Application Performance Prediction in the Datacenter to Accelerate Cloud Servers

Gujia Gao*

1West Sound Academy, Poulsbo, WA 98370, USA

*Corresponding author’s e-mail: gjg_2018@sina.com

Abstract. Nowadays, cloud services become more and more important while datacenters are growing in scale. But the low effectiveness on cloud servers becomes an increasing issue as time goes. In this research, we focus on predicting application performance on cloud servers for acceleration. Linear regression and random forest are used in this study. Random foreset proves to be a precise, stable and effective tool to do performance counter prediction.

1. Introduction
Nowadays, cloud services become more and more important while large-scale datacenters are growing in scale. Cloud computing is serviced provided by datacenter through hardware or internet [21]. Recently there is a shift from complex monolithic services to simple graphic computing microservices in cloud computing [22]. Cloud Computing reduces the cost of deploying the service of hardware and operation and does not need to be actively managed by the user. It's also economic and efficient since the customer can rent as many machines as are needed without worrying about managing the machines and only need to pay for the amount of time they are using the machines.

Cloud computing includes public cloud, private cloud and hybrid cloud which are defined by the cloud propriety. Namely, public cloud computing resources are operated by a third-party cloud provider, private cloud by a single organization and hybrid both.

This paper will be focusing on the public cloud which is multi-talented, where one machine is shared by multiple applications. [23] The public cloud has the ability to release computing resources upon short term demand and retrieve computing and storage resources when they are not used. Computing resources releasing and retrieving may cause interferences between applications due to resource sharing, which will cause performance unpredictable.

These prompt the developments of interactive applications such as search, email and web serving. Profitability is determined by latency since the longer the consumer wait, the lower the probability the customer will continue waiting for the service to be accessible. In order to do that, determining which data to store and which data to drop is the key to the problem, which in this case, is to do the application prediction in the datacenter.

The contribution of this project is using machine learning to predict performance and to find out what is the most significant factors in performance degradation. The significance of this project is to provide a path to improve the status and examples distribution of cloud serving in an efficient way.
2. Background

To improve the performance prediction to speed up the server requires discovering the features in the CPU. A CPU is the electronic circuitry that runs different input and output (I/O) instructions given by the computer programs. The picture below is a brief overview of the CPU architecture.[8]

![Brief overview of the CPU architecture](image1)

Morden CPU has many cores where each runs instruction through multiple pipeline stages. A simple CPU pipeline as an example has five stages, as shown in the figure above. Instructions will be processed in one pipeline stage and forwarded to the next stage in each CPU cycle. Each pipeline stage can only handle one instruction at a time. It is shown in the pipeline diagrams below that with a pipeline, time waiting on blank cycles to get the stage needed is reduced which makes the processing more efficient. So using pipelines could enforce the transaction throughput but not the transaction latency [17].

![CPU processing diagrams](image2)

The main performance metric of a CPU is CPI(cycles per instruction). CPI depends on ISA and microarchitecture and the smaller it is, the better the computer performs. Modern CPU has 4-8 issue ports, which can issue 4-8 instructions simultaneously. For instance, an 8-issue CPU has the ideal CPI of 0.125 since 1 cycle divided by 8 instructions equals 0.125. So the ideal CPI of a one-issue pipelined CPU is 1 since that means it only takes one cycle for each instruction which means the processor is fully used and no hazard appears. But in most of the cases, the actual CPI is bigger than the ideal CPI due to pipeline stalling, squashing and memory accessing.

CPU caches data and instructions through its cache hierarchy. Typical cache hierarchy in the modern CPU includes three levels of caches, which stores the hot data from the upper-level storage for accelerating memory access. The L1 cache is the fastest since they are closest to CPU core, but it has rather small capacity. They usually split equally in size to separate data cache and instruction cache, namely L1D cache and L1I cache. L2 cache has a moderate size and access latency. L3 cache, which is
also named as the last level cache (LLC), is the largest cache with slowest computing speed on the CPU die. Usually, each CPU core has its own L1 and L2 caches, while L3 cache is usually shared with all CPU cores. If the data or instructions requested are found in caches, they will be read directly from the caches, otherwise, there will be a cache miss and the CPU need to read the data or instructions from upper-level caches or the main memory hierarchically. The existence of caches reduces the access latency of the information that the CPU most likely need next. On the other hand, a cache miss will slow down the processing speed by several orders of magnitude.

Branch instructions in the ISA are used to implement the control flow of programs, which redirect the program counter of the CPU to another instruction address. The existence of branch instructions will disrupt the sequential execution of the instructions in the CPU pipeline, resulting the CPU to stall for several cycles to resolve the actual instruction address, flush the instructions which are not supposed to be executed in the pipeline and restart the pipeline to execute the correct instructions. Modern CPUs use branch prediction to alleviate performance degradation due to branches. A branch predictor can predict if a conditional branch will be taken based on the history of this branch and other related branches to reduce the number of wasted cycles of executing incorrect instructions. A branch target buffer (BTB) caches the target instruction addresses of a number of frequently executed branches to prevent stalls due to resolving target instruction addresses.

