



# Contention-aware prediction for performance impact of task co-running in multicore computers

Shenyuan Ren<sup>1</sup> · Ligang He<sup>1</sup> · Junyu Li<sup>1</sup> · Zhiyan Chen<sup>1</sup> · Peng Jiang<sup>1</sup> · Chang-Tsun Li<sup>2</sup>

Published online: 13 February 2019  
© The Author(s) 2019

## Abstract

In this paper, we investigate the influential factors that impact on the performance when the tasks are co-running on a multicore computers. Further, we propose the machine learning-based prediction framework to predict the performance of the co-running tasks. In particular, two prediction frameworks are developed for two types of task in our model: repetitive tasks (i.e., the tasks that arrive at the system repetitively) and new tasks (i.e., the task that are submitted to the system the first time). The difference between which is that we have the historical running information of the repetitive tasks while we do not have the prior knowledge about new tasks. Given the limited information of the new tasks, an online prediction framework is developed to predict the performance of co-running new tasks by sampling the performance events on the fly for a short period and then feeding the sampled results to the prediction framework. We conducted extensive experiments with the SPEC2006 benchmark suite to compare the effectiveness of different machine learning methods considered in this paper. The results show that our prediction model can achieve the accuracy of 99.38% and 87.18% for repetitive tasks and new tasks, respectively.

**Keywords** Performance prediction · Multicore computing · Scheduling

## 1 Introduction

In task scheduling, it is often assumed that the scheduler knows the execution time of the tasks, based on the assumption that the techniques are present to predict the performance of tasks. However, it is a non-trivial task to produce accurate performance prediction for tasks, although a number of techniques are indeed developed to predict the task performance [queuing-theory based] [12]. The situation worsens for the tasks that are running simultaneously (co-running) on multiple CPU cores in a multi-core processor for the following reasons. There is resource contention and interference among the co-running tasks, since they need to share (contend) the resources in the computer such as internal buses, cache, memory, hard disk, etc. The resource contention may lead to the longer

completion times of the tasks. The resource contention relation is complicated because both the intensity level of contention and the type of the contention (i.e., which type of the resource is contended by the tasks most intensively) do not only relate to the hardware specification of the system (such as cache size and memory bandwidth), but also vary from the characteristics of the co-running tasks (such as memory access frequency, I/O requirement and cache usage). The complexity nature of the contention among co-running tasks makes it difficult to develop the static formulas for accurate performance prediction.

In this paper, we investigate the performance of the co-running tasks on multi-core chips and present the method to identify the influential factors, which are the performance events provided by the Operating Systems, for the given co-running tasks. Further, we propose a machine learning-based approach to predicting the performance of the co-running tasks. Two prediction frameworks are developed for two types of task in our model: repetitive tasks (i.e., the tasks that arrive at the system repetitively) and new tasks (i.e., the tasks that are submitted to the system the first time). The difference between which is that we have the historical running information of the repetitive

---

✉ Ligang He  
ligang.he@warwick.ac.uk

<sup>1</sup> Department of Computer Science, University of Warwick, Warwick, UK

<sup>2</sup> School of Information Technology, Deakin University, Burwood, VIC 3125, Australia

tasks while we do not have the prior knowledge about new tasks.

Given the limited information of the new tasks, a two-stage online prediction framework is developed to predict the performance of the co-running new tasks by sampling the performance events on the fly for a short period and then feeding the sampled results to the prediction framework. We conducted extensive experiments with the SPEC2006 benchmark suite to compare the effectiveness of different machine learning methods considered in this paper and present our observations and analysis. The results show that our prediction model can achieve the accuracy of 99.38% and 87.18% for repetitive tasks and new tasks, respectively.

### 2 Motivating benchmark experiments

We used SPEC2006 to conduct the benchmarking experiments to investigate the impact of task co-running on performance.

