PartFlow: A Visualization Tool for Application Partitioning and Workload Offloading in Mobile Edge Computing

Minghao Li
The Chinese University of Hong Kong, Shenzhen
Shenzhen, Guangdong, China
minghaoli1@link.cuhk.edu.cn

Hao He
The Chinese University of Hong Kong, Shenzhen
Shenzhen, Guangdong, China
haohe@link.cuhk.edu.cn

Jian Zhao
University of Waterloo
Waterloo, Ontario, Canada
jianzhao@uwaterloo.ca

Wei Cai
The Chinese University of Hong Kong, Shenzhen
Shenzhen, Guangdong, China
caiwei@cuhk.edu.cn

ABSTRACT

In mobile edge computing (MEC), one of the optimization strategies for mobile applications is to offload heavy computing tasks to cloud and edge servers. Besides, modeling individual methods through static code profilers, the exploiting of dynamic user-driven execution patterns is also indispensable during the process of constructing partitioning algorithms. This paper presents PartFlow, an interactive visualization system, that supports the analysis of components in mobile applications from multiple aspects and effectively helps researchers make partitioning and offloading algorithms. By the binary instrument of mobile applications[14], PartFlow collects application component data from user devices remotely. We design several interactive diagrams to evaluate the component performances and indicate the transition patterns based on the gathered data. Further, to gain a higher accuracy of multi-step forecasting of component state, PartFlow deploys a deep learning-based method that effectively facilitates user experience.

CCS CONCEPTS

• **Networks** → Mobile networks; • **Human-centered computing** → **User interface programming**; **Information visualization**.

KEYWORDS

Visualization, Decision Making, Mobile Edge Computing, Workload Offloading,

ACM Reference Format:
Minghao Li, Hao He, Jian Zhao, and Wei Cai. 2023. PartFlow: A Visualization Tool for Application Partitioning and Workload Offloading in Mobile Edge Computing. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In recent years, mobile applications have started becoming abundant in various categories such as entertainment, health, games, business, social networking, travel, and news [6]. Through mobile cloud computing (MCC), mobile users can easily have the access to rich computational resources. As a solution for the delay issue in MCC, mobile edge computing (MEC) can utilize hardware facilities at the edge of the network [3]. Since the resources at the disposal of edge nodes themselves are not as powerful as those of the cloud, instead of migrating the entire mobile application, it is required to offload some of the heavy workloads to edge servers.

Most existing schemes tend to retrieve execution paths among different application methods, which firstly utilized static code analyzers to deconstruct the mobile application into a set of methods at a fine-grained level basically recognized as ***components*** [4]. Then, based on the method's static code, the partitioning algorithm is made by modeling the application into a graph, where vertices and edges represent components and their call relation. Finally, the offloading decision is determined through either empirical heuristics from researchers or deterministic optimization objectives.

The optimization measure of partitioning the application into a bunch of computational components and remotely executing heavier ones seems to alleviate the pressure on edge servers, whereas frequent switches between the local and remote method executions cause massive energy consumption and long response time [15]. Therefore, application execution patterns under user operation play crucial roles in the high-performance offloading system. For the purpose of promoting the system's efficiency, reducing unnecessary transmission of component states between end-users and edge servers is needed to be considered in application partitioning and workload offloading.

Some approaches do take notice of the significance of exploiting the dynamic patterns of applications' run-time execution for workload offloading [17]. Unfortunately, these strategies of dynamic partitioning face four challenges. Firstly, obtaining execution patterns and human factors in the modeling process and other requisite data for partitioning algorithms is complicated for researchers to utilize different tools and even environments. Secondly, there are no existing profiling and management tools for researchers to collect massive and comprehensive component execution data from various applications running on heterogeneous environments for experimental purposes. Thirdly, the current mainstream tools focus

on static code analysis, which neglects the demand of inspecting stochastic characteristics of components driven by users. Mobile computing researchers have been plagued by the intricacy of relationships between different components for a long time. Fourthly, it has become increasingly tough to select several methods from hundreds and thousands of components for partitioning and offloading. Researchers face the obstacle in selecting and arranging the objective sequence of components.

