УДК 004.85
DOI 10.52575/2687-0932-2021-48-4-764-770

Использование технологии гиперпоточности в целях повышения скорости обработки ML-алгоритмов

[1] Воробьев А.В., [2] Распопин Д.И.
[1] Курский государственный университет
Россия, г. Курск, ул. Радищева, 33
E-mail: 505216@inbox.ru
[2] Юго-Западный государственный университет
Россия, г. Курск, ул. 50 лет Октября, 94
E-mail: dr9524934961@gmail.com

Аннотация
В статье проведен анализ скорости обработки данных алгоритмами машинного обучения в зависимости от доступных вычислительных ресурсов CPU и объема набора данных. Испытания проводились на синтезированных тестовых наборах нарастающей размерности от 100 наблюдений и 100 предикторов до 2000 наблюдений и 2000 предикторов с использованием модернизированного ансамблевого алгоритма. В результате исследования определено, что при использовании исключительно доступных вычислительные ресурсов CPU прирост скорости выполнения алгоритма требует значительно опережающего темпа приростов суммарной вычислительной мощности, определена исчисляемая пропорция прироста для частной задачи. Рассмотрена технология гиперпоточности в качестве инструмента повышения производительности CPU. В ходе экспериментов определено, что обработка алгоритмов машинного обучения в однопоточном приложении – языковой среде Python – не является ограничением использования гиперпоточности, напротив, применение данной технологии может повысить скорость обработки ML-алгоритмов.

Ключевые слова: машинное обучение, ансамблевые алгоритмы, гиперпоточность, однопоточное приложение.

Для цитирования: Воробьев А.В., Распопин Д.И. 2021. Использование технологии гиперпоточности в целях повышения скорости обработки ML-алгоритмов. Экономика. Информатика. 48 (4): 764–770. DOI: 10.52575/2687-0932-2021-48-4-764-770.

Using hyperthreading technology to increase the processing speed of ML algorithms

[1] Alexander V. Vorobyev, [2] Daniil I. Raspopin
[1] Kursk State University
33 Radisheva St, Kursk, Russia
E-mail: 505216@inbox.ru
[2] South-West State University,
94 50 let Oktyabrya St, Kursk, Russia
E-mail: dr9524934961@gmail.com

Abstract
The paper analyzes the data processing speed of machine learning algorithms depending on available CPU computing resources and data set size. Tests were conducted on synthesized test suites of increasing dimensionality, from 100 observations and 100 predictors, to 2000 observations and 2000 predictors, using a modern ensemble algorithm. As a result of the research it is determined that to increase the training speed of an ML-algorithm a much larger increase in computational power is required, given that the only computational power used is that of the CPU. A numerical exemplary proportion valid for a specific task is

provided. Hyperthreading technology as a tool for increasing CPU performance is considered. In the course of experiments it is determined that processing of machine learning algorithms in a single threaded application – Python language environment – is not a limitation for hyperthreading; on the contrary, using this technology can increase the processing speed of ML algorithms.

Keywords: machine learning, ensemble algorithms, hyperthreading, single-threaded application.

For citation: Vorobyev A.V., Raspopin D.I. 2021. Using hyperthreading technology to improve the processing speed of ML algorithms. Economics. Information technologies. 48 (4): 764–770 (in Russian). DOI 10.52575/2687-0932-2021-48-4-764-770.

Введение

Вычисления общего назначения с использованием графического процессора (GPU), были представлены в начале 2000-х годов и с тех пор стали популярной концепцией. Первыми примерами были ускорение простых алгоритмов, таких как умножение матрицы на матрицу, путем перефразирования алгоритма в виде операций над графическими примитивами [Larsen и др., 2001]. Это было громоздко, и не существовало инструментов разработки для вычислений общего назначения. Тем не менее многие алгоритмы были реализованы на графическом процессоре в качестве доказательства концепции, демонстрируя значительное ускорение по сравнению с центральной обработкой на процессоре (CPU) [Seung Won Min и др. 2021]. Сегодня среда разработки для вычислений на графическом процессоре значительно эволюционировала и является одновременно зрелой и стабильной: доступны передовые отладчики и профилировщики, что делает отладку, разработку на основе профилей и оптимизацию производительности проще, чем когда-либо [Rex Ying и др., 2018; Seung Won Min и др., 2021]. При этом изменения в глобальной конъюнктуре рынка GPU 2020–2021 годов под влиянием ряда факторов (дефицит полупроводников, рост курсов криптовалют и др.) определили снижение массовой доступности высокоэффективных графических процессоров. В данной работе мы рассматриваем практику использования вычисления алгоритмов машинного обучения с обработкой на CPU, определением зависимости скорости обработки от объема доступных вычислительных мощностей и использование технологий, позволяющих увеличить производительность.

