislav Đorđević[1], Valentina Timčenko[2], Nemanja Maček[3*], Mitko Bogdanoski[4]

[1] Mihajlo Pupin Institute, University of Belgrade, Serbia & Academy of Technical and Art Applied Studies, School of Electrical Engineering, Belgrade, Serbia; borislav.djordjevic@pupin.rs

[2] Mihajlo Pupin Institute, University of Belgrade, Belgrade, Serbia; valentina.timcenko@pupin.rs

[3] Academy of Technical and Art Applied Studies, School of Electrical and Computer Engineering, Belgrade, Serbia & University Business Academy in Novi Sad, Serbia & SECIT Security Consulting, Serbia; macek.nemanja@gmail.com

[4] Military Academy General Mihailo Apostolski, Skopje, RN Macedonia, mitko.bogdanoski@ugd.edu.mk

* Corresponding author: macek.nemanja@gmail.com

Received: 2022-10-08 • Accepted: 2022-12-05 • Published: 2022-12-30

Abstract: This paper proposed an approach to mathematical modeling of the file system performance in a hypervisor-based virtual environment, with special focus on the file system pair interactions. The main goal of this research is to conduct an in-depth analysis of the filesystem pair behavior with respect to the performance costs originating from the employed technologies, such as H-Trees, B-Trees and Copy-on-Write/Overwrite update method, and different application workload types. The modeling provides a collection of hypotheses about the expected behavior. The modeling and the hypotheses are validated based on the results obtained for a specific case study. Our study reports on a file system performance comparison in the context of KVM hypervisor-based full hardware virtualization, application-level benchmarking, and 64-bit Linux filesystems Ext4, XFS, and Btrfs. The Filebench benchmark tool is applied for comprehensive testing of the filesystem performance under fair-play conditions. According to the obtained results, we provide a set of recommendations (i.e., a Knowledge Data Base) for optimal filesystem pair selection for the KVM hypervisor. Finally, it is important to note that the proposed modeling is also applicable to other hypervisor-based virtualizations.

Keywords: filesystems, operating systems, performance evaluation, platform virtualization, virtual machine monitors.

1. INTRODUCTION

Virtualization is a method for organizing computer resources within multiple operational environments, by means of hardware and software partitioning, time-sharing, partial or complete hardware emulation, paravirtualization, etc. [1–3]. Modern approaches to virtualization include two fundamental methods: hypervisor-based virtualization and container-based virtualization. Virtualization has gained popularity in many different areas, including cloud computing (CC), Internet of Things, cyber physical systems, and big data. It represents the core technology behind proper cloud computing functioning, which mainly depends on the sophistication of its design and implementation.

Quality of Service (QoS) is another important aspect to consider. QoS represents guaranteed levels of performance and availability of a service provided to users [4–6]. A large number of factors affect QoS, but the three primary aspects are computing, storage, and network performance. In a hypervisor-based virtual environment, there are three fundamental components: a host operating system, a hypervisor, and guest operating systems. A host operating system represents the driver support and management layer for virtualization, and a hypervisor is a software layer that serves as an intermediary between the host operating system and virtual machines, i.e., hypervisors behave as kernels for virtualization.

A hypervisor and a host operating system create a virtual environment for guest operating systems. This environment does not necessarily have the same characteristics as the physical environment. In the context of hypervisor-based virtualization, three types of virtualization are dominant: full hardware virtualization, paravirtualization, and operating system (OS) level virtualization. The full hardware virtualization represents complete hardware emulation so that the installation and execution of guest OSs require no additional adaptations. This type of virtualization is the most appropriate for employment, but it suffers from low performance level, which can be boosted by using Intel VT-x or AMD-V as special CPU features for virtualization. Paravirtualization requires significant modifications to the host and guest OSs but enables significantly better hypervisor-based performance of virtual machines. OS-level virtualization is based on applying the same kernel for several OS instances.

There are two types of hypervisors. Type-1 hypervisors (i.e., so-called native hypervisors) execute directly on physical hardware, and their prominent instances include ESXi, Xen, KVM/Proxmox, and Hyper-V. Type-2 hypervisors execute as applications within a host OS, and their prominent instances include Oracle-Virtual Box and VMware Workstation. Normally, type-1 hypervisors have much better performance than type-2 hypervisors. Related to host OSs, type-1 hypervisors can be Linux-based (e.g., ESXi, Xen, KVM/Proxmox), and MS Windows-based (e.g., Hyper-V).

Hypervisor-based virtualization builds upon interactive pairs of OSs, i.e., interaction between a host OS and one or more potentially different guest OSs. Both the host and guest OSs support a number of filesystem (FS) types. The host OS stores VM image files into the underlying filesystems, whereas the guest OSs employ one or more of these filesystems. As hypervisor-based virtualization imposes an OS pair, it also establishes interactive FS pairs. There are many available combinations of OSs and underlying FSs, but the overall FS performance may significantly vary among different FS pairs depending on the characteristics of the workload (WL). Out of a relatively large number of factors that determine FS performance, we focus on the influence of the interactive FS pair. This paper evaluates the performance of different filesystem pairs in a virtualization environment, and the obtained results are aimed at being applied in the context of CC.

2. RELATED WORK, OBJECTIVE AND MOTIVATION

The paper reports on a FS performance analysis of FS pairs for type-1 hypervisors. We emphasize that FS performance is a fundamental factor for achieving an adequate level of QoS in a CC environment. In related work, FS performance in Virtual Environment (VE) has been analyzed in different ways. The most common approach includes the FS performance comparison of different hypervisors, such as KVM, VMWare, Xen, and Hyper-V. This approach relies on the use of filesystem benchmark applications such as HD Tune Pro, Bonnie++, Iozone, LMbench, LINPACK, etc. For details on certain performance comparisons, reader may cf. [7–12].

Some evaluation approaches include the analysis of I/O speed with respects to overall IO performances in a virtual environment for wide range of cloud applications [13–14].

In some papers, the experimental results relate to the impact of the estimated costs for the realization of cloud technologies [15]. Further work is dedicated to the question of virtual infrastructure management, showing cloud resources can be limited in order to respond to dynamic changes in a VE [16].

Finally, certain research efforts have been dedicated to performing comparative analysis of the modern hypervisors, which is in line with the approach applied in our paper [12], [17–23].

The main contribution of this study relates to comprehensive mathematical modeling of the FS performance in a VE employing type-1 hypervisors, with special focus on the interactive FS pairs. The FS pair modeling includes many factors that can be explored as if being independent or mutually correlated. The model proposed in this paper is applicable to most of the type-1 VE. The basic idea underlying our approach is to provide a specific mathematical model, apply it to a particular case study, and then interpret and validate the experimental results. We also contribute by proposing a Knowledge Data Base (KDB) comprising the collection of optimal FS pairs, available to VE administrators.

Compared to related work, we believe that our study introduces more comprehensive modeling of the FS performance in VE. At the practical level, while most of the related approaches consider just a single case study [17–23], we consider three case studies. Compared to related work, our main focus is FS pair modeling and KDB with the optimal FS pairs. Like most of the related studies, we show that there is no optimal FS pair that suits all possible use cases, but that the optimal FS pairs depend on WL and many other factors and change over time with the emergence of new FS versions and other VE factors.

Our paper presents the FS performance evaluation in fair-play conditions, i.e., it reports on the performances of the FS pairs formed from the selected FS types (Ext4, XFS, and Btrfs) applied with KVM as a representative of the type-1 hypervisors. The fair-play conditions assume the use of identical hardware for all the evaluated elements, the same characteristics of the generated virtual machines (VM) and the identical version of the guest OS. We select KVM with full hardware virtualization, whereas FS types for host and guest FS are Ext4, XFS, and Btrfs. In addition, we use Filebench, which is a modern multi-threaded based benchmark, easily configurable, and adequate for the simulation of real-world applications. Four different test workloads are selected to simulate realistic workload conditions and applications: web server, e-mail server, fileserver, and random file access. The research protocol can be briefly described as follows. We introduce the mathematical model for FS pairs, select FS pairs for the evaluation, define the hypotheses related to the expected behavior of FS pairs, and finally proceed with the benchmark measurements. The results are interpreted in the context of the introduced mathematical model and hypotheses. We perform an in-depth analysis of guest and host OS FS behavior with different workload types and believe that the reported results are insightful for system administrators dealing with virtualization and some QoS issues in small-scale CC environments.

3. SELECTED TYPE-1 HYPERVISORS AND FILESYSTEM PAIRS

The type-1 hypervisor-based virtualization representatives are the following: ESXi with the original VMware FHV (Full Hardware Virtualization); Xen with two virtualization types FHV (QEMU based) and PV (paravirtualization) which is suitable for open-source PV guests; KVM/Proxmox with the FHV (QEMU based) virtualization; and Hyper-V with two kind of virtualizations, FHV (Microsoft original) and PV (paravirtualization) for Microsoft Windows OS.

In a hypervisor-based virtualization architecture, there are three fundamental components: the hypervisor as a kernel optimized for a particular VE, a host OS, and a guest OS. The host OS supplies drivers and supports management. It contains a host FS as a storage for host OS (hOS) and VM images. A VM contains a guest OS (gOS) and guest FSs which serve as storage for guest OS and guest applications. The hypervisor-based virtualization imposes interactive FS pairs (gFS-on-hFS). Each OS can support several modern FS types that undergo long-term development. Most of them are 64-bit, extent based, with accelerating techniques for allocation and searching (H-Tree/B-Tree). The overwriting or Copy-on-Write (CoW) techniques are adopted as write/update methods. The FS performance depends on the file caching, journaling, and different tunable parameters.