To address these challenges, in this paper, we present an interactive visualization system for researchers who require analytical assistance in discovering solutions focus on user-driven execution patterns for application partitioning and workload offloading. Our primary contribution is the design and implementation of PartFlow, a visual analytic system providing essential data for researchers to choose objective sequences of components for dynamic fine-grained analysis. Based on the gathered data from [14], we employ comprehensive analytical methods to estimate component performance and visualize the statistical results based on flame graph [9] and sunburst graph [1]. We further design an interactive digraph to reveal transition patterns of component states from various aspects. Supported by the previous visualization, PartFlow enables the user to arrange a simulation series of component states by a Sankey-based interactive decision tree [12] for the simulation of the actual component series called by the user. Given the inadequate capability of handling missing data or unobserved states, we deploy a DL-based forecasting model to provide a better user experience.

2 RELATED WORK

In this section, we review related work on visual analytics techniques and designs for profiler output, methods relationships, and uncertainties in the event sequence.

Visualization for Profiler Output. People often choose some professional performance analysis tools for statistical prediction methods when testing application performance, such as Android Profiler, Solaris Studio, Java Flight Recorder (JFR). These tools can gather information like application stack information, real-time device status, total power consumption, etc. Such information is usually large and complex, and it is often difficult for users to observe the desired data features directly. So, to address this challenge, JFR, a JVM-specific analysis tool, uses line charts and pie charts to show CPU usage and stack. Solaris Studio and Android Profiler have introduced the Flame Graph [9], which visualizes how long threads run and how hierarchical they are. In addition, there are some performance analysis visualization tools such as Frequency Trails [11], Latency Heatmap [10]. However, such tools are generally less interactive and cannot target a single component. PartFlow enhances the user experience of viewing a single component. It can also be used together with the subsequent visual panel to enable users to obtain the analysis data of component performance more quickly.

Visualization for Methods Relationship. In the research of dynamic partitioning, it is necessary to analyze not only a single component but also the component before or after the execution of the component and the related components with call relationships. The current solution to this problem is to visualize the method call graph. For instance, FlowDroid [1] and TaintDroid[5], visualize a digraph with massive nodes based on Gephi [2]. However, Flowdroid based on Soot[3] is mainly based on static bytecode call relationship without considering the influence of user behavior and device performance on component state timing when using the Android program. The lower-level, more dynamic TaintDroid[5] is the variable-level tracing of untrusted app code through the virtual machine (VM) interpreter. Such tools focus more on static code analysis and cannot dynamically analyze the sequence set of component states in combination with the actual operation of applications by users. PartFlow is different from the above tools in that PartFlow targets the actual user-driven component state and reflects their state transition relationship according to the stochastic patterns.

Visualization of Uncertainty and Event Sequence. Application partitioning is for the entire sequence of components, which means adjacent components offload together on the same or several adjacent edges. At this point, it is necessary to visualize the event sequence. Many visualizations are capable of analyzing how different sequences of events lead to different outcomes, which can help analysts generate hypotheses about causation. For example, DecisionFlow [8], OutFlow [18] and MatrixWave [19] aggregate similar event sequences into progression pathways and visually encode the correlations between the pathways and possible outcomes. Moreover, in the process of selecting the sequence of components, the state transfer of components often depends not only on the internal logic of the application but also on the application logic of the program and the operation behavior of users. Therefore, reasonable presentation of uncertainty in dynamic partitioning can improve users' understanding of data and the quality of their decisions. According to extensive surveys [2], techniques of one of the categories incorporate uncertainty information by adding extra visual components in the form of glyphs [13], and studies focusing on visualizing both event sequence and uncertainty[12] enhances users' confidence in decision-making, but it does not show the occurrence probability of the whole queue. However, these works are designed for the state of each node without considering the occurrence probability of the whole queue. PartFlow allows users to select component queues with a relatively high probability of occurrence and develop corresponding dynamic partitioning strategies according to the selected component queues.

3 DATASET

In this section, we introduce the brief process of data acquisition and the description of the prerequisite dataset, which was previously collected by [14].

3.1 Data Acquisition

Our previous work [14], a blockchain-based profiling system, has already collected behavioral data from several invited subjects. The profiling system is an external software to PartFlow, which is supposed to run in the subject's local environment.