Figure 1 shows the execution time of three benchmarks in SPEC2006, 401 (401.perlbench is a compression program), 416 (416.gamess is a wide range of quantum chemical computations) and 470 (470.lbm is a computational fluid dynamics program using the lattice boltzmann method) [9], when they co-run with other SPEC2006 benchmarks on a multi-core processor. In Fig. 1a, the execution time of the solo-run of SPEC 401 (i.e., when the benchmark runs on a core without other programs co-running on other cores) is 89.4 s. When co-running with other benchmarks, the execution time of 401 vary from 89.50 (co-running with 462) to 114.92 (co-running with 470). Its performance degradation is noticeable: from 0 to 28%. The same phenomenon occurs with other benchmarks in SPEC. Through the experiment, we also observed that some benchmarks, such as SPEC 470 are more contention-sensitive (up to 65%) than others, such as SPEC 444 (up to 5%). In Fig. 1b, experiments are conducted on a quad-core processor to show the performance impact of SPEC 416(470) when it is running with different degrees of contentions. The x-axis represents the number of co-running

tasks. For example, x-axis '0' means that SPEC 416(470) is solo-running with no resource contention; x-axis '3' means that there are four SPEC 416(470) tasks co-running on different cores within the quad-core processor. The y-axis represents the ratio of  $Makespan_{co-run}$  to  $Makespan_{solo}$ . We can see that the resource contention leads to an enormous increasing of completion time of SPEC 470; While there is not any performance impact on SPEC 416 during the co-running.

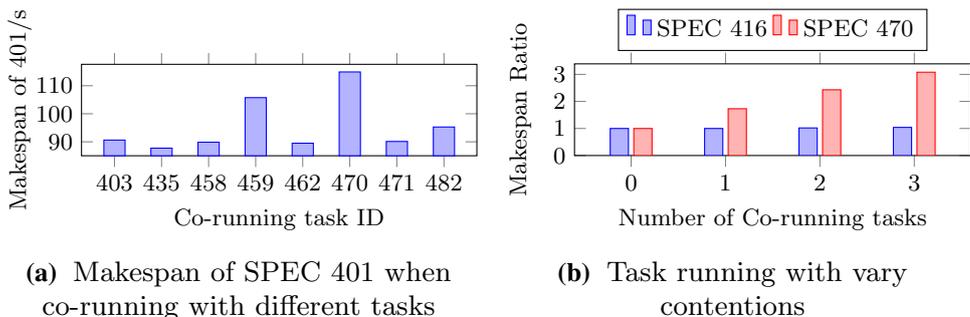
Almost all latest researches cite cache miss and memory bandwidth as the most important factors that affect the performance of co-running tasks. Based on these two factors, several performance models are constructed to formulate the impact of task co-running on multi-core processors [2, 5, 6, 18]. However, our research show that more factors show noticeable impact on the co-running performance, such as branch-misses, context-switches, and minor faults, etc. We collect 30 performance events provided by the Operating System during the execution of co-running tasks, as shown on the x-axis of Fig. 2. Figure 2 shows the ratio of the values of these performance events gathered when SPEC 459 co-runs with 470 to those when SPEC 459 solo-runs. As can be seen from this figure, the values of the performance events have considerable changes when the benchmarks co-run.

### 3 The performance prediction framework for co-running tasks

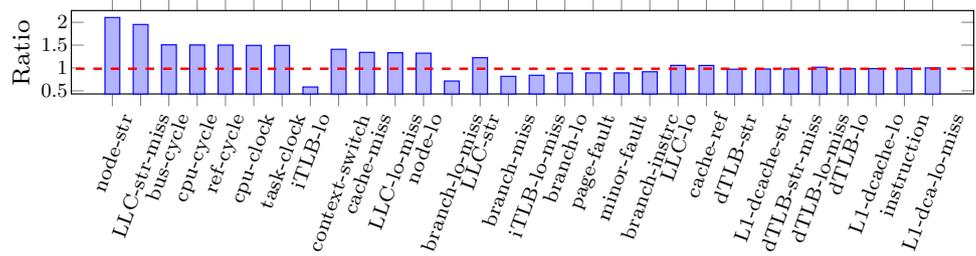
We develop the performance prediction framework for both repetitive tasks, which has been run in the system before and therefore we have the historical performance event data when the task solo-runs, and new tasks, which are submitted to run on the system the first time.

When the co-running tasks are the repetitive tasks, we use the historical performance event data of individual tasks (their solo-run performance data) as the input of the prediction framework. When the co-running tasks contain new tasks, we do not have the prior knowledge about the new tasks and therefore different procedure is developed in this paper for new tasks.

Fig. 1 Motivation experiments



**Fig. 2** Comparison of performance events of SPEC 459 between solo execution and co-running



The data of all 30 performance events can be collected using Perf (a profiling tool in Linux) during the execution of the tasks. We introduce a notion called the *the rate of performance event*, which equals to the collected value of a performance event divided by the execution time of the task (i.e., the frequency at which the performance event occurs during the execution of the task). We then use the rates of the performance events as the attributes of the training model.