For a VE, both the host and guest OSs can be Linux-based or Microsoft Windows-based. Similarly, hypervisors with an accompanying host OS can be a Linux-based (e.g., ESXi, Xen and KVM/Proxmox) or Windows-based (e.g., Hyper-V). Linux OSs support a number of FS types, whereas the Microsoft Windows OS family implements only two FS types: NTFS and FAT (the latter of which is unsuitable for this purpose).

Thus, considering the number of available FS pairs (gFS on hFS) we can observe the following:

• Linux-based hypervisors and Linux VMs can include a very large number of FS pairs (gFS on hFS);

• Microsoft Windows-based hypervisors and Linux VMs can still include a significant number of FS pairs (gFS on NTFS);

• Microsoft Windows hypervisors and Windows VMs include only one FS pair (NTFS on NTFS).

Since the number of FS types is large, in the reported study we focused on the three popular 64-bit FSs: Ext4, XFS, and Btrfs. The main reason behind this decision is that all of them are modern, extent-based, and commonly used in Linux environments. Ext4 is robust and powerful in its performance, with relatively simple technologies (such as H-Tree, extent-Tree, pre-allocation, delayed allocation), and it is unlikely that any significant progress of its features will be made in the near future. XFS and Btrfs are two modern and promising FSs with data structures based on B+ Trees. The B+ Tree technology is constantly being developed and improved. Btrfs relies on the CoW method, which is novel when compared with the traditionally applied overwrite methods. This study is particularly focused on the case when virtualization is applied to both the host and guest Linux OSs. The performance of the chosen FSs in a VE pair is different from their performance in a real-life hardware environment. A very interesting comparison between the aforementioned FSs is presented in this paper as a case study encompassing a combination of 9 (3x3) FS pairs.

In the rest of this section, we briefly describe the three chosen Linux FSs.

Ext4 is a native Linux FS developed to resolve the capability and scalability issues of its predecessor (ext3 FS) caused by double and triple indirect block mapping characteristics. Ext4 manages storage in extents (a range of continuous physical blocks that improve large file performance and reduce fragmentation). It employs a tree-based index to represent files and directories in the form of H-Trees [24–25]. A write-ahead journal is applied to ensure the operation atomicity, and the checksumming is performed on the journal, but not on the user data. Although it has many advantages over its predecessor (such as extents, persistent pre-allocation, delayed allocation, and improved timestamps), the backward compatibility enforces some limitations (e.g., no support for snapshots).

XFS was originally developed as a native Silicon Graphics IRIX FS and ported to Linux in 2001. Nowadays, it is supported by most Linux distributions and some of them recommend it as the default FS for home or boot partitions. XFS is a high-performance 64-bit FS that allocates space in extents with data stored in B+ Trees [26]. The efficient allocation of free extents is achieved by dual indexing (one tree is indexed by the size, and the other by the starting block of the free extent), whereas the delayed allocation prevents FS fragmentation. Although snapshots are not supported and the underlying volume manager is expected to support that operation, the meta-data journaling and write barriers ensure data consistency. Extreme scalability of I/O threads and FS bandwidth originate from the parallel execution of I/O operations. The issue of addressing slow meta-data operations, which result in poor performance when write operations are performed on a large number of small files, has been partially resolved with a delayed logging feature.

B-Tree FS (Btrfs) is a native CoW Linux FS designed to offer more efficient storage management and better data integrity features. It aims at solving scalability problems for larger and faster storage, such as lack of pooling, snapshots, checksums, integral multi-device spanning, and built-in RAID support. The FS layout is based on a forest of CoW friendly B-Trees [27–31]. The main idea behind the CoW friendly B-Trees is to use standard B+ Tree construction, employ top-down update procedure, remove leaf-chaining, and use lazy reference-counting for space management. Based on the CoW technique, the FS may be self-healing in some configurations. Disk blocks are managed in extents, with checksumming being performed for the purpose of integrity, and reference counting for the purpose of space reclamation. A vast variety of other useful features are implemented in Btrfs, including online defragmentation, online volume growth and shrinking, online block device addition and removal, online balancing, online data scrubbing, subvolumes, hierarchical per-subvolume quotas, and out-of-band data de-duplication. The B-Tree is a data structure that stores generic items organized by a key. Nodes contain only keys and pointers to the child node or leaf below, whereas leaves contain the actual variable sized data of the Tree. As they are tailored to systems reading and writing large blocks of data, B-Trees are suitable data structures for databases or FSs. Thus, FSs use B-Trees to search directories and extent descriptors, and for file allocation and file retrieval. Btrfs is particularly organized as a forest of B-Trees.

4. MATERIALS AND METHODS

In general, the proposed mathematical modeling of FS pair performance includes a large number of factors, but in this research, special attention is dedicated to interactive FS pairs. The modeling encompasses the following characteristics: the Workload (WL), VMs (with the accompanying gOS and gOS FS), the hypervisors, and the gOS and hOS FS. For the purpose of FS performance evaluation, the benchmark or real-life applications can be used, where all kinds of test procedures generate specific FS WLs. For each workload, we consider the parameter TW representing the total processing time. Each workload contains a mix of four cycle types: random reading (RR), random writing (RW), sequential reading (SR), and sequential writing (SW), whereas writing can be synchronous or asynchronous, so writing performance depends significantly on the FS caching.

In a given FS, each workload generates different kinds of operations related to directories, metadata, free lists, file blocks, journaling, and housekeeping (HK).

In hypervisor-based virtualization, the analysis of the workload processing time is quite complex. The FS performance in VEs depends on a number of factors originating from the type of virtualization applied (FHV, PV), hOS and gOS. Additionally, considering the context of the hypervisor VE environment, the overall data path becomes quite complex and relies on six components: application (benchmark), gOS kernel, guest OS FS, hypervisor as hOS kernel, VM image file, and host OS FS.

Figure 1 depicts an overview of the overall data path of a workload. The path is created by four objects (benchmark, gOS FS, VMI, and hOS FS), and two kernels (gOS kernel and hypervisor as hOS kernel).

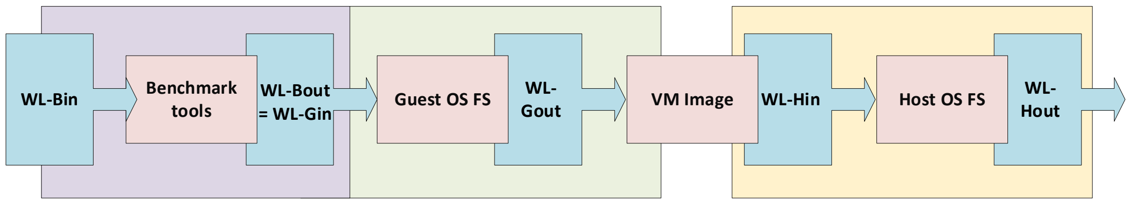


Figure 1. *Data path in virtualized environment.*

The benchmark has its own definition at the beginning of the procedure, and it provides a framework for the generation of the workload at the input (WL-Bin), taking into account the following parameters: the total number of workload files, the mean tree depth, the average file size, the file read/write block size, and the mean size of the appending block.

The input files are benchmarked with a range of different file operations such as opening, creation, deletion, reading, writing, and file appending. The sequence of the applied operations represents the benchmark output workload (WL-Bout). WL-Bout is applied to the gOS FS through the gOS kernel. It generates the guest output workload, WL-Gout, that consists of file block, inode, extent, free list, and directory read/write operations in the guest FS. The caching of gOS FS has a strong impact on WL-Gout. The WL-Gout component can be assumed to be a function of the benchmark request characteristics and the gOS FS processing procedures. This processing includes gOS FS features, gOS caching and virtual disk drivers.

The output workload from gOS FS is further redirected to the hypervisor, which maps it to a large VMI file. In other words, WL-Gout is mapped to a VMI file through the hypervisor, thus mapping all read/write operations onto VMI file operations. This mapping is marked as WL-Hin. WL-Hin is executed in the hostOS FS, resulting in a final sequence WL-Hout which comprises the following components: file block, inode, extent, freelist, and directory read/write operations in the host FS. Due to a large image file, the extent of read/write operations and the write method (overwrite or CoW) can have a great influence on the performance of WL-Hout. The final processing includes hOS FS features, hOS caching, and physical disk drivers.

The whole data path depends on various factors, including the characteristics of FS types on the guest and host sides, the file caching on the guest and host sides (i.e., a specific co-operation of these two caches), a large VMI file, the hypervisor interconnection of virtual and physical disk drivers, the hypervisor-CPU scheduling, and other.

For parameter T_w in a given VE, we consider six components (cf. equation 1):

$$T_w = f\left(App, g-kernel, g-FS, H-proc, Hyp-proc, h-FS\right) \tag{1}$$

1. Application (App) represents the interaction between WL-Bin as the benchmark inputs (definitions) and WL-Bout as the benchmark request for the guest OS FS. The selected application generates WL-Bout with random and sequential components.

2. g-kernel represents the processing time of the gOS kernel which takes the WL-Bout requests and forwards them to gOS FS.

3. Guest OS FS processing, g-FS, is a component targeting the gFS processing, which includes the gOS FS features, gOS FS caching, and virtual disk drivers. This component is very similar to the 6th component, h-FS. For both of these components (i.e., the 3rd and 6th components), the time for the OS-FS processing is represented by the function of the FS processing and FS cache processing (cf. equation 2):

$$g-FS = f(FSproc, FScache), \quad h-FS = f(FSproc, FScache) \tag{2}$$

4. Virtual hardware processing, *VH-proc*, represents the processing time of the virtual disk hardware. *VH-proc* strongly depends on the type of virtualization.