The system consists of the blockchain module, which provides secure and private communication, and the profiling module, which

[1] https://github.com/secure-software-engineering/FlowDroid
[2] https://gephi.org/
[3] https://github.com/soot-oss/soot

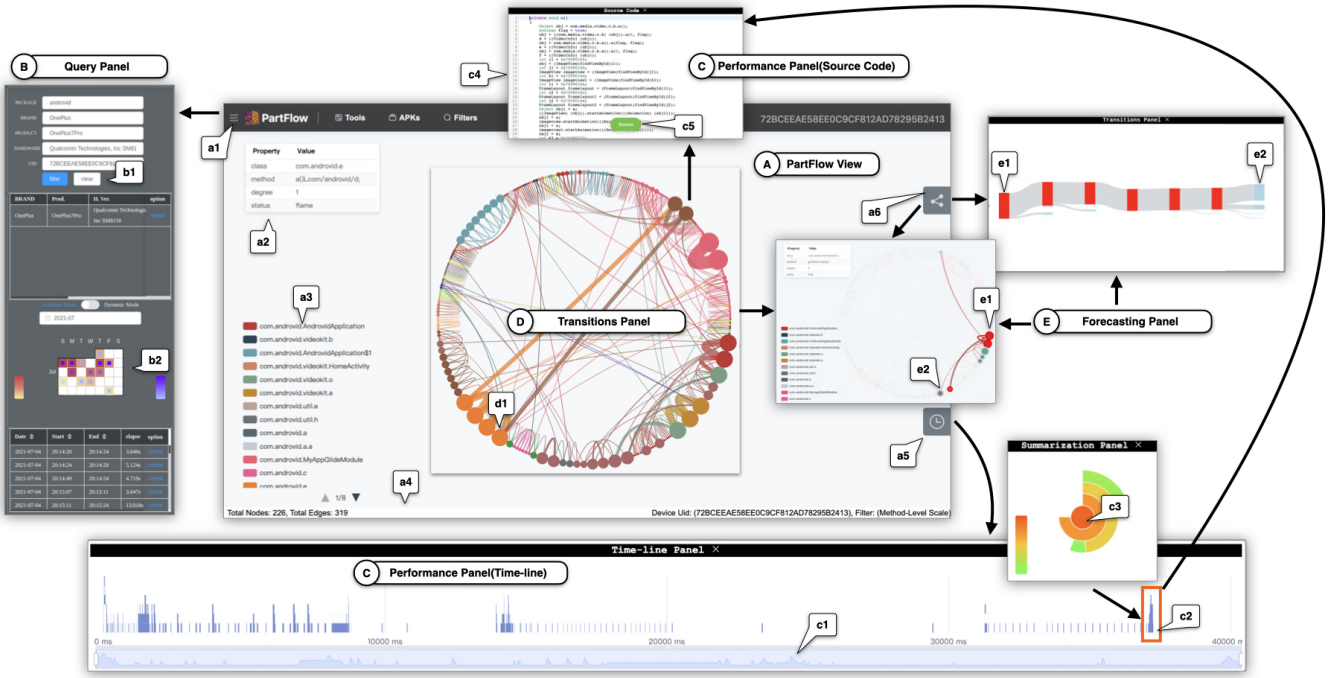

Figure 1: The PartFlow interface. (A) The initial PartFlow interface. (B) The Query Panel for filtering user activities. (C) The Performance Panel for evaluating the performance of components. (D) The Transition Panel for indicating component transitions is based on the stochastic matrix. (E) The Forecasting Panel for selecting the high-probability sequence based on interactive Sankey diagram.

Table 1: Specifications Table

Subject	Computer Networks and Communication
Specific subject area	Mobile Edge Computing, Partition Algorithm
Type of data	Method-level component state log data in JSON
Data format	Raw
Parameters for data collection	The parameters, state and timestamps of components in Android apps
Data accessibility	Direct URL to data: https://github.com/liminghao0914/user-component-dataset

retrieves behavioral data from running APKs. Powered by two core modules, the data-sharing process can be started and terminated within seconds.

3.2 Data Description

Until March 2022, we have collected a total of 85081 records from seven devices. The component status data is a discrete sequential structure stored in JSON files. The preliminary dataset, whose specifications are introduced in Table 1, available in our Github Repository and will be continuously updated to include more users, devices and apps. We divide the data set by device into these folders. Each folder contains two JSON files, one containing device information and the other is consists of a list of component state data received from the device.

4 SOFTWARE

Guided by the aforementioned design requirements, we designed and developed PartFlow, a visual analytics system integrating a series of analytical methods with a multi-aspects visual interface for analyzing component performance and transition generated by one certain user's device. In the following, we first provide an overview of PartFlow, and then describe its visual interface in detail.

4.1 PartFlow Overview

Fig. 2 shows an overview of PartFlow architecture, indicating the relationship among the front-end visualization, the back-end analytics measures, data storage, and external user devices.