The performance impact of a task is defined as the ratio of co-running completion time to its solo completion time. When task  $t_i$  and  $t_j$  co-run, the predicted performance impact of  $t_i$ , denoted by  $PI\_T_i$ , is represented as in 1, where  $\Gamma_{co-running}$  is a trained model,  $PE_i$  and  $PE_j$  are the set of solo-run performance events of task  $t_i$  and  $t_j$ , respectively.

$$PI\_T_i = \Gamma_{co-running}(PE_i, PE_j) \tag{1}$$

In order to understand why our prediction model for new tasks works, see a benchmarking experiment we conducted. Figure 3 shows the trend of the selected performance events of SPEC 401 during its co-running with SPEC 403. In the experiments, we collect the performance event data once every 500ms. The execution time of SEPC 401 (1 iteration) is 30 s. Thus we obtained 60 sampled data (time intervals). As can be seen from this figure, many performance events show repeated or similar trend as the co-running tasks progress, which provides a ground for our prediction model for new tasks.

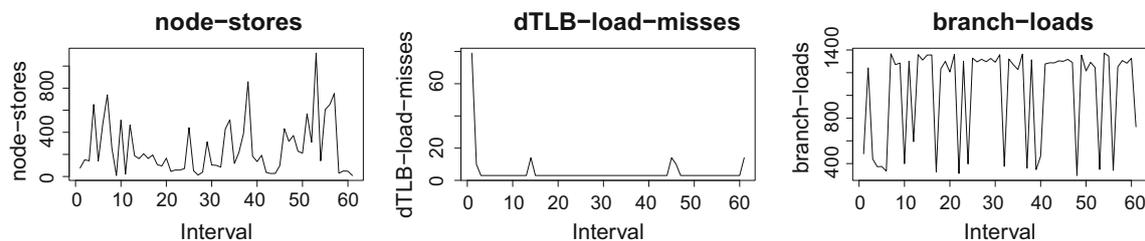
Since we do not have the prior knowledge about the new tasks, we develop a two-stage prediction framework. In the first stage, we construct a *prediction model for each performance event*. Thus we have 30 models in total, corresponding to the 30 performance events. We sample the

performance events at a preset sampling rate for a preset period when the task is solo-executing; Then we sample the performance events for the same length of period when the task is co-running with other tasks. For example, the performance events are sampled once every 200ms for 2 s for a task of 100 s. The prediction model for performance event takes the sampled data of the performance event as input and predicts the value of the performance event when the solo-running and the co-running task complete, respectively. In the second stage, we use the impact ratio of the performance events to predict the performance impact of the co-running tasks. The impact ratio is defined as the ratio of the value of the co-running performance event to the value of the solo-running performance event that is predicted in the first stage.

The two-stage prediction model is formulated as follows.  $t_{sp}$  denotes the time period of sampling and  $int$  represents the sampling interval (the inverse of the sampling rate). Then  $t_{sp}/int$  is the number of sampled data we obtain for a performance event.

The predicted data of the performance event when the solo-running task (or co-running task)  $t_i$  completes, denoted by  $PPE_i^{solo}$  (or  $PPE_i^{co}$ ), can be represented by the vector that is derived from Eq. 2 (or 3).  $F_1^n$  represents the prediction model for performance event  $n$ .  $s\_PE_j^n$  and  $c\_PE_j^n$  denote the performance event data of the  $j$ -th sampling interval for performance event  $n$  under solo-run and co-run, respectively. In our work, there are in total 30 performance events (i.e.,  $m = 30$ ). Thus we train 30 models in the first stage.

$$PPE_i^{solo} = [F_1^1(s\_PE_1^1, \dots, s\_PE_{t_{sp}/int}^1), \dots, F_1^m(s\_PE_1^m, \dots, s\_PE_{t_{sp}/int}^m)] \tag{2}$$



**Fig. 3** The trend of performance events as the co-running tasks progress

$$PPE_i^{co} = \left[ F_1^1(c_{PE_1^1}, \dots, c_{PE_{sp/int}^1}), \dots, F_1^m(c_{PE_1^m}, \dots, c_{PE_{sp/int}^m}) \right] \quad (3)$$