5. Hypervisor processing time, *Hyp-proc*, is the time necessary for the hypervisor to receive the requests from the VM (virtual disk driver) and forward them to the host VMI file. The FS requests from gFS (gOS-FS) are forwarded to hFS (hOS-FS) via the hypervisor and mapped through the VMI file, whereas many hypervisor parameters affect the FS performance.

6. Host OS FS processing, *h-FS*, is a component targeting the host FS processing, which includes the hOS FS features, hOS FS caching, and physical disk drivers. It works with a large VMI file and is a function of the FS processing and cache processing (cf. equation 2).

Some components of equation (1) are closely interrelated, especially the 3rd component and the 6th component. Virtual machines include a guest OS, which support several gOS FS types. Also, each hypervisor is related to its own hOS, which provides the virtual disk drivers and physical disk drivers. The hOS can support one or several hOS FS types.

In hypervisor-based VE, we must consider a FS pair, i.e., g-FS/h-FS, and consider the rather complex interaction between two FS caches. The interaction of FS pair is given in equation (3):

$$T_{FSpair} = f(gFSt/hFSt, gFSc/hFSc, HypFSparam) \tag{3}$$

The first component in equation (3), i.e., *gFSt/hFSt*, relates to FS types in the underlying FS pair. We recall that FS types may have different characteristics, as discussed in Section III.

The second component of equation (3), i.e., *gFSc/hFSc*, represents a pair of FS caches. These caches can be cooperative with Write Back (WB) or Write-through (WT) semantics or exclusive (none mode) when the hypervisor excludes the hOS FS cache for VMs.

The third component of equation (3), i.e., *HypFS param*, is related to the hypervisor tunable parameters. Each hypervisor has a number of tunable parameters, and some of them are significant for FS performance, such as CPU scheduling.

It is important to note that in this study we primarily focus on newer versions of FSs, gOSs and guest kernels, hypervisors with hOS, and CPU models with HW extensions. In a Linux-based VE, we consider three popular FSs, i.e., Ext4, XFS, and Btrfs, which allows for the generation of 9 FS pairs. We model the performance of these 64-bit Linux FS indexed by different techniques: B+ Trees, H-Trees, extent-Trees, linear lists, linked lists, etc.

For a B-Tree of order d and with n records, the cost of all operation processing operations grows at logarithmic rate, as $log_d(n)$. For all four types of actions (insertion, retrieval, updating, deleting), general equations for a B-Tree are:

$$T_{B+Tree}(mngmnt) \approx O(log_d(n) \tag{4}$$

$$T_{B+Tree}(mngmnt) \approx f(indexes, keys, nodes, balancing, etc) \tag{5}$$

For all writing operations, Btrfs employs the CoW method (cf. eq. 6), whereas XFS and Ext4 employ the update (overwrite) method (cf. eq. 7).

$$Twriting=f(CoW) \quad (6)$$

$$T_{writing} = f(overwrite) \quad (7)$$

For directory operations, Btrfs and XFS employ B+ Trees, due to which T_{dir} exposes logarithmic cost as in eqs. (4) and (5), whereas Ext4 employs H-Trees (cf. eq. 8).

$$T_{Dir} = T_{Htree}(mngmnt) = f(hash\_function, hash\_colision) \quad (8)$$

For metadata operations, Btrfs and XFS employ B+ Trees, due to which T_{meta} exposes logarithmic cost as in eqs. (4) and (5), whereas Ext4 employs linear inode table (cf. eq. 9).

$$T_{Meta} = T_{linear-inode-table}(mngmnt) \quad (9)$$

For free list operations such as free inode, free block, and free extent lists, Btrfs and XFS employ B+ Trees, due to which TFL exposes logarithmic cost as in eqs. (4) and (5), whereas Ext4 employs linear bitmap (cf. eq. 10).

$$T_{E} = T_{linear-bitmap}(mngmnt) \quad (10)$$

For FileBlock accesses, Btrfs and XFS employ B+ Trees, due to which $T_{fileblock}$ exposes logarithmic cost as defined in eqs. (4) and (5), whereas Ext4 employs H-Trees in extent Tree structures (cf. eq. 11).

$$T_{FileBlock} = T_{Extent\_tree}(mngmnt) \quad (11)$$

For housekeeping, the most activities are performed by Btrfs (CRC for all operations), eq. (12), whereas XFS and Ext4 are much simpler (CRC for journaling), as given in eq. (13).

$$T_{HK} = f(data\_CRC, metadata\_CRC, Journaling\_CRC) \quad (12)$$

$$T_{HK} = f(Journaling\_CRC) \quad (13)$$

The performance costs of I/O operations are summarized in Table 1. We introduce labels *Cx.y* to denote performance costs of operation *x* performed on FS *y*. Every feature based on a B+ Tree has a logarithmic cost defined in eqs. (4) and (5). As Ext4 relies on the use of linear lists and H-Trees, Table 1 shows the cost values that are dependent on the hash operation and those that are subordinated by the linear search operation. Although B+ Tree has its general principles of generation and file manipulation, there is a cost difference between the FSs that rely on a B+ Tree as they employ this data structure in different ways. This includes differences in the organization of the indexes, keys, nodes, search, balancing, and other techniques. A particular difference is related to the use of the B-Tree in Btrfs and XFS environments, as Btrfs applies the CoW method to the underlying B-Tree, whereas XFS applies the traditional overwrite method. Btrfs FS employs the CoW update method, whereas XFS and Ext4 employ the overwrite update method for writing operations. The CoW method is expected to have a significantly higher cost.

When working with directories, Btrfs and XFS employ B+ Trees and operate with logarithmic cost properties (eqs. 4 and 5), where *n* is the number of objects in the directory. Ext4 employs H-Trees, resulting in item searching being penalized by the hash performance, which depends on the hash function and the hash collision eq. (8).

The meta-data operations on Btrfs and XFS retain B+ Tree logarithmic cost properties dependent on the number of inodes in the filesystem, cf. eqs. (4) and (5). Ext4 manages a linear inode table and has linear cost properties, eq. (9).

Btrfs and XFS manage free lists (free inode list, free blocks list and free extents) with logarithmic cost properties, cf. eqs. (4) and (5), where *n* is number of objects in Free Lists (blocks, extents, inodes). Ext4 employs a linear bitmap for its free lists and has linear cost properties, cf. eq. (10).

Direct file block access consists of different time components: item retrieval, item reading, item appending, item writing, and item deletion. Each file is made up of data extents, thus the direct file block manipulation is practically extent-Trees manipulation. Direct file block access is performed with a logarithmic cost, both for Btrfs and XFS cf. eqs. (4) and (5), where *n* is the number of file extents. Ext4 employs H-Trees for extents and thus suffers from H-Tree penalties, cf. eq. (11).

The housekeeping operations on Btrfs depend on data, meta-data, and journaling CRC, cf. eq. (12), which means that they are very intensive, especially in the case of a large number of write operations. The housekeeping operations on XFS and Ext4 depend only on the journaling CRC, cf. eq. (13). Thus, the housekeeping costs are small for Ext4 and XFS, but can be significant for Btrfs.

Table 1. *Performance Costs.*

Operation / FS	Ext4	cost	XFS	Cost	Btrfs	cost
Update method (writing)	Overwrite	C1.1	Overwrite	C2.1	CoW	C3.1
Directory operations	H-Tree	C1.2	B+ Tree	C2.2	B+ Tree	C3.2
Meta-data operations	Linear inode table	C1.3	B+ Tree	C2.3	B+ Tree	C3.3
Free lists operations	Linear bitmap	C1.4	B+ Tree	C2.4	B+ Tree	C3.4
File block access	H-Tree	C1.5	B+ Tree	C2.5	B+ Tree	C3.5
House keeping	Journaling CRC	C1.6	Journaling CRC	C2.6	Data, meta-data, and journaling CRC	C3.6
Small file embedding	None	C1.7	Moderate performance	C2.7	Good performance	C3.7

5. THE HYPOTHESES

This research employs Linux both as the guest and host OS. Different features of Ext4, XFS and Btrfs as host and guest OS underlying FSs are analyzed, and the following assumptions about the expected I/O performance are adopted.

B+ Trees boost data retrieval, cf. eqs. (4) and (5). Each B-Tree based FS has its own B-Tree organization, which has a direct impact on performance. This hypothesis applies to all B+ Tree-based FS reading operations, including directory, meta-data, and free-list operations as well as direct file-block access (Table 1, costs C2.2, C2.3, C2.4, C2.5, C3.2, C3.3, C3.4, and C3.5). Small file embedding is expected to have a major positive impact on random performance (Table 1, costs C1.7, C2.7, and C3.7).

CoW has a negative impact on write performances due to changed pages and CoW-ed extents (cached and written elsewhere) (cost C3.1) when compared to overwrite update method (costs C1.1 and C1.2). Garbage collection is also required for CoW.

CoW turns small, random updates into sequential cycles, thus providing the workload with more sequentially. The negative impact of CoW on sequential writing is expected to be more significant when compared to random writing operations (cost C3.1).

Housekeeping is expected to provide the largest negative impact on the Btrfs performance when compared to the Ext4 and XFS performance (costs C3.6 related to cost C1.6 and C2.6).