Влияние вычислительной мощности на скорость обработки ML-алгоритмов

В целях определения влияния вычислительной мощности на скорость обучения ML-алгоритмов был сформирован код с генерацией тестового набора данных нарастающей размерности от 100 наблюдений и 100 предикторов до 2000 наблюдений и 2000 предикторов соответственно. Таким образом общее количество значений к обработке в наборе варьировалось от 10 тысяч до 4 миллионов. Тестовый алгоритм был построен в языковой среде Python с использованием одного из наиболее распространённых и популярных алгоритмов машинного обучения – XGBoost, основанного на градиентном бустинге деревьев решений [Chen и др., 2016; Guolin Ke и др., 2017; Воробьев, 2021].

В качестве основных мощностей для проверки были выбраны ЭВМ с идентичной или схожей операционной системой, CPU с датой выхода до 2014года и техпроцессом 22-32 нм (Windows 10 для Intel(R) Core(TM) i5-4210U и AMD FX(tm)-8320; Windows 10 СЕРВЕР для Intel(R) Xeon(R) E5-2660). Вычисления выполнялись исключительно на CPU.

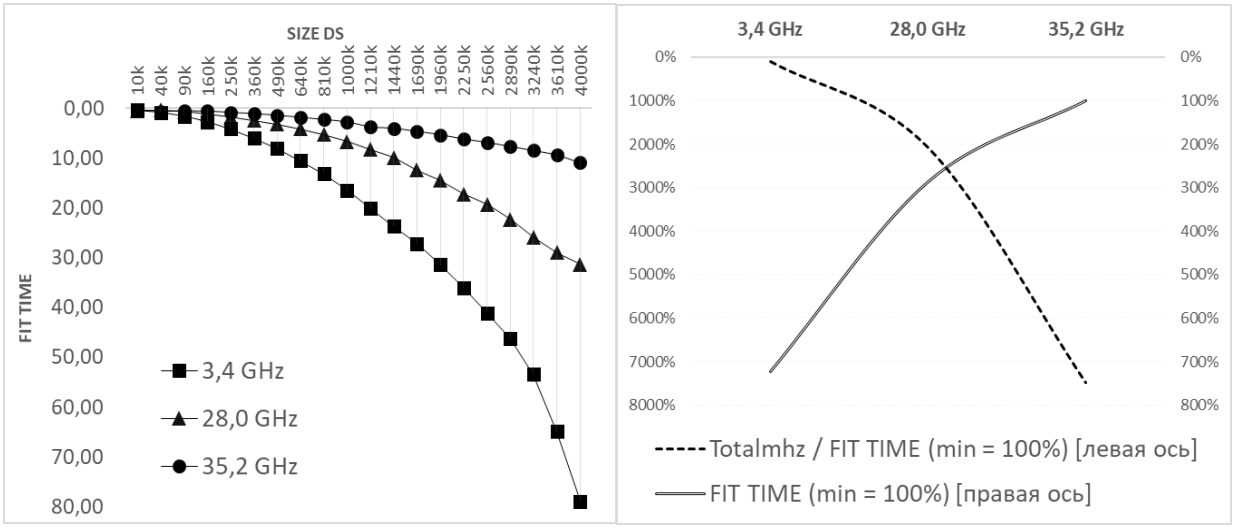
Для соблюдения возможности сопоставимости результатов в плоскости «время обработки – доступные вычислительные ресурсы», для представленных CPU была рассчитана условная общая частота по методике VMware (для хоста ESXi): соответствует произведению количества физических ядер, их тактовой частоте и количеству сокетов [Рекомендации по повышению производительности для VMware].