In the second stage, the impact ratio vector of task  $t_i$ , denoted by  $IR_i$ , can be derived from:

$$IR_i = \frac{PPE_i^{co} - PPE_i^{solo}}{PPE_i^{solo}} \quad (4)$$

Then the prediction model for the performance impact of task  $t_i$  can be represented by

$$PI_{T_i} = \Gamma'_{co-running}(IR_i) \quad (5)$$

where  $\Gamma'_{co-running}$  represents the trained model for predicting the performance impact of new tasks. Note that we do not need the performance event data for the co-running task  $t_j$  as the input of this formula because the execution information of  $t_j$  has been reflected in the sampled performance event data since tasks  $t_i$  and  $t_j$  are co-running. Furthermore, we can predict the performance impact of a specific task in the same way, no matter how many tasks it is co-running with in a multi-core processor.

In the above model representations,  $PPE_i^{solo}$  and  $PPE_i^{co}$  represents the first stage work while  $PI_{T_i}$  represents the second stage work in the two-stage prediction model. The machine learning approaches used in our prediction frameworks are: linear regression, naive Bayes, support-vector machine (SVM) and random forest. We examined the four popular machine learning approaches mentioned, aiming to identify most effective approach for our scenario.

## 4 Evaluation

The accuracy of our prediction frameworks plugged with the above four machine learning approaches is evaluated in this section. The testing environment is a Personal Computer with a 3.30 GHz dual-core Intel i5 CPU and 8 GB memory. It has 32k L1d cache, 32k L1i cache, 256k L2 cache and 6144k L3 cache. 28 benchmarks used in the evaluation come from the Standard Performance Evaluation Corporation (SPEC) benchmark suite. There are For each application, the features are collected only once statically. These data are repeatedly used for training and predicting. Note that the training time of our models vary from 2.1 to 4.6 s and the prediction time of one instance is very short and can be neglected.

We co-run the benchmarks in SPEC on two cores. There are in total  $\binom{28}{2} = 406$  combinations. Each combination generates two instances. Therefore, we obtained 812 pieces

of training data. We choose a 4:1 split for training and testing. Table 1 shows the experiment results for predicting the performance impact of co-running repetitive tasks and new tasks. We divide our experiments into four categories: Static regression and static classification are for repetitive tasks while online regression is for new tasks.

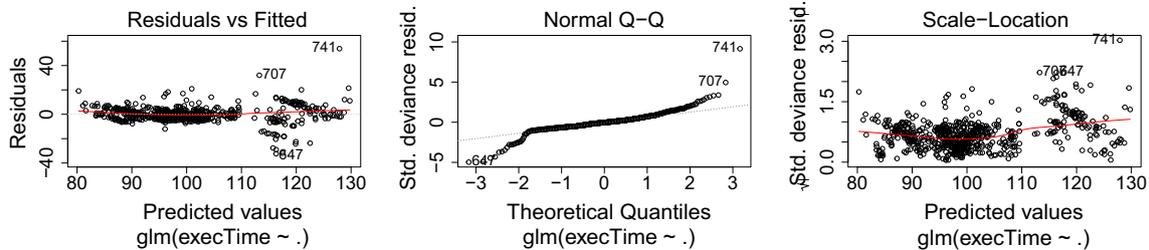
Figures 4 and 5 are the residual analysis of the trained model. 1) In the “Residual vs. Fitted” figure, the y axis is the residual while the x axis is the fitted values. The residuals “bounce randomly” around the 0 line. This suggests that the assumption that the relationship is linear is reasonable. There are several outliers that need to be removed from the training set, such as 647, 619 and 707, etc. 2) In the “Normal Q-Q” figure, the results suggest that the residuals (and hence the error terms) are normally distributed but with the several outliers. 3) The “scale-location” plot shows whether the residuals are spread equally along the ranges of predictors. Here, we can see a horizontal line with equally (randomly) spread points, which suggests that the two trained models satisfy homoscedastic. 4) the “Residuals vs Leverage” plot helps to find the influential cases if any. The results for the static model are to be expected except there exit several potential problematic cases in the training set of the online model (with the row numbers of the data in the dataset).

The predicting result for the regression is a number while the predicting result for classification is a range. For the regression, we set a tolerance of 3%. Namely, if the difference between the predicting result and the actual measurement is less than 3%, we regard the predicting result as being correct. We set this tolerance because the measured co-running time is not constant. The execution time of a specific task fluctuates even when it co-runs with the same task. For the classification, we set the reasonable ranges for the data. The difference between the upper bound and lower bound is around 3% of the average performance impact of the application. If the actual performance impact resides within the range that we predict, we regard the predicting result as being correct.