For the interpretation of the FS performance, we consider interactive FS pairs, and we think that each FS type in FS pair cannot be analyzed separately, but only as an interactive FS-pair. The gOS FS has a specific behavior in the FS-pair. It works similarly to physical conditions by generating a sequence of requests (files/directory operations) for the gOS FS, which operates based on the gOS FS features, gOS caching, and virtual disk drivers. For FHV, virtual disk drivers are identical to physical disk drivers, whereas virtual disk is represented as a large VMI file. Each gOS FS generates a specific WL-Gout which is mapped to a large VMI and then the WL-Hin is generated. For each gOS FS observed in the same benchmark, a quite different WL-Hin is generated. Each hOS FS works specifically, it gets a WL-Hin that does not look like any application, and it is the output of the whole FS with a benchmark as input. The hOS FS works in real physical conditions – processing a sequence request for one large file, VMI. The gOS FS operates based on the hOS FS features, hOS caching, physical disk drivers, and physical disks. In short, gOS FS generates WL-Gout, a complex sequence of requests for hOS FS, which hOS FS processes through a large VMI file. Complex interaction of two FS is the reason why both FSs must be viewed integrally as a pair.

Clear indications of the best/worst host and guest OS FS pairs are expected, where one or more of those pairs will provide the highest/weakest I/O performance.

The aforementioned assumptions are experimentally validated with a set of performance measurements (synthetic benchmarking), and the interpretation of the results is presented in the next section of the paper.

6. EXPERIMENTAL EVALUATION

We consider three different case studies of the KVM hypervisor-based VE. In each case study, we apply the same protocol but different hardware. We employ two different CPUs, two magnetic hard drives, and two kernel versions of the same Linux distribution. For the first case study (CS1), experiments were performed on the Intel Xeon E3110 @ 3.00GHz, 8GB DDR3-RAM, with Seagate Barracuda 500GB SATA-3 hard disk (7200 rpm, 6 GB/s). Centos 7.2 with Linux Kernel 3.10.0-327.36.3.el7.x86_64 is chosen as the native host for KVM hypervisors and the guest operating systems. For the second case study (CS2), experiments were performed on the Intel Xeon E3110 @ 3.00GHz, 8GB DDR3-RAM, with Toshiba DT01ACA050 500GB SATA-3 hard disk (7200 rpm, 6Gb/s). Centos 7.9 with Linux Kernel 3.10.0-1160.21.1.el7.x86_64 is chosen as the native host for KVM hypervisors and the guest OSs. For the third case study (CS3), experiments were performed on the dual core AMD Ryzen 5 3400G @ 3.7GHz, 8GB DDR4-RAM, with Toshiba DT01ACA050 500GB SATA-3 hard disk (7200 rpm, 6Gb/s). Centos 7.9 with Linux Kernel 3.10.0-1160.21.1.el7.x86_64 is chosen as the native host for KVM hypervisors and the guest OSs.

Because of the relatively small amount of available RAM (8GB), each of the three VMs was assigned 2GB of RAM, thus allowing the host OS to operate with the remaining RAM. The experiment was performed on one, two, and three VMs simultaneously to examine the impact of the host OS caching on the KVM virtualization, proceeding with the experiments for the writeback (WB) cache mode.

The hypervisor's random and sequential performances are tested in the Filebench benchmark environment, setting four different application workloads.

For practical reasons, the obtained experimental results are just partially presented in this paper, i.e., we discuss the third case study (CS3), in which we consider the performance for four WL, for native performance, and 1VM, (cf. Figs. 2–5).

7. DISCUSSION

Testing of each server workload is briefly discussed below. Measurements are performed for a Web-server, mail-server, and file-server workloads as well as for a random file access. One Virtual Machine is used for each test. Different pairs of the aforementioned filesystems were considered during the test.

For each evaluated workload, starting with the WL-Bin, we have measured the characteristics of the WL-Hout (Figure 1), taking into consideration the processing time and the overall throughput. For each workload, we use the following main criteria: the best and worst FS pair results (gOS FS on hOS FS) considering all pairs, and presence of FSs in the best and worst combination on the guest and host side. For FS names, we will use abbreviation, Btrfs as B, Ext4 as E, XFS as X.

In the Web-WL, there are 100 threads, where each thread selects 10 of the 1000 files in directory-Tree, makes the 10 sequences of open-read-close operations and 11th sequence as log append operation. The web-workload is characterized by the dominant data and metadata random reads. There are small components of the data and metadata random writes in log file, which are synchronous. The dominant random reads indicate a small influence of the WB cache mode on the guest/host side. Results for native and 1VM performance for web WL are depicted in Fig. 2.

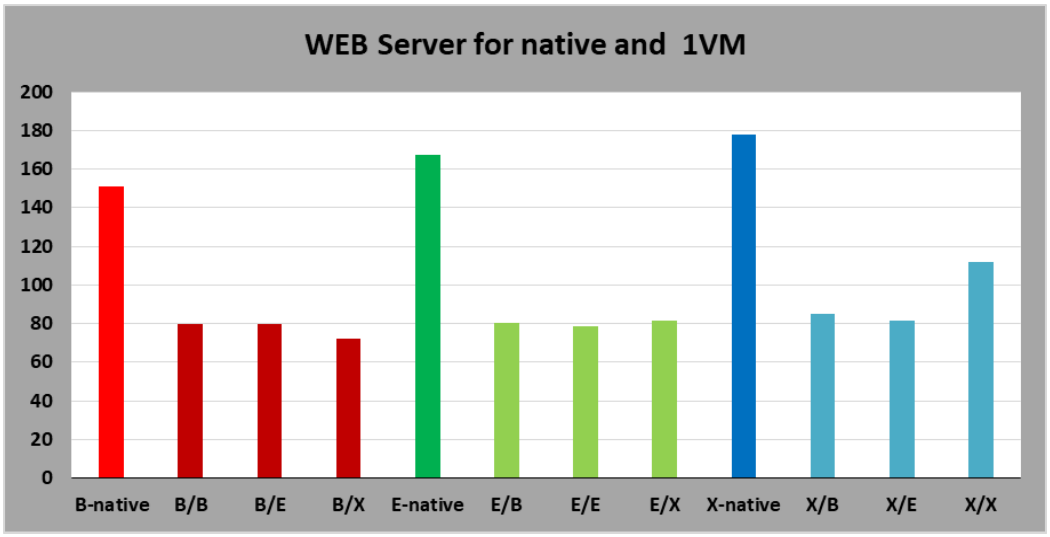


Figure 2. *WebServer throughput [MB/s] for native and 1VM environment.*

For all three case studies, we detected the best FS pairs for Web-WL: X/X, E/E, E/B, B/E, E/X, X/E, among which X/X, E/E, E/B stand out. In the best FS pairs, XFS and Ext4 dominate as gFS and hFS.

Particularly bad FS pairs for Web-WL are: B/X, B/B, X/B. In the worst FS pairs, Btrfs dominates as gFS and hFS.

When considering the best/worst FS pairs, for Web-WL X/X or Ext4 should be selected with all hFS, and B/X and B/B should be avoided. XFS and Ext4 are good choices for making FS pairs, whereas Btrfs should be avoided, especially on the guest side.

For RR components, on both sides (guest/host), B-Trees have the best performance for data retrieval. B+Trees of XFS used for directories [C2.2], inodes [C2.3], extents [C.2.5], FreeList [C2.4] and FileBlock [C2.5] are crucial for good random read performances. Similar insights regarding random read performances hold for Btrfs [C3.2, C3.3, C3.4, and C3.5]. However, there is a small random write component, hence the write methods still have an important role. Because of the overwrite method applied instead of CoW, XFS outperforms Btrfs [C2.1 and C3.1] on the guest/host sides.

We argue that the Btrfs CoW penalty is the main reason underlying the bad FS pairs such as B/X, X/B, B/B. Ext4 with its relatively simple but fast technologies (H-Tree, extent-Tree C1.1, C1.5), exhibits good characteristics in the best combinations (E/E and E/B) and behaves quite well on both the guest/host sides.

In the Mail-WL, there are 16 threads, each of which selects four of the 1000 files in a single directory. With these files, thread makes the following sequences of operations: delete, create-append-fsync-close, open-read-append-fsync-close, open-read-close. In the case of the varmail-workload, synchronous random writes are dominant for both data and metadata. A large amount of the random reads for both data files and metadata are noticeable. Random writes are synchronous, so the data writing must reach the disk drive. Due to the dominant RR/RW-synchronous cycles, there is a small impact of WB caching on the guest/host sides. Housekeeping is also important to consider as it generates many writes. The results for native and 1VM performance for mail WL are depicted in Fig. 3.

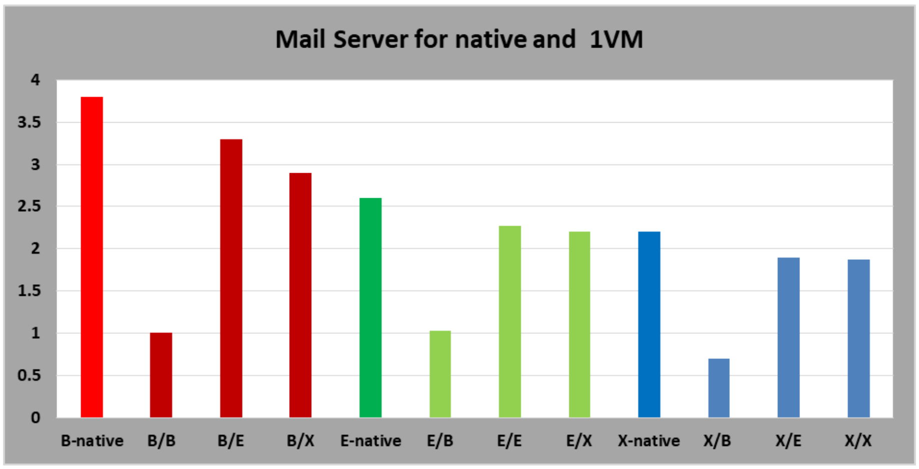


Figure 3. *MailServer throughput [MB/s] for native and 1VM environment.*

For all three case studies, we detected the best FS pairs for Mail-WL: X/E, E/X, E/E, B/E, B/X, among which B/E, E/X, E/E stand out. In the best FS pairs, Btrfs and Ext4 dominate as gFS, while Ext4 and then XFS dominate as hFS.