Таблица 1
Table 1

Ключевые характеристики CPU ЭВМ при проведении тестирования
Main technical details of PC's CPU in testing

Ключевые характеристики	Intel(R) Core(TM) i5-4210U	AMD FX(tm)-8320	Intel(R) Xeon(R) E5-2660
Количество физических ядер	2	8	8
Количество потоков	4	8	16
Базовая тактовая частота процессора (GHz)	1,7	3,5	2,2
Условная общая частота (GHz)	3,4	28	35,2

Полученные данные ожидаемо демонстрируют зависимость снижения скорости реализации алгоритма при увеличении размера доступных вычислительных ресурсов (рис. 1, а) [Hamotskyi и др., 2017; Kochura Y. и др., 2017; Holm и др., 2020]. При этом наблюдается изменение данной зависимости с линейной для CPU с относительно высоким показателем общей частоты на экспоненциальную для CPU с минимальным размером ресурсов.



а б

Рис. 1. Скорость выполнения ML-алгоритма в зависимости от размерности набора данных (а) и отношение доступных вычислительных ресурсов к скорости выполнения алгоритма в конечной точке наблюдения (б)

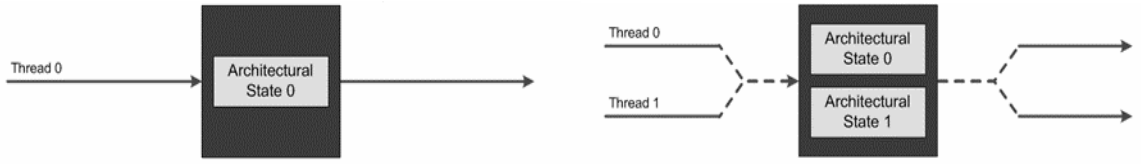
Fig. 1. The speed of ML-algorithm execution as a function of data set size (a) and the ratio of available computational resources to the algorithm's execution speed at the final point of observation (b)

Более выраженная диспропорция наблюдается при рассмотрении условного показателя «Totalmhz / FIT TIME» (рис. 1, б) в конечной точке наблюдения (набор данных размерностью 2000 х 2000), характеризующего отношение доступных вычислительных ресурсов к скорости выполнения алгоритма в данной точке наблюдения. Характер динамики данного показателя, в совокупности с рассмотрением относительного изменения скорости выполнения алгоритма свидетельствует о необходимости значительного наращивания размера доступных вычислительных ресурсов для повышения скорости выполнения тестового алгоритма. Так, исключительно в тестовых наборах увеличение скорости выполнения алгоритма на 1 % требует увеличения вычислительных ресурсов на 10 %.

Использование технологии hyper-threading в целях повышения скорости ML-алгоритмов на однопоточном приложении

Гиперпоточность – это технология Intel, которая обеспечивает второй набор регистров (т. е. второе архитектурное состояние) на одном физическом процессорном ядре. Это позволяет операционной системе или гипервизору компьютера получить доступ к двум логическим процессорам для каждого физического ядра системы (рис. 2). Гиперпоточность прошла долгий путь с тех пор, как она была впервые выпущена в 2002 году. Во многом это улучшение обусловлено улучшенной поддержкой гиперпотоков в операционных системах и гипервизорах [Witten и др., 2016; Zhen Jia и др., 2019].

Hyper-threading создает два логических процессора из одного физического процессорного ядра. Он делает это, предоставляя два набора регистров (называемых архитектурными состояниями) на каждом ядре. Когда в сокете Intel включена многопоточность, второе архитектурное состояние каждого ядра может принимать потоки от операционной системы (или гипервизора). Эти два потока по-прежнему будут совместно использовать внутренние компоненты микроархитектуры, называемые исполнительными блоками. Это может привести к увеличению производительности обработки до 30 % в одной системе сокетов. В системах с двумя разъемами гиперпоточность может обеспечить улучшение до 15 % [Sahil Munjal и др. 2014]



а б

Рис. 2. Доступ к логическим процессорам для каждого физического ядра системы без технологии Hyper-threading (а) и с учетом ее применения (б)

Fig. 2. Access to logical processors for each physical core of the system without Hyper-threading technology (a) and with its application (b)

Однако гиперпоточность не всегда способствует повышению производительности системы. В ряде случаев включение гиперпоточности может снизить ее производительность [Hassanein и др. 2008].