Modern operating systems use the virtual address for better isolation and memory management for the programs. The mappings between physical addresses and virtual addresses are stored in a multi-level page table located in the main memory. If the data or instructions aren't be found in the last level cache, the process will look up the mapping of the physical address and virtual address by a page walk and then access the data from memory with the physical address. In order to accelerate the memory address translation, a translation lookaside buffer (TLB) located in the memory management unit on the CPU die, as the cache for address mapping, will be used for fast looking up the addresses of the frequently accessed pages. A miss in the TLB will cause multiple memory accesses before fetching the requested data, significantly degrade the CPU performance.

Performance degradation is affected by instruction cache miss, i-tlb(Instruction-TLB) miss from the front-end of the CPU and data cache miss, d-tlb(Data-TLB) miss from the backend of the CPU. Also, performance degradation happens when there’s a branch misprediction.

Some hardware performance counters below collected in the CPU are considered as features for predicting the performance of the server, shown in Table 1.[13]

| Feature name                | Feature description                                                                 |
|-----------------------------|--------------------------------------------------------------------------------------|
| instructions                | Number of instructions.                                                              |
| CPU-cycles                  | Number of instruction cycles.                                                        |
| branch-instructions         | Number of branch instructions that are used to reorder instruction execution.         |
| branch-misses               | The amount of branch misprediction.                                                   |
| frontend_retired.lli_miss   | Times of retired Instructions that Instruction L1 Cache miss.                         |
| l1d_pend_miss.pending       | Counts duration of L1D miss outstanding                                              |
| l2_rqsts.miss               | The number of requests that L2 cache miss.                                            |
| l2_rqsts.references         | Amount of L2 requests.                                                                |
| mem_load_retired.lli_hit    | Amount of retired load instructions that have at least one uop hit in the L3 cache.   |
| mem_load_retired.lli_miss   | Amount of retired load instructions that have at least one uop missed in the L3 cache. |
mem_inst_retired.all_loads
Amount of retired load instructions.
mem_inst_retired.all_stores
Amount of retired store instructions.
dtlb_load_misses.walk_pending
“Counts 1 per cycle for each PMH (Page Miss Handler) that is busy with a page walk for a load.”
dtlb_store_misses.walk_pending
“Counts 1 per cycle for each PMH that is busy with a page walk for a store.”
Itlb_misses.walk_pending
“Counts 1 per cycle for each PMH that is busy with a page walk for an instruction fetch request.”

2.1. Related work
In one previous study, forecasting network loads using auto-regressive linear predictor and artificial neural network shows that both algorithms predict effectively and the linear predictor has the highest accuracy.[20] Nevertheless, in this paper, the results indicate the instability of a linear predictor and its incapability while dealing with non-linear parameters. The result also shows that the random forest has higher accuracy on the performance counter. Although high accuracy could also be achieved by other neural networks, the random forest takes less time than these neural networks on the performance counter, so the efficiency is higher and uses less computing resources.[24][25]

3. Methods
In this experiment, linear regression and random forest regressor are used to predict the performance and figure out which ones are the important parameters.

3.1. Linear regression[16]:
An Algorithm that model the relationship between X and Y as a linear function, so that whenever we get a new X, we can predict its corresponding Y given N data point \{(X_i, Y_i) | i \in [1, N]\}.

The basic formula of linear regression is:

\[ y = X\beta + \varepsilon \] (1)

where

\[ y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad X = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ 1 & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ 1 & \cdots & x_{np} \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix} \] (2)

\( y \) is a vector of observed values \( y_i \) \( (i = 1, \ldots, n) \), \( \hat{y} \) is the predicted value. \( X \) is usually a constant that included as one of the regressors. \( \beta \) is the dimensional parameter vector. \( \varepsilon \) is the error term (noise) in the model.

Linear regression finds the linear relationship between the variables quickly and is intuitive and easy to use and understand since each variable’s relationship with the final result could be described by the coefficient. However, linear regression only models linear relationships which is very sensitive to the anomalies. And if there are too many parameters compared to the samples, then the model would not model the relationships between the variables, instead, it would model the noise.[14]

3.2. Random forest [9]:
A random forest is a collection of decision trees (also known as tree predictors or tree-structured classifiers). Each tree has the same distribution as others and depends on the sampled random values. Decision tree always picks the attributes that best separate the data. The results of the decision trees are
aggregated into one final result: a greedy solution which always picks the attributes that best separates the data.