In addition, we detected the following bad FS pairs for Mail-WL: X/B, E/B, B/B, X/X, among which X/B stands out. In the worst FS pairs, XFS dominates as gFS, while Btrfs dominates as hFS.

When considering the best/worst FS pairs, for Mail-WL, B/E, E/X, E/E should be selected, and B/X, B/B should be avoided. Btrfs and Ext4 are good choices for gFS, and XFS should be avoided on the guest side. Btrfs should be avoided on the host side. Btrfs performs very well as gFS, and very bad as hFS, XFS is relatively bad as gFS, and Ext4 performs well in pairs on both sides.

For the RR component, the best are the B-Trees of XFS [C2.2, C2.3, C2.4, and C2.5] and Btrfs [C3.2, C3.3, C3.4, and C3.5]. However, for RW-synchronous cycles, XFS already has the well-known problem of low performance for random writes, whereas the Btrfs implements the CoW method. That is why Ext4 is the best selection for mail-WL. On the guest side, Btrfs and Ext4 perform quite well, while XFS generates an unfavorable WL-Gout sequence, which exhibits the worst behavior on the host side. In addition, on the host side, synchronous RW transfers pass through a large VMI file, so Ext4 performs as well as XFS, whereas Btrfs shows very bad performance.

Due to RW-synchronous accesses through a large image file, there is a growing influence of the extent_read [C1.5, C2.5, and C3.5] and extent_write operations [C1.5, C2.5, and C3.5] for the image file manipulation. XFS/Btrfs is at an advantage due to the B-Trees, and especially because of the use of the B-Trees for extents [C2.5 and C3.5], which are more efficient than the Ext4 extent-Trees [C1.5]. In addition, due to the dominant RW-synchronous, the influence of the writing method costs [C1.1, C2.1, and C3.1] is crucial, while due to the dominant RW, there is a growing negative impact of CoW and enhanced HK. Therefore, the dominant random write component makes Btrfs the worst hostOS file system [C3.1].

In FS-WL, there are 50 threads, each of which selects 5 of the 10,000 files in the directory-Tree, making the following sequences of operations: create-write-close, open-write-close, open-read-close, delete, and stat. In the case of fileserver-workload, all the throughput components are equally presented (random reads, sequential reads, random writes, sequential writes, create/delete/metadata operations). The impact of the WB cache mode is significant, as the sequential reads and writes are intensive. Writes are not synchronous, thus the impact of the cache can be exceptionally significant. The results for native and 1VM performance for fileserver WL are depicted in Fig. 4.

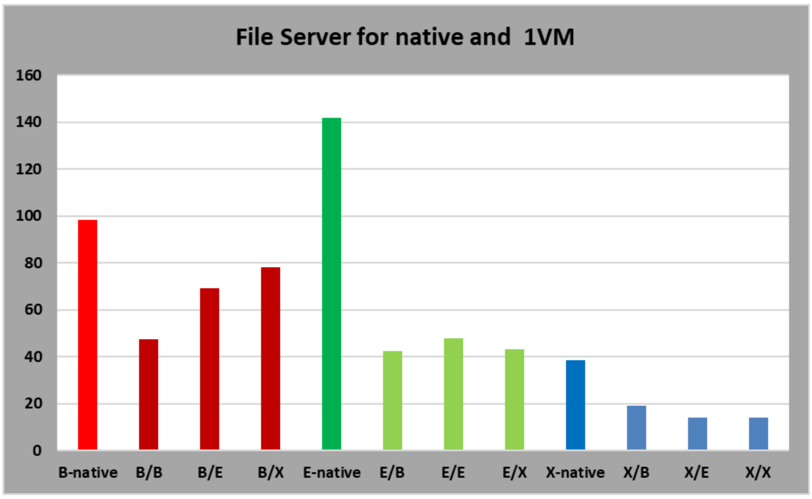


Figure 4. *FileServer throughput [MB/s] for native and 1VM environment.*

For all three case studies, we detected that the best FS pairs for FS-WL are B/X, B/E, E/X, B/B, E/B, among which B/X and B/E stand out. In the best FS pairs, Btrfs and then Ext4 dominate as gFS, while XFS slightly dominated as hFS, but all three FSs perform well. For all three case studies, we detected the following bad FS pairs for FS-WL: X/X, X/E, X/B. In the worst FS pairs, XFS dominates as gFS, while there is no dominance for hFS.

Considering the best/worst FS pairs for FS-WL, the best selection is either B/X or B/E, while it is recommendable to avoid XFS as a guest FS.

Each gFS (with its features) generates a unique WL-Gout, which passes through a large VMI file. For complex FS-WL the WL-Gout becomes also very complex. In FS-WL, WL-Gout with XFS performs the worst when processed through a large VMI file in hFS, whereas WL-Gout with Btrfs fits very well on all hosts hFS.

For sequences in which all the components are equally presented (random/sequential, reads/writes, file-data/metadata), the file block access components (Btrfs B-Trees [C3.5], XFS B-Trees [C2.5] and Ext4 extent-Trees [C1.5]) can have a significant impact on the performances. Also, due to a large number of files in a workload Tree, directory operations play a more important role (Btrfs B-Trees [C3.3], XFS B-Trees [C3.2] and Ext4 H-Trees [C3.1]). Due to the sequential SW components, the negative impact of CoW on Btrfs is reduced.

The experiment shows that WL-Gout sequences for Btrfs are optimal for all hFS, whereas XFS generates unfavorable sequences for all evaluated hFS. For Btrfs as gFS, when such optimal WL-Gout sequences are performed on a large VMI, all three hFS perform well. It is only necessary to ensure an optimal FS pairing on the guest/host side.

In the RFA-WL, five threads are created, where each thread selects five of the 10,000 files in the directory-Tree, making the following sequence of operations: open, open, open, append-close, read-read, close, close. In the case of the RFA-workload, random reads and writes are dominant for both data and metadata. Random writes are non-synchronous, so the impact of WB cache mode is significant. The housekeeping operations abound with writes, which is important for consideration. The results for native and 1VM performance for RFA WL are depicted in Fig. 5.

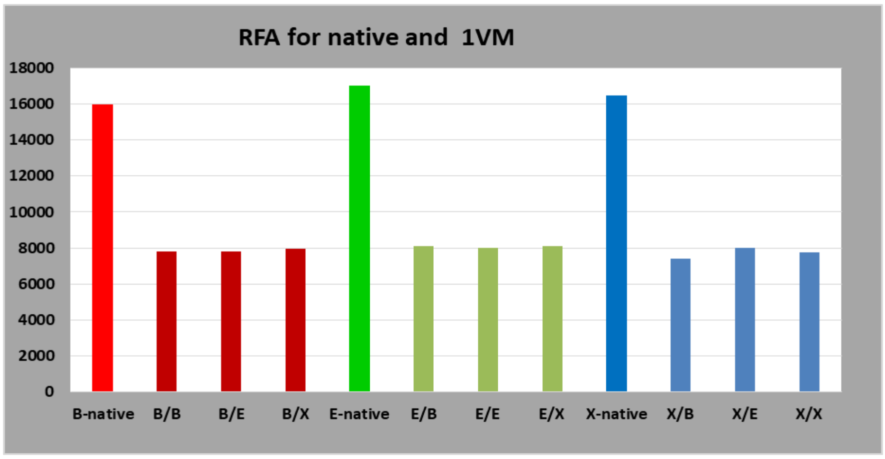


Figure 5. *RFA throughput [MB/s] for native and 1VM environment.*

For all three case studies, we have detected that the best FS pairs for RFA-WL are X/X, E/X, E/E, E/B, among which X/X, E/X stand out. In the set of the best FS pairs, Ext4 dominates as gFS, while XFS and Ext4 dominate for hFS. In addition, we detected inefficient FS pairs for the case of the RFA-WL: B/X, X/B, among which the pair B/X stands out. In the worst FS pairs, Btrfs dominates as gFS, while there is no dominance for hFS.

Considering the best/worst FS pairs for RFA-WL, the best selection is either X/X or E/X, whereas B/X should be avoided. Ext4 is very good as gFS, whereas FS pairs formed by both Btrfs and XFS should be avoided (B/X and X/B).

RFA-WL is dominated by RR and RW-asynchronous components. Due to RW-asynchronous components, WB caching has a large positive impact on the guest/host sides. B+Trees of XFS [C2.2, C2.3, C2.4, and C2.5] and B+Trees of Btrfs [C3.2, C3.3, C3.4, and C3.5] are the most promising candidates for RR components. Ext4 is very good as gFS, and it generates the most favorable WL-Gout sequences, which perform best for all hFS. XFS generates WL-Gout sequences that can well perform with XFS and Ext4 as hOS, whereas Btrfs makes the least favorable WL-Gout. Although the WB cache absorbs most RW accesses, some RW still passes to disk drivers, and this will have the most negative effect on Btrfs. In general, for RFA-WL, the rather unfavorable WL-Gout (Btrfs) sequences perform worst on XFS hFS. Due to the WL-Gout sequences of XFS, as well as CoW and HK of Btrfs as hFS, the combination of XFS on Btrfs is also unfavorable.