Гиперпоточность может привести к незначительному улучшению, отсутствию улучшения или снижению производительности системы в случаях:

– наличия большого количества физических ядер CPU,

– операционная система не поддерживает гиперпоточность (например: Windows Server 2003),

– приложение является однопоточным или не может эффективно обрабатывать несколько потоков,

– приложение уже разработано таким образом, чтобы максимально использовать исполнительные блоки в каждом ядре, или приложение имеет очень высокую скорость ввода-вывода памяти.

В целях определения влияния гиперпоточности на скорость выполнения ML-алгоритма в однопоточном приложении (Python) с использованием CPU со значительным количеством физических ядер (8), т. е. в случае нескольких ограничивающих использование гиперпоточности факторов, был проведен дополнительный тест на Intel(R) Xeon(R) E5-2660, демонстрирующим лучшие результаты скорости обработки алгоритма среди тестовых ЭВМ.

В проведенном эксперименте наблюдается разнонаправленное отклонение значений скорости выполнения обработки алгоритма. При размерности набора обрабатываемых данных до 1300 х 1300 более высокие показатели фиксируются при использовании доступных вычислительных ресурсов CPU без гиперпоточности – в среднем показатели лучше более чем на 1/3. С ростом объема тестового набора данных использование технологии hyper-threading позволяет улучшить результаты, линейно увеличивая их пропорционально растущему объему. При размерности набора свыше 1600 х 1600 диапазон улучшения варьируется от 10 % до 12 %, что близко к заявленным, по росту производительности, параметрам Intel.

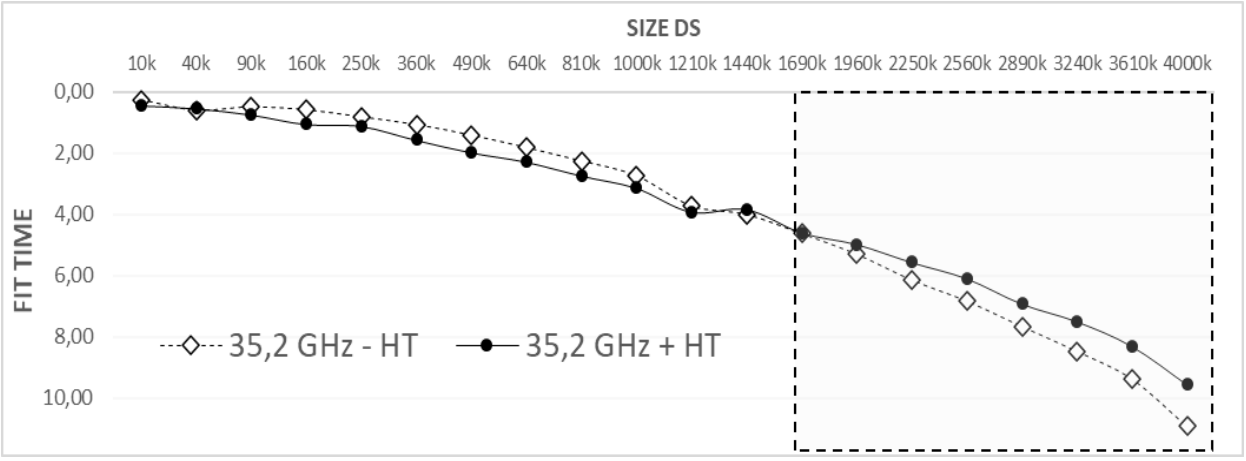


Рис. 3. Скорость выполнения ML-алгоритма в зависимости от размерности набора данных без технологии Hyper-threading (-HT) и с учетом ее применения (+HT)
Fig. 3. ML-algorithm execution speed as a function of data set size without Hyper-threading (-HT) and with Hyper-threading (+HT)

Таким образом, использование технологии hyper-threading допустимо при выполнении ML-алгоритмов в однопоточном приложении с допущением необходимости тестирования производительности системы в зависимости от методов обработки и объема набора данных.

Заключение

Рост производительности вычислительных мощностей не всегда приводит к пропорциональному снижению скорости обучения. При работе с ML-алгоритмами в однопоточной среде (Python) с использованием исключительно доступных вычислительных ресурсов CPU прирост скорости выполнения алгоритма требует значительно опережающего темпа приростов суммарной вычислительной мощности (до 1 к 10).