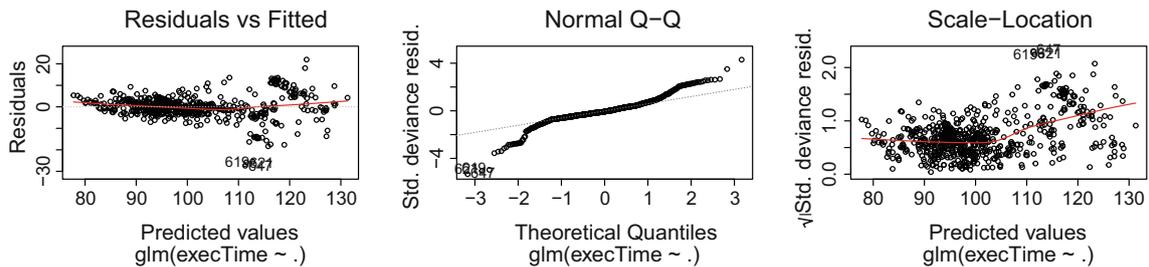
In Table 1, we observe that our model can predict the co-running time accurately for most applications. Furthermore, among the predictors, SVM and random forest have the best accuracy for both static and online tests. Random forest achieves over 90% accuracy in both regression and classification for repetitive tasks. We also preprocess the data by discretizing and normalizing the data (denoted by d&n), comparing with the prediction accuracies in the cases of non-discretizing (denoted by n-d) and non-normalizing (denoted by n-n). We discretizes the attributes (i.e. performance event) into specific number of bins. This operator discretizes the selected numerical attributes to nominal attributes. To achieve this, a width is calculated from  $(max-min)/bins$ .  $max$  and  $min$  represent the maximum

**Table 1** Experiment results for predicting the co-running time for repetitive tasks and new tasks, respectively

Predictors	Static reg			Static classifi			Online reg
	d&n (%)	n-d (%)	n-n (%)	d&n (%)	n-d (%)	n-n (%)	d&n (%)
Regression	87.65	70.99	86.42	87.04	81.48	83.59	56.41
Naive Bayes	–	–	–	70.20	66.89	68.21	–
SVM	98.15	90.12	96.91	86.06	78.15	76.16	87.18
Random forest	99.38	96.68	98.77	92.05	84.77	94.04	87.18



**Fig. 4** Residual analysis for co-running task prediction (online)



**Fig. 5** Residual analysis for co-running task prediction (static)

and the minimum of each attribute, respectively. The range of numerical values is partitioned into segments of equal size (i.e. *width*). Then we assigned the attributes to the corresponding segments. Discretization can help to reduce the categories to a reasonable number that the classifier is able to handle.

Comparing to the predicting result with non-discretizing and non-normalizing, the discretization and normalization performs a better accuracy in most cases. Discretized feature has strong robustness to the abnormal data. It makes the prediction model more stable and reduces over-fitting. Normalization accelerates the gradient descent and also achieves a higher accuracy, especially for SVM and linear regression. In the online prediction model (for new tasks), we did not normalize the data because the input data are ratios.

Furthermore, we order the features by their importance scores under the random forest predictor. The IncNodePurity reflects the total decrease in node impurities from splitting on the variable, averaged over all trees. We find the features which are important for the prediction. In the order of their decreasing importance scores, these features are instructions, bus cycles, LLC loads, L1 dcache

loads, dTLB loads, branch instructions, branch.loads and context switches. These features are essential for the prediction of performance impact because the prediction accuracy drops significantly (79.3% and 71.79% for static prediction and online prediction, respectively) when we remove any of these features from the feature set of the input data.