As the main results, we present the best and the worst FS pairs for individual workloads. The obtained results are given in Table 2, where the fundamental results are shown in columns 4 and 5. The column 2 contains the information on the best/worst guest FS detected in the corresponding best/worst FS pairs. The column 3 contains the information on the best/worst hFS detected in the corresponding best/worst FS pairs.

Table 2. *Performance Results Summarized.*

Workload	gOS FS best/ worst	hOS FS best/ worst	best pairs	worst pairs
Web	[E, X]/B	[E, X]/B	X/X, E/E, E/B,B/E, E/X, X/E	B/X, B/B, X/B
Mail	[B, E]/X	[E, X]/B	X/E, E/X, E/E, B/E,	X/B, E/B , B/B, X/X, B/X
File Server	B/X	X/no	B/X, B/E, E/X, B/B, E/B	X/X, X/E, X/B
RFA	E/B	[X, E]/no	X/X, E/X, E/E, E/B	B/X, X/B, X/E

The analysis of the results presented in Table 2, without taking into account the workloads characteristics, can be summarized as follows: there is neither the best nor the worst option for the guest or host. Their performance depends on workload characteristics (i.e., each FS can perform either as the best or as the worst depending on the workload), and they are very sensitive to the guest/host FS pairs.

Based on the reported three case studies and four WLs for Centos gOS, we provide the following recommendations. For Web WL, we recommend the Ext4/XFS as gFS and Ext4/ XFS as hFS, while Btrfs as gFS and Btrfs as hFS should be avoided. For Mail WL, we recommend Btrfs/Ext4 as gFS and Ext4 as hFS, while XFS as gFS and Btrfs as hFS should be avoided. For FileServer WL, we recommend Btrfs as gFS, while XFS as gFS should be avoided. For RFA WL, we recommend Ext4 as gFS, while Btrfs as gFS should be avoided.

For individual FS, we recommend Btrfs for Mail as gFS and for FileServer as gFS, while Btrfs should be avoided for Web as gFS/hFS, for Mail as hFS, and for RFA as gFS. In addition, we recommend XFS for Web as gFS and for Web as hFS, while XFS should be avoided for Mail as gFS and for Fileserver as gFS. Finally, we recommend Ext4 for Web as gFS, for Web as hFS, for Mail as gFS, for Mail as hFS and for RFA as gFS. There is no recommendation for avoiding Ext4.

Regarding solid-state drives, their examination will follow this research in the field on hypervisors case studies. Although the authors conducted some research on this topic, the results are not yet mathematically explainable. Certain studies regarding Microsoft Windows environment are available in [32]. More on this topic will be discussed in the research that follows.

Regarding real environments, not virtual ones, a reader may conclude that virtualization takes a toll on performance, i.e., dropping I/O throughput. However, typical small-to-medium-sized service providers would apply virtual environments, such as providing a machine with sufficient processing power and storage devices to reduce electricity consumption, providing rack spaces for servers, etc.

8. CONCLUSION

Virtualization techniques have a significant impact on operating system performance, and in this paper, we examine this impact on specific components, such as different FS pairs. When considering a hypervisor virtualization environment based on different filesystems on the host and guest sides, a number of FS pairs could be included in the analysis. We analyzed the behavior of three 64-bit Linux FS, both on the guest and on the host side, resulting in nine pairs for examination. We introduced the model for FS pairs and validated it for the KVM hypervisor, but our model is also applicable for most Linux-based hypervisors, such as Xen, ESXi, and Proxmox.

Our KVM-based case studies showed that FS pairing is quite sensitive and that there is no optimal FS pair. The choice of a suboptimal FS pair depends on WL, due to a complex VE with a large number of input variables, complex structures in FS itself, and the complex FS pair interaction. One of the important conclusions is that the FS performance in a VE can be interpreted by analyzing the FS pair as a whole instead of an individual FS analysis.

The reported results are quite consistent with the best practices obtained in experiments related to environments with and without virtualization. The results also indicate that Btrfs is not a good solution if the workload is generating big amounts of random writes. Ext4 is still good enough when compared to XFS and Btrfs, although it uses relatively simple technologies and does not have a sophisticated B+ Tree structure. Ext4 shows excellent performance.

Our model and KVM results show that optimal pairing of FS types on the host and guest sides may be rather challenging. Thus, an additional contribution of this paper is that it provides insight into the performance of selected FS pairs for typical application WL, based on three case studies in a KVM hypervisor-based VE.

We think that the optimal pairing of FS types on the host and guest sides is particularly complex. Administrators of VE face a complex problem in determining the optimal VE for their applications of interest. First, for each type-1 hypervisor (except Hyper-V) that will be included in VE, a pool of hOS FS with different types should be created, on which they can place VMs and migrate if necessary. Secondly, administrators should determine the optimal FS pair for applications of interest. Our model can be used for hypotheses about expected behavior, whereas good benchmark or real-app testing can provide real validation. For VMs with gOS, possible application-based adjustment of gOS FS types should also be provided.

Beside the model, another contribution of this paper is the beginning of Knowledge Data Base (KDB) creation that is related to FS performance of FS pairs in hypervisor-based VE. We consider the KDB to be another significant theoretical contribution of this paper. Based on three case studies in KVM VE, we get the knowledge about optimal and very bad FS pairs for typical application WL. For now, KDB includes three case studies for KVM VE, with 9 FS pairs, Centos 7 as gOS, Filebench as WL generator. KDB is open for extension with new case studies, which include another FS pairs (with NTFS, F2FS, JFS), other hypervisors (ESXi, Xen, Proxmox, Hyper-V), other hOS and gOS, other benchmarks or real applications, and new versions of all of these components. We consider that our KDB can serve administrators to create VE for specific system case.

Future work on optimal FS pairing in VE may include the introduction of new FS types in the analysis, such as NTFS, F2FS, and JFS on the guest/host side, and evaluation of the case studies based on other hypervisors, such as ESXi, Xen, Proxmox, and Hyper-V. As a part of the future work, we may consider the application of artificial intelligence as the possibility for intelligent management of FS pairs, assuming that large hOS storage pools based on different FS (Btrfs, Ext4, XFS, etc.) exist. The proposed system would be trained with information about the best/worst FS pairs supplied from KDB. Based on the detected application WLs on VMs and training data, the system should migrate guest VMs to another hOS FS pool, and realize optimal FS pairs.

FUNDING:

This research received no external funding.

INSTITUTIONAL REVIEW BOARD STATEMENT:

Not applicable.

INFORMED CONSENT STATEMENT:

Not applicable.

CONFLICTS OF INTEREST:

The authors declare no conflict of interest.

REFERENCES

[1] R. Y. Ameen, A. Y. Hamo, "Survey of server virtualization," *International Journal of Computer Science and Information Security (IJCSIS)*, Vol. 11, No. 3, 2013. arXiv preprint arXiv:1304.3557

[2] E. Correia, "Hypervisor-based server virtualization," In *Encyclopedia of Information, Science and Technology*, 3rd Edition, IGI Global, pp. 1182–1187, 2015. DOI 10.4018/978-1-4666-5888-2.ch112

[3]A. Varasteh, M. Goudarz, "Server consolidation techniques in virtualized data centers," *IEEE System Journal*, Vol. 2, No. 11, pp. 772–783, 2017. DOI 10.1109/JSYST.2015.2458273

[4] T. Imada, M. Sato, and R. Kimura, "Power and QoS Performance Characteristics of Virtualized Servers," In Proceeding of the 10th IEEE/ACM International Conference on Grid Computing (GRID), 2009, pp. 232–240. DOI 10.1109/GRID.2009.5353054.

[5] J. Liu, Y. Zhang, Y. Zhou, D. Zhang, and H. Liu, "Aggressive Resource Provisioning for Ensuring QoS in Virtualized Environments," *IEEE Transactions on Cloud Computing*, Vol. 3, No. 2, 2015, pp. 119–131.

[6] J. W. Lin, C. H. Chen, and C. Y. Lin, "Integrating QoS awareness with virtualization in cloud computing systems for delay-sensitive applications," *Future Generation Computer Systems*, Vol. 37, pp. 478–487, 2014. DOI https://doi.org/10.1016/j.future.2013.12.034.

[7] C. D. Graziano, "A performance analysis of Xen and KVM hypervisors for hosting the Xen worlds project," M. A. thesis, Iowa State University, Ames, IA, 2011. DOI 10.31274/etd-180810-2322

[8] S. Pawar and S. Singh, "Performance comparison of VMWare and Xen hypervisor on guest OS," *IJICSE*, Vol. 2, No. 3, pp. 56–60, 2015. ISSN 2393-8528. https://ijicse.in/index.php/ijicse/article/view/43/41

[9] A. Kumar and S. Shiwani, "Guest operating system based performance comparison of VMWare & Xen hypervisor," *International Journal of Science, Engineering and Technology*, Vol. 2, No. 5, pp. 286–297, 2014. ISSN 2348-4098. Available: http://ijset.in/wp-content/uploads/2014/06/IJSET.0620140075.1011.1906_Ankit_Kumar_286-2971.pdf

[10] A. Bhatia and G. Bhattal, "A comparative study of various hypervisors performance," *International Journal of Scientific and Engineering Research*, Vol. 7, No. 12, pp. 65–71, 2016.