Использование гиперпоточности может повысить скорость обработки ML-алгоритмов в однопоточной среде до 10–15 %, при этом рост производительности может быть нелинеен и зависит от содержания и объема набора данных, используемых алгоритмов и методов обработки, количества потоков. Необходимо тестирование производительности системы с включенной гиперпоточностью и без нее для каждой частной задачи обработки ML-алгоритмов в целях определения оптимального значения количества потоков под конкретную размерность выбранного DS.

Список литературы

1. Воробьев А.В. 2021. Метод выбора модели машинного обучения на основе устойчивости предикторов с применением значения Шепли. Экономика. Информатика, 48 (2): 350–359. DOI 10.52575/2687-0932-2021-48-2-350-359.

2. Chen T., Guestrin C. 2016. XGBoost: A Scalable Tree Boosting System. arXiv:1603.02754. DOI: 10.1145/2939672.2939785.

3. Gordienko, Y. et al (2015). IMP Science Gateway: from the Portal to the Hub of Virtual Experimental Labs in e-Science and Multiscale Courses in e-Learning. Concurrency and Computation: Practice and Experience, 27(16), 4451–4464.

4. Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu: LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Advances in Neural Information Processing Systems 30 (NIPS 2017).

5. Hamotskyi, S., Rojbi, A., Stirenko, S., Gordienko, Y.: Automatized generation of alphabets of symbols for multimodal human computer interfaces. In: Proceedings of Federated Conference on Computer Science and Information Systems, FedCSIS-2017, Prague, Czech Republic (2017).

6. Håvard H. Holm, André R. Brodtkorb and Martin L. Sætra. GPU Computing with Python: Performance, Energy Efficiency and Usability. Computation. MDPI. 01/2020 Volume 8. https://doi.org/10.3390/computation8010004.

7. Kochura Y., Stirenko S., Alienin O., Novotarskiy M., Gordienko Y. Performance Analysis of Open Source Machine Learning Frameworks for Various Parameters in Single-Threaded and Multi-threaded Modes. Advances in Intelligent Systems and Computing II pp 243–256. DOI 10.1007/978-3-319-70581-1_17.

8. Larsen E.; McAllister D. Fast matrix multiplies using graphics hardware. In Proceedings of the 2001. ACM/IEEE Conference on Supercomputing, SC'01, Denver, CO, USA, 10–16 November 2001.

9. Performance Best Practices for VMware vSphere® 5.1. VMware, Inc. 3401 Hillview Ave. Palo Alto, CA 94304. Revision: 20120910. https://www.vmware.com/pdf/Perf_Best_Practices_vSphere5.1.pdf

10. Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (London, United Kingdom) (KDD '18). Association for Computing Machinery, New York, NY, USA, 974–983.https://doi.org/10.1145/3219819.3219890.

11. Sahil Munjal, Nikhil Singla, Nitin Sinha. Hyper-Threading Technology in Microprocessor. International Journal for Research in Applied Science & Engineering Technology (IJRASET). Volume 2 Issue X. 2014.

12. Seung Won Min, Kun Wu, Sitao Huang, Mert Hidayetoğlu, Jinjun Xiong, Eiman Ebrahimi, Deming Chen, Wen-mei Hwu. Large Graph Convolutional Network Training with GPU-Oriented Data Communication Architecture. arXiv:2103.03330 (2021).

13. Wessam M. Hassanein, Layali K. Rashid & Moustafa A. Hammad. Analyzing the Effects of Hyperthreading on the Performance of Data Management Systems. International Journal of Parallel Programming volume 36, pages 206–225 (2008) DOI:10.1007/s10766-007-0066-x.

14. Witten I. H., Frank E., Hall M. A., Pal C. J. (2016). Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann.

15. Zhen Jia, Wanling Gao, Yingjie Shi, Sally A. McKee, Zhenyan Ji, Jianfeng Zhan, Lei Wang, Lixin Zhang. Understanding Processors Design Decisions for Data Analytics in Homogeneous Data Centers. IEEE Transactions on Big Data. Volume: 5, Issue: 1.2019. DOI 10.1109/TBDATA.2017.2758792.