### 5 Related work

Several works have explored the performance degradation problem of co-running tasks. Performance and energy model are built to analyze and predict the performance impact [3, 11, 13, 15–17, 19] and [14]. Reference [2] studies the impact of L2 cache sharing on the concurrent threads; Reference [19] proposes an interference model that considers the time-variant inter-dependency among different levels of resource interference to predict the application QoS metric. Reference [18] decomposes parallel runtime into compute, synchronization and steal time, and uses the runtime breakdown to measure program progress and identify the execution inefficiency under

interference (in virtual machine environment). Reference [1] reveals that the cross-application interference problem is related to the amount of simultaneous access to several shared resources. Reference [10] predicts the execution time of an application workload for a hypothetical change of configuration on the number of CPU cores of the hosting VM. Reference [8] gained the insight into the principle of enriching the capability of the existing approaches to predicting the performance of multicore systems. Reference [7] develops an efficient ELM based on the Spark framework (SELM), which includes three parallel subalgorithms, is proposed for big data classification. Reference [4] proposes a Patient Treatment Time Prediction (PTTP) algorithm to predict the waiting time for each treatment task for a patient.

Most of the above studies consider that the features such as cache and bandwidth will impact on the co-running performance. However, our experimental results reveal that more performance events (such as ref-cycles, context-switches, branch-load-misses, etc.) may play influential roles. In this work, we take into account all performance events provided by the profiling tool, Perf.

## 6 Conclusions and future work

This paper investigate the influential factors that affect the performance of the co-running tasks. A performance model is built and several machine learning methods are applied to predict the performance impact of the co-running tasks. Experiments conducted with SPEC 2006 benchmark suite show that our prediction model of performance impact achieves the accuracy of 99.38% and 87.18% on repetitive tasks and new tasks, respectively.

In future, our research will extend to modeling the performance impact of running the tasks with different CPU frequencies and also modeling the performance impact of time-sharing execution.

**Acknowledgements** The project is partially supported by the China Scholarship Council and EU Horizon 2020—Marie Skłodowska-Curie Actions through the project entitled Computer Vision Enabled Multimedia Forensics and People Identification (Project No. 690907, Acronym: IDENTITY).

**Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## References

- Alves, M. M., de Assumpção, D., & Lúcia, M. (2017). A multivariate and quantitative model for predicting cross-application interference in virtual environments. *Journal of Systems and Software*, 128, 150–163.
- Chandra, D., Guo, F., Kim, S., & Solihin, Y. (2005). Predicting inter-thread cache contention on a chip multi-processor architecture. In *11th International Symposium on High-performance Computer Architecture, 2005 (HPCA-11)* (pp. 340–351). IEEE.
- Chen, J., Li, K., Tang, Z., Bilal, K., Yu, S., Weng, C., et al. (2017). A parallel random forest algorithm for big data in a spark cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 28(4), 919–933.
- Chen, J., Li, K., Tang, Z., Bilal, K., & Li, Keqin. (2016). A parallel patient treatment time prediction algorithm and its applications in hospital queuing-recommendation in a big data environment. *IEEE Access*, 4, 1767–1783.
- Dauwe, D., Jonardi, E., Friese, R., Pasricha, S., Maciejewski, A. A., Bader, D. A., et al. (2015). A methodology for co-location aware application performance modeling in multicore computing. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW)* (pp. 434–443). IEEE.
- Dauwe, D., Jonardi, E., Friese, R. D., Pasricha, S., Maciejewski, A. A., Bader, D. A., et al. (2016). Hpc node performance and energy modeling with the co-location of applications. *The Journal of Supercomputing*, 72(12), 4771–4809.
- Duan, M., Li, K., Liao, X., & Li, K. (2018). A parallel multi-classification algorithm for big data using an extreme learning machine. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6), 2337–2351.
- Frank, M., Hilbrich, M., Lehrig, S., & Becker, S. (2017). Parallelization, modeling, and performance prediction in the multi-/many core area: A systematic literature review. In *2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2)* (pp. 48–55). IEEE.
- Henning, J. L. (2006). Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4), 1–17.
- Li, H.-W., Yu-Sung, W., Chen, Y.-Y., Wang, C.-M., & Huang, Y.-N. (2017). Application execution time prediction for effective cpu provisioning in virtualization environment. *IEEE Transactions on Parallel and Distributed Systems*, 28(11), 3074–3088.
- Liu, C., Li, K., Chengzhong, X., & Li, K. (2016). Strategy configurations of multiple users competition for cloud service reservation. *IEEE Transactions on Parallel and Distributed Systems*, 27(2), 508–520.
- Matsunaga, A., & Fortes, J. A. B. (2010). On the use of machine learning to predict the time and resources consumed by applications. In *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing* (pp. 495–504). IEEE Computer Society.
- Mei, J., Li, K., & Li, K. (2017). Customer-satisfaction-aware optimal multiserver configuration for profit maximization in cloud computing. *IEEE Transactions on Sustainable Computing*, 2(1), 17–29.
- Oxley, M. A., Jonardi, E., Pasricha, S., Maciejewski, A. A., Siegel, H. J., Burns, P. J., et al. (2018). Rate-based thermal, power, and co-location aware resource management for heterogeneous data centers. *Journal of Parallel and Distributed Computing*, 112, 126–139.
- Tang, Z., Ma, W., Li, K., & Li, K. (2016). A data skew oriented reduce placement algorithm based on sampling. *IEEE Transactions on Cloud Computing*. <https://doi.org/10.1109/TCC.2016.2607738>.