[11] V. P. Singh, "Analysis of system performance using VMWare ESXi server virtual machines," M. Sc. thesis, Thapar University, Patiala, India, 2012. Available: http://hdl.handle.net/10266/1809

[12] H. Kazan, L. Perneel, and M. Timmermann, "Benchmarking the performance of Microsoft Hyper-V server, VMWare ESXi and Xen hypervisors," *Journal of Emerging Trends in Computing and Information Sciences*, Vol. 4, No. 12, pp. 922–933, 2013. ISSN 2079-8407

[13] M. Polenov, V. Guzik, and V. Lukyanov, "Hypervisors comparison and their performance," In Computer Science On-line Conference, 2018, pp. 148–157. DOI 10.1007/978-3-319-91186-1_16

[14] P. Kedia, R. Nagpal, "Performance evaluation of virtual environment with respect to physical environment," *International Journal of Computer Applications (0975 – 8887),* Vol. 89, No. 11, pp. 17–22, 2014. DOI 10.5120/15676-4425

[15] P. Vijaya V. Reddy, L. Rajamani, "Evaluation of different hypervisors performance in the private cloud with SIGAR framework," *International Journal of Advanced Computer Science and Applications (IJACSA)*, Vol. 5, No. 2, pp. 60–66, 2014. DOI 10.14569/IJACSA.2014.050210

[16] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," In 2015 IEEE International Conference on Cloud Engineering, 2015, pp. 386–393.

[17] J. Hwang, S. Zeng, F. Wu, and T. Wood, "A component-based performance comparison of four hypervisors," In 13th IFIP/IEEE Int. Symposium on Integrated Network Management (IM) Technical Session, 2013, pp. 269–276. ISBN 978-3-901882-50-0 978-1-4673-5229-1, 978-3-901882-51-7

[18] A. Elsayed and N. Abdelbaki, "Performance evaluation and comparison of the top market virtualization hypervisors," IEEE International Conference on Computer Engineering and Systems, 2013, pp. 45–50. DOI 10.1109/ICCES.2013.6707169

[19] W. Graniszewski and A. Arciszewski, "Performance analysis of selected hypervisors (Virtual Machine Monitors-VMMs)," *International Journal of Electronics and Telecommunications*, Vol. 62, No. 3, pp. 231–236, 2016. DOI 10.1515/eletel-2016-0031

[20] S. A. Algarni, M. R. Ikbal, R. Alroobaea, A. S. Ghiduk, and F. Nadeem, "Performance evaluation of Xen, KVM, and Proxmox hypervisors", *International Journal of Open Source Software and Processes*, Vol. 9, No. 2, pp. 39–54, 2018. DOI 10.4018/IJOSSP.2018040103

[21] B. Djordjevic, N. Macek, and V. Timcenko, "Performance issues in cloud computing: KVM hypervisor's cache modes evaluation," *Acta Polytechnica Hungarica*, Vol. 12, No. 4, pp. 147–165, 2015. DOI 10.12700/APH.12.4.2015.4.9

[22] V. K. Manik and D. Arora, "Performance comparison of commercial VMM: ESXi, XEN, HYPER-V & KVM," In 3rd International Conference on Computing for Sustainable Global Development, 2016. ISBN Electronic ISBN 978-9-3805-4421-2, DVD ISBN 978-9-3805-4420-5, Print on Demand (PoD) ISBN 978-1-4673-9417-8

[23] D. Vujičić, D. Marković, B. Đorđević, and S. Randić, "Benchmarking Performance of Ext4, XFS, and Btrfs as Guest File Systems under Linux Environment," In Proceedings of 3rd International Conference on Electrical, Electronic and Computing Engineering IcETRAN 2016, Zlatibor, Serbia, June 13–16, 2016, pp. RTI1.3.1-5.

[24] K. V. Kumar, A. M. Cao, J. R. Santos, and A. Dilger, "Ext4 block and inode allocator improvements," In Proceedings of the Linux Symposium, Vol. 1, 2008, pp. 263–273.

[25] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Thomas, and L. Vivier, "The new Ext4 filesystem: current status and future plans," In Proceedings of the Linux Symposium, Vol. 2, 2007, pp. 21–33.

[26] M. Holton and R. Das, "XFS: a next generation journalled 64-Bit filesystem with guaranteed rate I/O," SGI Corp, Internet White Paper, 1995.

[27] O. Rodeh, "B-Trees, shadowing, and clones," *ACM Transactions on Storage (TOS)*, Vol. 3, No. 4, Article No. 2, pp. 1–27, 2008.

[28] O. Rodeh, "Deferred Reference Counters for Copy-On-Write B-Trees," IBM Corporation, Technical Report rj10464, 2010.

[29] O. Rodeh, J. Bacik, and C. Mason, "BTRFS: The Linux B-Tree filesystem," *ACM Transactions on Storage (TOS),* Vol. 9, No. 3, Article No. 9, pp. 1–32, 2013.

[30] H. Powell, "ZFS and Btrfs: a quick introduction to modern filesystems," *Linux Journal*, Vol. 2012, No. 218, Article No.: 5, 2012.

[31] Silicon Graphics Inc. XFS Filesystem Structure, Documentation of the XFS filesystem on-disk structures, 2006.

[32] B. Đorđević, V. Timčenko, and N. Maček, "NTFS fajl sistem u MS Windows i Linux okruženju," *Zbornik radova XIII međunarodnog naučno-stručnog Simpozijuma INFOTEH 2014*, 2014, pp. 805–808. ISBN 978-99955-763-3-2

Journal of
COMPUTER AND
FORENSIC SCIENCES

Instructions and Information for Authors

Thank you for considering to submit a manuscript to the Journal of Computer and Forensic Sciences. The points below provide general instructions and information for authors. If you have any questions, please contact us at **comput.forensic.sci@kpu.edu.rs**.

Submission Checklist

- Ensure that your manuscript fits the **Aims and Scope** of the Journal.
- Use the **Microsoft Word template** or the **LibreOffice template** to prepare your manuscript.
- Ensure that your manuscript *complies with our* **research** *and* **publishing ethics** guidelines.
- Ensure that all authors have signed the **Author Statement**.

Aims and Scope

Journal of Computer and Forensic Sciences covers advanced and innovative research across the fields of computer and forensic sciences. More information about the Aims and Scope is available **here**.

Open Access

The Journal of Computer and Forensic Sciences is an open access, peer-reviewed scientific journal. All accepted manuscripts are made freely and permanently available online immediately upon publication, without subscription charges.

No Article Processing Charges (APC)

The journal does not have *submission charges or article processing charges.*

Manuscript Types

The journal publishes:

- Original research papers – recent research results in computer and forensic sciences,
- Review articles (solicited reviews) – comprehensive and up-to-date systematic review of a specific area,
- Case reports – describing interesting and exceptional cases, providing new information to the readership.

Research Ethics

Manuscripts reporting on research involving human subjects, animals, cell lines, and plants will be scrutinized by the editorial office. Editors may ask the authors for documentary evidence or reject any submission that does not meet the research ethics requirements.

Please note the following research ethics guidelines:

- Research involving human subjects must be carried out following the rules of **the Declaration of Helsinki**[1] of 1975, revised in 2013.
- For research on animals, authors should ensure that their research complies with the principles of the 3Rs (i.e., **Replacement, Reduction and Refinement**[2]) which provide a framework for ethical decision making in the use of animals in research and teaching.
- For research involving cell lines, the origin of any cell line should be stated.

Publication Ethics

We adhere to the Core Practices and Guidelines of the **Committee on Publication Ethics**[3], and expect authors to comply with its best ethical publication practices. Plagiarism, data fabrication, and image manipulation are not acceptable. If evidence of misconduct is found, appropriate action will be taken to correct or retract the publication.

Manuscript Submission

The authors may submit a manuscript that has not been published before, that is not under consideration for publication elsewhere, and that has been approved by all co-authors. All manuscripts should be submitted through **the online submission system.**

1 https://www.wma.net/what-we-do/medical-ethics/declaration-of-helsinki
2 https://www.animalethics.org.au/three-rs
3 https://publicationethics.org/

Templates

Please use the **Microsoft Word template** or the **LibreOffice template** to prepare your manuscript.

Language

All manuscripts should be written in English. Please note the following:

- Our reviewers are advised to distinguish between the quality of writing and the quality of ideas. However, authors are strongly encouraged to carefully edit and proofread their manuscripts.
- All accepted manuscripts undergo professional English editing (free of charge), and proofreading by the authors.
- Authors are also strongly urged to avoid using language or examples that may be perceived as discriminatory.

Manuscript Length

Different types of manuscripts require more or less space. Therefore, we imply no restrictions on the length of manuscripts, provided that the text is concise and comprehensive.

Manuscript Structure

All manuscripts should consist of three main parts: the front matter, the main body, and the back matter.

The front matter should include:

- **Title**: The manuscript title should be specific and relevant.
- **Author list**, **affiliations,** and **email addresses**: At least one author should be named as the corresponding author.
- **Abstract**: The a*bstract should be a single paragraph and must not exceed* 200 words. It should express the purpose of the study, indicate the main methods applied, and summarize the main findings.
- **Keywords**: Three to five specific keywords should be added.

The main body structure depends on the type of manuscript.

- In original research articles, it should include the following sections: **Introduction**, **Materials and Methods**, **Results**, **Discussion**, and **Conclusions** (authors can make appropriate minor modifications to this section structure).
- In review articles, it should consist of literature review sections.
- In case studies, it should consist of sections describing and discussing the case study.