Users upload their objective applications and download processed applications from server storage. Through launching and using processed applications on their external devices, the MongoDB

database receives invocation data of methods. From the database storing the component data, the back-end performs three analytical processes, including 1) performance estimation, 2) stochastic matrix calculating, and 3) series forecasting. Based on the data from analytical processes, four visualization panels are presented including 1) the Query Panel (B), 2) the Performance Panel (C), 3) the Transition Panel (D) and 4) the Forecasting Panel (E), which are provided to the visualization as needed in views of method execution patterns. Those panels and their sub-panels are shown in Fig. 1.

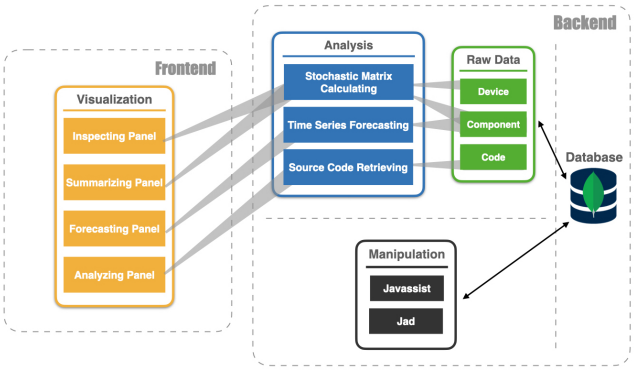

Figure 2: PartFlow Architecture Overview.

4.2 Evaluating Component Performance

The Performance Panel is shown as Fig. 1(C), which consists of three parts, Source Code Panel, Timeline Panel, and Tree Panel. By clicking the node (d1) or the rectangle (c2) in Timeline Panel, user can automatically release the Source Code Panel (c4) and obtain the source code fragment of the selected component.

We use the flame graph in Fig. 1(C) to visualize the timeline of method execution and use the sunburst graph to visualize the inner structure and execution time of a certain component. By clicking on rectangle (c2) in the Timeline Panel, there will be the Tree Panel (c3) displaying above the rectangle. The sunburst-like diagram is an interactive graph. The selected method is represented by the round located in the inner center. Around the central dot are several annulus sectors on different layers, representing different child methods levels inside the parent method. The ratio of the angle of the annulus sector represents the percentage of the execution time of the child method to its parent method. The green-to-red color represents the short to long execution time. For example, in Fig. 3, the central round in red surrounded by annulus (A) represents the execution time of a certain parent method `onClick()`, and its red color indicates that the time is over 200ms. The first annulus sector (B) around annulus (A) represents one of the first level child methods of this parent method, method `c()`. When the user click the annulus sector (B), the angle of the annulus sector (B) will transfer to 360 degree in order to focus on this child method `c()` and the annulus sector (A) will disappear. As Fig. 3(b) indicates, method `N()` spends mainly time of the method `c()`, it is necessary for the user to click the Fig. 3(c), the user easily obtain inner structure and execution time of third-level child method `N()`. In general, this sunburst diagram allows the user to see the internal call tree of each component as a reference for selecting components that need to be optimized.

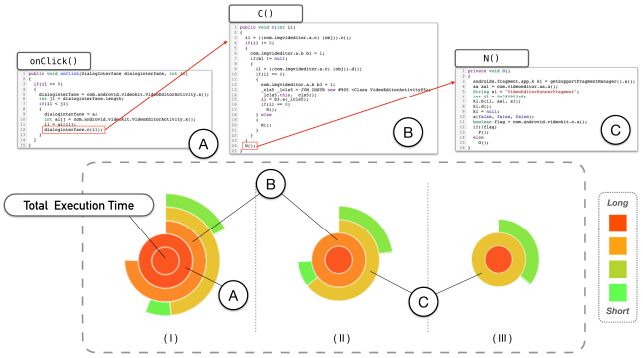

Figure 3: Each diagram below presents a nested method-level components.