References

1. Vorobev A. 2021. Feautre stability based machine learning model selection method with usage of Shapley values. Economics. IT, 48 (2): 350–359. DOI 10.52575/2687-0932-2021-48-2-350-359.

2. Chen T., Guestrin C. 2016. XGBoost: A Scalable Tree Boosting System. arXiv:1603.02754. DOI: 10.1145/2939672.2939785.

3. Gordienko, Y. et al. 2015. IMP Science Gateway: from the Portal to the Hub of Virtual Experimental Labs in e-Science and Multiscale Courses in e-Learning. Concurrency and Computation: Practice and Experience, 27(16), 4451–4464.

4. Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu: LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Advances in Neural Information Processing Systems 30 (NIPS 2017).

5. Hamotskyi, S., Rojbi, A., Stirenko, S., Gordienko, Y. 2017. Automatized generation of alphabets of symbols for multimodal human computer interfaces. In: Proceedings of Federated Conference on Computer Science and Information Systems, FedCSIS-2017, Prague, Czech Republic.

6. Håvard H. Holm, André R. Brodtkorb and Martin L. Sætra. GPU Computing with Python: Performance, Energy Efficiency and Usability. Computation. MDPI. 01/2020 Volume 8. https://doi.org/10.3390/computation8010004

7. Kochura Y., Stirenko S., Alienin O., Novotarskiy M., Gordienko Y. Performance Analysis of Open Source Machine Learning Frameworks for Various Parameters in Single-Threaded and Multi-threaded Modes. Advances in Intelligent Systems and Computing II pp 243–256. DOI 10.1007/978-3-319-70581-1_17

8. Larsen E.; McAllister D. Fast matrix multiplies using graphics hardware. In Proceedings of the 2001. ACM/IEEE Conference on Supercomputing, SC'01, Denver, CO, USA, 10–16 November 2001.

9. Performance Best Practices for VMware vSphere® 5.1. VMware, Inc. 3401 Hillview Ave. Palo Alto, CA 94304. Revision: 20120910. https://www.vmware.com/pdf/Perf_Best_Practices_vSphere5.1.pdf

10. Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (London, United Kingdom) (KDD '18). Association for Computing Machinery, New York, NY, USA, 974–983.https://doi.org/10.1145/3219819.3219890

11. Sahil Munjal, Nikhil Singla, Nitin Sinha. Hyper-Threading Technology in Microprocessor. International Journal for Research in Applied Science & Engineering Technology (IJRASET). 2 (X). 2014.

12. Seung Won Min, Kun Wu, Sitao Huang, Mert Hidayetoğlu, Jinjun Xiong, Eiman Ebrahimi, Deming Chen, Wen-mei Hwu. Large Graph Convolutional Network Training with GPU-Oriented Data Communication Architecture. arXiv:2103.03330 (2021).

13. Wessam M. Hassanein, Layali K. Rashid & Moustafa A. Hammad. Analyzing the Effects of Hyperthreading on the Performance of Data Management Systems. International Journal of Parallel Programming volume 36, pages 206–225 (2008) DOI:10.1007/s10766-007-0066-x

14. Witten I. H., Frank E., Hall M. A., Pal C. J. (2016). Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann.

15. Zhen Jia, Wanling Gao, Yingjie Shi, Sally A. McKee, Zhenyan Ji, Jianfeng Zhan, Lei Wang, Lixin Zhang. Understanding Processors Design Decisions for Data Analytics in Homogeneous Data Centers. IEEE Transactions on Big Data. Volume: 5, Issue: 1.2019. DOI 10.1109/TBDATA.2017.2758792

Конфликт интересов: о потенциальном конфликте интересов не сообщалось.
Conflict of interest: no potential conflict of interest related to this article was reported.

ИНФОРМАЦИЯ ОБ АВТОРАХ

Воробьев Александр Викторович, аспирант кафедры ПОАИС Курского государственного университета, Курск, Россия

Распопин Даниил Игоревич, студент кафедры таможенного дела и мировой экономики факультета государственного управления и международных отношений Юго-западный государственный универ-ситет, Курск, Россия

INFORMATION ABOUT THE AUTHORS

Alexander V. Vorobyev, Postgraduate Cathedra of SISA, Kursk State University, Kursk, Russia

Daniil I. Raspopin, Student of the Department of Customs and Global Economy of South-West State University, Kursk, Russia