The back matter should include the following sections:

- **Funding: Authors should disclose a**ll sources of funding for their research. For research that did not receive external funding, please add "This research received no external funding".
- **Acknowledgments (optional):** The authors may acknowledge any support that contributed to their manuscript, which is not included in the funding section.
- **Author Contributions (optional):** For manuscripts with several authors, their individual contributions can be specified.
- **Institutional Review Board Statement: For studies involving humans or animals, p**lease add "The study was conducted according to the guidelines of the Declaration of Helsinki" and a**dd the Institutional Review Board Statement and approval number. If ethical review and approval were waived, the authors are required to provide a detailed justification. For studies not involving humans or animals, please add "Not applicable".**
- **Informed Consent Statement: For studies involving humans, p**lease add "Informed consent was obtained from all subjects involved in the study"**. If informed consent was waived, the authors are required to provide a detailed justification. For studies not involving humans, please add "Not applicable".**
- **Conflicts of Interest:** Authors must disclose all conflicts of interest that may directly or potentially influence or impart bias on the work. Examples of potential conflicts of interest include but are not limited to: research grants, honoraria, financial support, employment, consultancies, affiliations, intellectual property rights, financial relationships, personal or professional relationships, and personal beliefs. If there is no conflict of interest, please add "The authors declare no conflict of interest."
- **References:** The Reference section must provide a numbered list of references, as recommended by the **IEEE Citation Guidelines**[4]. The list is comprised of the sequential enumerated citations. A number enclosed in square brackets, placed in the text of the report, indicates the specific reference. Citations are numbered in the order in which they appear.

Figures, Tables, and Equations

- All figures and tables should be inserted into the main body of the manuscript (preferably close to their first citation) and numbered following their number of appearance.
- All figures and tables should have an explanatory caption.
- All figures should be at a sufficiently high resolution (i.e., 300 dpi or higher) and provided in a single zip archive. Preferable formats are TIFF, JPEG, and EPS.
- All equations should be numbered following their number of appearance.
- All equations should be editable by the editorial office (i.e., not provided in a picture format).

4 https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf

Citation Policy

If a manuscript includes material (e.g., figures, tables, text passages, etc.) taken from other sources, its source must be clearly cited. When appropriate, the authors should obtain permission from the copyright owner(s) and include evidence that such permission has been granted when submitting their manuscripts.

References cited in the text must appear in the References list, and vice versa. Personal communications and classical works are cited in text only and are not included in the References list.

Authors should not engage in citation manipulation, including but not limiting to excessive self-citation and "honorary" citations. Authors should not cite advertisements or advertorial material.

Editorial Procedures and Peer-Review

- **Initial checks:** All submitted manuscripts are first checked whether they fit the aims and scope of the Journal and meet its standards. At this stage, your manuscript may be rejected before peer-review or returned to the authors for revision and resubmission.
- **Peer review:** Once a manuscript passes the initial checks, it is assigned to at least two independent experts for peer-review. A double-blind review is applied. The guidelines for reviewers are available **here**.

Editorial decision and revision: The decision on a manuscript is one of the following:

- Accept in present form,
- Minor revision,
- Major revision,
- Reject.

Author appeals: In a response to the reviewers, the authors should address all reviewers' comments. The response should be organized by presenting reviewers' comments one by one, followed by the authors' response. Authors may appeal a rejection by sending an e-mail including a detailed justification to the Editorial Office.

Guidelines for Reviewers

Thank you for considering reviewing a manuscript for the Journal of Computer and Forensic Sciences. We rely upon the knowledge and commitment of our peer reviewers to ensure the academic integrity of our Journal. The points below provide general reviewing guidelines. If you have any questions, please contact us at **comput.forensic.sci@kpu.edu.rs**.

Before Reviewing

Before you accept our invitation to review a manuscript, please consider the following:

- **Timeliness:** Please try to submit your reviews on time. If you cannot meet a given deadline, please let the editor know.
- **Reviewer qualifications:** You have been invited to review the manuscript because the editor believes that your expertise covers the topic of the manuscript. However, if the manuscript is outside your expertise, you should decline to review it. If the manuscript is *generally within your expertise but you do not feel confident assessing certain parts of it, please notify the editor.*
- **Conflicts of interest:** You should disclose potential conflicts of interest. If you recognize the author's work, have a financial or commercial conflict of interest related to the reported results, or have strong feelings about a controversial question considered in the manuscript, you should disqualify yourself. If you are unsure whether you have a conflict of interest, discuss your concerns with the editor.
- **Confidentiality:** You should keep the content of the manuscript confidential. If you want to involve your students or postdocs in your review, you must obtain permission from the editor. If permitted, your assistants must be informed of the confidentiality requirement.
- **Anonymity:** Our journal operates double-blind peer review, which means that the reviewers and authors are unaware of each other's identities. You must not reveal your identity to the authors (e.g., in your comments, in metadata of submitted files, etc.).
- **Interactions:** There is no open interaction between reviewers and no public commenting during formal peer review. After you have submitted your report, you will have access to other reviewers' reports. We will also inform you on the final editorial decision of the paper. The reviewer reports, the author responses to reviews, and the editor decision letters are not published.
- **Language:** All review reports must be written in English.
- **Reviewer acknowledgment:** Once a year, we recognize our reviewers with annual listings in the journal. If you do not wish to have your name included in this list, please let us know.
- **Ethical guidelines:** Please note that all reviewers for our Journal are expected to follow **the COPE Ethical Guidelines for Peer Reviewers**[5].

5 https://publicationethics.org/

Evaluating Manuscripts

Regarding your comments for authors

Start your review report by writing a paragraph or two in which you summarize the manuscript, emphasize its main contributions, and list its strengths and weaknesses. Then continue with the assessment of the individual sections of the manuscript. You should consider questions such as:

- Is the manuscript relevant for the field and suitable for the Journal?
- Is the research question original and well-defined?
- Is the manuscript clear and well-structured? Does it contain all of the sections you would expect?
- Are the cited references relevant and complete?
- Is the methodology clearly explained? Are the methods appropriately selected?
- Are the data underlying the research representative and balanced?
- Is the manuscript scientifically and technically sound? Is the experimental design appropriate? Are the reported results reproducible?
- Are the results analyzed and interpreted correctly? Are the conclusions supported by evidence? Is the manuscript statistically sound?
- Does the theory fit the data?
- Are the figures, tables, source codes, etc. appropriate?
- Is the manuscript of interest to the scientific community and the Journal's audience?
- Do you think that the reported results may advance the field?
- Is the English language of sufficient quality?

When preparing your review report, you should:

- **Ensure that your identity is not disclosed;**
- Be objective and constructive;
- Be detailed; your feedback should help the authors improve their manuscript;
- **Number each comment;**
- **Cite page numbers when referring to specific parts of the manuscript;**
- Scrutinize the manuscript, not the authors; avoid any derogatory personal comments or unfounded accusations;
- Make sure to distinguish between the quality of writing and the quality of ideas, especially for authors whose first language may not be English;
- Immediately report any suspected breaches of ethics, including scientific misconduct, fraud, and plagiarism.

Regarding your comments for editors

Your comments to the editors will not be revealed to the authors or other reviewers. These comments are optional. However, if provided, they should be consistent with your comments to the authors.

Regarding your final recommendation

To make a final recommendation on a manuscript, please choose one of the following options:

- **Accept in present form:** The manuscript fulfills all of the requirements described above, although some small fixes may be required (e.g., typos or grammatical corrections, etc.). No additional action by the review is required.
- **Minor revision:** The manuscript requires a small number of easily correctable errors or minor content correction or clarification. The article is in principle accepted after revision based on the reviewer's comments.
- **Major revision:** The manuscript offers relevance or value but contains significant deficiencies and requires a major rework. The acceptance of the manuscript depends on the revisions.
- **Reject:** The manuscript has serious flaws or does not offer relevance or value.

Your final recommendation should match your comments for the authors. Please note that the final recommendation will be visible to editors and other reviewers, but not to the authors.

Acknowledgment to Reviewers in 2022

The editorial team of the Journal of Computer and Forensic Sciences wishes to thank the following reviewers, who in 2022 have performed an essential role in ensuring the academic integrity of this publication:

- Prof. Anton Kos, PhD, University of Ljubljana, Faculty of Electrical Engineering, Slovenia;
- Prof. Zlatogor Minchev, PhD, Joint Training Simulation and Analysis Center, Institute of ICT, Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Bulgaria;
- Prof. Mitko Bogdanoski, PhD, Military Academy General Mihailo Apostolski, Skopje, NR Macedonia;
- Prof. Ivan Tot, PhD, University of Defence, Military Academy, Belgrade, Serbia;
- Miroslav Lutovac, PhD, Academy of Engineering Sciences of Serbia;
- Dragiša Žunić, PhD, Institute for Artificial Intelligence Research and Development of Serbia;
- Prof. Dušan Joksimović, PhD, University of Criminal Investigation and Police Studies, Belgrade, Serbia; and
- Prof. Petar Čisar, PhD, University of Criminal Investigation and Police Studies, Belgrade, Serbia.

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

343.9

JOURNAL of Computer and Forensic Sciences / editor in chief
Milan Gnjatović. - [Štampano izd.]. - Vol. 1, no. 1 (2022)-   . - Belgrade :
University of Criminal Investigation and Police Studies, 2022-
(Beograd : Službeni glasnik). - 30 cm

Polugodišnje. - Drugo izdanje na drugom medijumu:
Journal of Computer and Forensic Sciences (Online) = ISSN 2956-0799
ISSN 2956-087X = Journal of Computer and Forensic Sciences (Štampano izd.)
COBISS.SR-ID 85369353