4.3 Summarizing Component Transition

One of the main functions of PartFlow is to visualize the transition patterns among different states of components. We rely on the stochastic matrix to express the transition patterns in the form of a digraph shown in Fig. 1(D). Each node of the digraph in Fig. 1 represents a component state. For instance, when method `a()` starts executing (recorded as `a()#Start`), there will simultaneously generate the node (a) in the Fig. 4. The node color represents its class. By clicking different class legends, users can easily filter those massive nodes. The node size represents the number of the corresponding component state that has occurred, which can also be interpreted as the execution frequency of the corresponding component. The edge represents the transition of the component state, whose width indicates the number of transition occurrences. Therefore the wider edge means a higher probability of the transition occurring. Instead of using a traditional Markov Chain graph with arrows, we use edge color to distinguish the direction of an edge. For example, the directed edge from `a()#Start` to `b()#End` has the same color as the `a()#Start`. In conclusion, this circular digraph plays the same role as Markov Chain but performs in a more concise way.

4.4 Forecasting Sequence of Component States

In order to make dynamic partitioning strategies based on stochastic patterns, researchers also hope that there can be an interactive decision tree to arrange a sequence of states for picking objective sequences, which are actually possible to happen.

4.4.1 Markovian-based Forecasting Visualization. Inspired by Guo *et al.*'s Work[12], we borrow the same design of event sequence using a state transition graph based on Sankey diagrams. Fig. 5 shows the changing process of our forecasting sequence and the corresponding relationship between the digraph and the Sankey diagrams.

The squares are the same as nodes in Fig. 4 which both represent component states. When the user selects a certain node (a) in Fig. 5

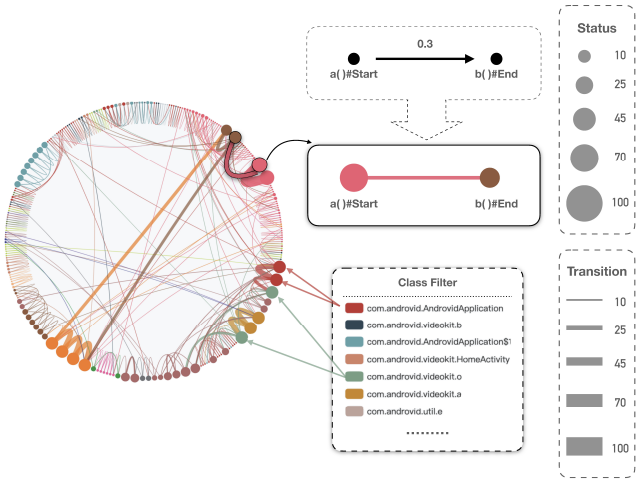

Figure 4: The visualization of transition patterns of one certain call log set.

as the start node, in the Forecasting Panel, there will generate a red rectangular node (a), while the border of node (a) in the digraph will transfer to red indicating those two nodes are same. Synchronously, several light blue adjacency nodes are generated. These light blue nodes represent all the possibilities of transition from the current component state to the next state.

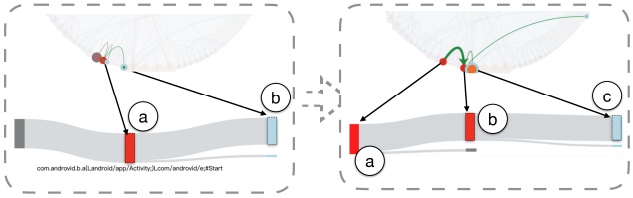

Figure 5: Two successive states of the Forecasting Panel.

As the component state sequence is time-discrete. we assume this event sequence as

$$S = \{X_1, X_2, X_3, \ldots, X_n\}$$

Assuming that the value of the start node is $\Pr(X_1 = x_1) = 1$, according to **R5**, when x_1 occurs, users are supposed to obtain the probability of flowing to each component state. When x_j occurs, the value of node j would be equal to the probability of $S' = \{x_1, x_2, \ldots, x_j\}$ occurring given the occurrence of $X_1 = x_1$ would be

$$\begin{aligned}\Pr(S'|X_1 = x_1) = &\Pr(X_j = x_j|X_{j-1} = x_{j-1}, \ldots, X_1 = x_1)\cdot \\ &\Pr(X_{j-1} = x_{j-1}|X_{j-2} = x_{j-2}, \ldots, X_1 = x_1)\cdot \\ &\ldots \cdot \Pr(X_2 = x_2|X_1 = x_1)\end{aligned}$$

In order to reveal the probability variance, we use the size of the node to indicate the probability of the sequence occurrence. For example, the start node of X_1, whose value is 1, and the value of node of X_2 would be $\Pr(X_1 = x_1) \cdot \Pr(X_2 = x_2) < 1$, whose size is also smaller than the node of X_1. In Parallel, the size of the node of X_6 represents the probability of this entire sequence occurrence.