16. Tang, Z., Qi, L., Cheng, Z., Li, K., Khan, S. U., & Li, K. (2016). An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment. *Journal of Grid Computing*, 14(1), 55–74.
17. Yuming, X., Li, K., Jingtong, H., & Li, K. (2014). A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Information Sciences*, 270, 255–287.
18. Zhao, Y., Rao, J., & Yi, Q. (2016). Characterizing and optimizing the performance of multithreaded programs under interference. In *2016 International conference on parallel architecture and compilation techniques (PACT)* (pp. 287–297). IEEE.
19. Zhu, Q., & Tung, T. (2012). A performance interference model for managing consolidated workloads in QOS-aware clouds. In *2012 IEEE 5th international conference on cloud computing (CLOUD)* (pp. 170–179). IEEE.



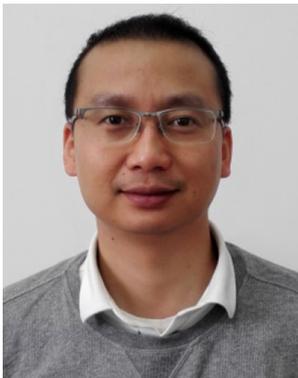
**Shen yuan Ren** is a PhD student in the Department of Computer Science at the University of Warwick, UK. Her research area is parallel and distributed computing, and Cloud computing.



**Junyu Li** is a PhD student in the Department of Computer Science at the University of Warwick. His research area is deep learning, parallel and distributed computing.



**Zhiyan Chen** is a PhD student in the Department of Computer Science at the University of Warwick, United Kingdom. His research area is mobile cloud computing.



**Ligang He** received the BSc and MSc degree from Huazhong University of Science and Technology, China, and received the PhD degree in Computer Science at the University of Warwick, United Kingdom. After PhD studies, he worked as a post-doctoral researcher at the University of Cambridge, UK, and then joined in the Department of Computer Science at the University of Warwick as Assistant Professor. He is now the Associate Pro-

fessor in the department. His research interests focus on parallel and distributed processing and Cloud computing. He has published more than 100 papers in international conferences and journals. He has been a co-chair or a member of the program committee for a number of international conferences, and been the reviewers for many international journals. He is a member of the IEEE.



**Peng Jiang** is a PhD student in the Department of Computer Science at the University of Warwick. His research area is Cloud computing, and parallel and distributed computing.



**Chang-Tsun Li** received the BEng degree in electrical engineering from National Defence University (NDU), Taiwan, in 1987, the MSc degree in computer science from U.S. Naval Postgraduate School, USA, in 1992, and the PhD degree in computer science from the University of Warwick, UK, in 1998. He was an associate professor of the Department of Electrical Engineering at NDU during 1998-2002 and a visiting professor of the Department of

Computer Science at U.S. Naval Postgraduate School in the second half of 2001. He was a professor of the Department of Computer Science at the University of Warwick (UK) until January 2017 and a professor of Charles Sturt University (Australia) from January 2017 to

February 2019. He is currently a professor of the School of Information Technology, Australia. His research interests include multimedia forensics and security, biometrics, data mining, machine learning, data analytics, computer vision, image processing, pattern recognition, bioinformatics, and content-based image retrieval. The outcomes of his multimedia forensics research have been translated into award-winning commercial products protected by a series of international patents and have been used by a number of police forces and courts of law around the world. He is currently Associate Editor of the EURASIP Journal of Image and Video Processing (JIVP) and Associate of Editor of IET Biometrics. He involved in the organisation of many international conferences and workshops and also served as member of the international program committees for several international conferences. He is also actively contributing keynote speeches and talks at various international events.

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.