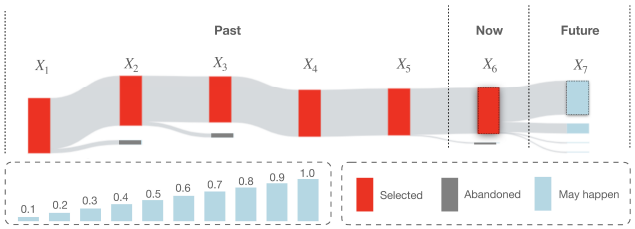

Figure 6: A Sankey-based decision tree for component state forecasting.

Obviously, the sequence of component states is from left to right, and the nodes' size is gradually small, which shapes the whole diagram to be similar to river flow.

4.4.2 DL-based Sequential State Prediction. Markov models can give us reasonable prediction results due to their interpretability. However, they make the assumption that the future is only dependent on the current state, whereas in reality, previous states may also have an impact. More frequently, generated wrong and missing forecasting results can not be handled by Markov models because they will make the prediction based on the previous incorrect states. To break the limitations of Markov models, we introduce a DL-based model for sequential state prediction. Specifically, we chose a typical kind of recurrent neural network (RNN) called long short-term memory (LSTM) as the prediction model. LSTMs are able to handle missing data and handle data with temporal dynamics, which makes them suitable for time-series prediction.[16] Before training, the raw sequential data need to be encoded into the one-hot format. Since one component has two states as mentioned, there will be $2 \times n_{method}$ features contained in the input data. Users can define their preferred input length n_{input}. Here we set $n_{input} = 10$ by default. After encoding, a DNN with a 120-units LSTM layer is deployed as the first hidden layer. Then a densely-connected layer is subsequently applied, which is shaped as $\langle 2 \times n_{method} \rangle$. Then, by the softmax process, we can get a method most likely to be executed. The general description of the architecture of the LSTM-based model is shown in Fig. 7.

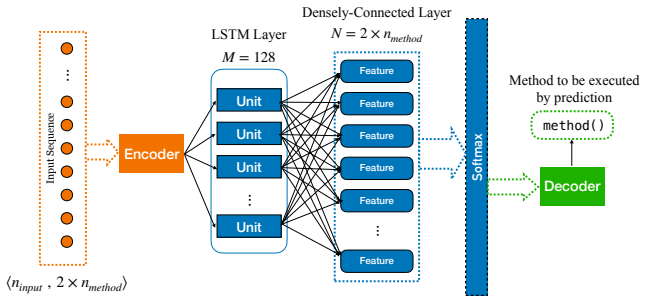

Figure 7: The brief architecture of LSTM-based DNN model.

In terms of a totally wrong inferred sequence, we recommend users refer to the prediction result from the LSTM model first and choose the next component states based on the inference.

5 EVALUATION

In this section, we evaluate the accuracy and reliability of the DL-based sequential states prediction module. The results indicate that PartFlow can fulfill the users' need for multi-step prediction of method-level component states.

Multi-step forecasting is different from single-step forecasting, which only predicts the next value in a sequence.[7] To perform multi-step forecasting, a model is trained on historical data and then used to make predictions for a specific number of steps in the future. The model takes into account the patterns and trends in past data to make predictions about future values. To estimate the overall performance of the LSTM model for multi-step forecasting, we choose the basic Markov Model (MM) and support vector machine (SVM) as two baseline models.

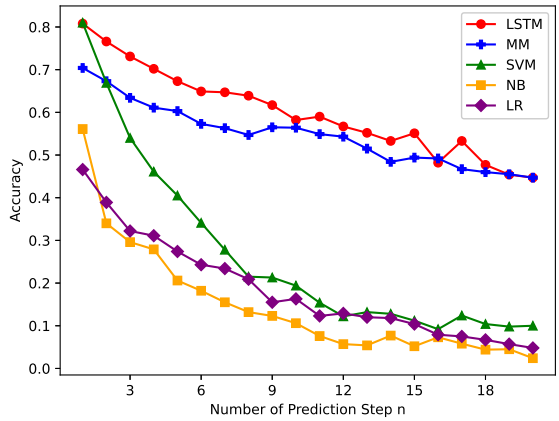

Figure 8: Prediction results of LSTM, MM, and SVM along the number of prediction step n.

The results shown in Fig. 8 suggest that the LSTM model is reliable while facing the multi-step component states prediction. Compared to the LSTM model, MM achieves the same high accuracy when $n = 1$, which is over 80%. However, when facing the situation of $n > 1$, the accuracy of MM steeply decreases as the n increases. Compared to MM, SVM has a much better performance when dealing with the increasing n, whereas it has a relatively low accuracy when $n = 1$.

Thus, according to the above results, we recommend you to select the component states by both the transition probability and the result predicted by the LSTM model for better user experience.

6 CONCLUSION

We have presented PartFlow, an interactive visualization for assisting researchers to dynamic analyze the user-driven execution patterns for application partitioning and workload offloading. PartFlow dynamically displays multi-dimensional information of application component states based on the data collected from multiple users and devices.

By involving expert users, we derived a set of design requirements and iteratively refined the system in a human-centered design process. For evaluation, we reported three case studies and a user study to illustrate the effectiveness, usefulness, and quality of According to the feedback from the evaluation section, PartFlow received an excellent overall review respecting all the provided functions.

REFERENCES

[1] Ragaad AlTarawneh and Shah Rukh Humayoun. 2016. Visualizing software structures through enhanced interactive sunburst layout. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, 288–289.

[2] Georges-Pierre Bonneau, Hans-Christian Hege, Chris R Johnson, Manuel M Oliveira, Kristin Potter, Penny Rheingans, and Thomas Schultz. 2014. Overview and state-of-the-art of uncertainty visualization. In *Scientific Visualization*. Springer, 3–27.

[3] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. 2016. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24, 5, 2795–2808. DOI: 10.1109/TNET.2015.2487344.

[4] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 49–62.

[5] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. 2014. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32, 2, 1–29.

[6] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. 2013. Mobile cloud computing: a survey. *Future generation computer systems*, 29, 1, 84–106.

[7] Antonio Galicia, R Talavera-Llames, A Troncoso, Irena Koprinska, and Francisco Martınez-Álvarez. 2019. Multi-step forecasting for big data time series based on ensemble learning. *Knowledge-Based Systems*, 163, 830–841.

[8] David Gotz and Harry Stavropoulos. 2014. Decisionflow: visual analytics for high-dimensional temporal event sequence data. *IEEE transactions on visualization and computer graphics*, 20, 12, 1783–1792.

[9] Brendan Gregg. 2016. The flame graph. *Communications of the ACM*, 59, 6, 48–57.

[10] Brendan Gregg. 2010. Visualizing system latency. *Commun. ACM*, 53, 7, 48–54. DOI: 10.1145/1785414.1785435.

[11] Brendan Gregg. 2010. Visualizing system latency: heat maps are a unique and powerful way to visualize latency data. explaining the results, however, is an ongoing challenge. *Queue*, 8, 5, 30–42.

[12] Shunan Guo, Fan Du, Sana Malik, Eunyee Koh, Sungchul Kim, Zhicheng Liu, Donghyun Kim, Hongyuan Zha, and Nan Cao. 2019. Visualizing uncertainty and alternatives in event sequence predictions. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–12.

[13] Alexander Kumpf, Bianca Tost, Marlene Baumgart, Michael Riemer, Rüdiger Westermann, and Marc Rautenhaus. 2017. Visualizing confidence in cluster-based ensemble weather forecast analyses. *IEEE transactions on visualization and computer graphics*, 24, 1, 109–119.

[14] Minghao Li and Wei Cai. 2022. A blockchain-based profiling system for exploring human factors in cloud-edge-end orchestration. In *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 13–18.

[15] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, and Oliver Spatscheck. 2010. Characterizing radio resource allocation for 3g networks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 137–150.

[16] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Charu Aggarwal, Prasenjit Mitra, and Suhang Wang. 2020. Joint modeling of local and global temporal dynamics for multivariate time series forecasting with missing values. In *Proceedings of the AAAI Conference on Artificial Intelligence* number 04. Vol. 34, 5956–5963.

[17] Liang Tong and Wei Gao. 2016. Application-aware traffic scheduling for workload offloading in mobile clouds. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 1–9.

[18] Krist Wongsuphasawat and David Gotz. 2012. Exploring flow, factors, and outcomes of temporal event sequences with the outflow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18, 12, 2659–2668.

[19] Jian Zhao, Zhicheng Liu, Mira Dontcheva, Aaron Hertzmann, and Alan Wilson. 2015. Matrixwave: visual comparison of event sequence data. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 259–268.