To build a random forest, take \(n_{\text{tree}}\) bootstrap samples from the data first and grow an unpruned regression tree for each bootstrap sample. Then, at each node choose the best split among all predictors. Take a random sample \(m_{\text{try}}\) of the predictors and find the best split among their variables. (Bagging is when \(m_{\text{try}} = p\), the number of predictors.) After that, aggregate the predictions of the \(n_{\text{tree}}\) trees to predict test data.

So if there is a set of \(n\) trees with individual weight functions \(W_j\), the prediction would be

\[
\hat{y} = \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{n} W_j(x_i, x') y_i = \sum_{i=1}^{n} \left( \frac{1}{m} \sum_{j=1}^{m} W_j(x_i, x') \right) y_i.
\]

where \(x'\)'s neighbors are points \(x_i\) that shares the leaf with any tree \(j\). And the neighbors of this interpretation are the points sharing the same leaf in any tree \(j\). \(i\)-th training point that is relative to \(x\) has the non-negative weight in the same tree \(W(x_i, x')\).

Random forests regression could limit overfitting through generalizing errors of a forest of tree classifiers. The strength of the decision trees and the correlation between them affect the performance of generalization error. Training on different samples also reduces variance. These features would not increase bias errors but give useful internal estimates and be robust to noise and outliers. The algorithm has relatively high speed and accuracy too. But the results of random forests are not easy to interpret visually. Data with correlated features of similar relevance for the output even weighted more than larger groups. The higher-level categorical variable in data is more biased by the random forest than the lower-level ones. This makes the random forest unreliable sometimes. Also, random forests may be overfitting it there are too many noises in the data. [12]

4. Evaluation

| Machine parts       | Machine configuration       |
|---------------------|----------------------------|
| CPU                 | Intel Xeon E2699-v4        |
| Sockets             | 2                          |
| Cores/socket        | 22                         |
| Threads/core        | 2                          |
| Frequency           | 2.2GHz                     |
| L1 Inst/Data Cache  | 32/32KB                    |
| L2 Cache            | 256KB                      |
| L3 Cache            | 55MB, 20 ways              |
| Cache line          | 64B                        |
| uTLB                | 64 entries                 |
| MTLB                | 1,536 entries              |
| Memory              | 128GB, DDR4 2666MHz        |

I used Cloudsuite 3.0 for evaluating the accuracy of the performance prediction using hardware performance counters. CloudSuite is a benchmark suite for cloud services.[4] Many companies like Lee Memorial Health and URM Stores Inc. applies Cloudsuite in the datacenter [18] Cloudsuite contains eight applications. But the focus of the paper is on three of them: graph analytics, In-memory analytics, and web serving.
- **Graphic analytics:** It contains the docker images for Graph Analytics benchmark which relies on the Spark framework and performs PageRank on a Twitter dataset.
- **In-memory analytics:** This benchmark uses Apache Spark to run on datasets of user-movie rating through a collaborative filtering algorithm in-memory. The algorithm it runs is the alternating least squares (ALS). The benchmark is for predicting the 'rating' and 'preference' that the audience would give to a film. The outputs are the time in seconds of computing movie recommendations.
- **Web serving:** The benchmark has three layers where each has its own labeled images: the web server, the data-caching server, and the database server. The web server is connected with the data-caching server and the database server. The clients send requests. In this experiment, Nginx is used as a web server. Memcached is used for data caching. MongoDB is used as a database server.
  - **Web server:** It’s a computer that sets up in data centers and runs Nginx which can be used as a load balancer, reverse proxy, HTTP cache, and mail proxy. Web server transplants dynamic and static content into the cloud to improve fault-tolerance and dynamic scalability by providing a load balancer's required servers amount. Its function is to access static files cached in data caching layer. If the static files are cached, the web server retrieves the data. Otherwise, the web server would find the data in the database and store it in Memcached as well.
  - **Data caching:** Memcached, a large scale distributed in-memory service, used by a lot of large companies (Twitter, Amazon, Facebook, Netflix), is used in data caching. The function of data caching is to cache popular data.
  - **Database:** It uses MongoDB which is a cross-platform document-oriented database program. Also, MongoDB is one of the most Popular NoSQL database programs.

All the data are used to do data analytics which in this case is the performance counter prediction where codes are written using linear regression and Random Forest (RF-model) to figure out the relationships between the parameters of the data and CPU-cycles. The procedure of applying the algorithm is to train them to predict the parameters of the performance on the data of performance counter without knowing some key parameters like “CPU-cycles”. The data sets are randomly distributed based on 90% are the training data and 10% are the test data. The algorithms would also tell the coefficients or feature importance of the parameters so that we know their effects towards the performance. After that, we compare the actual performance to the prediction to see the accuracy of the prediction.

5. Results
Since two regression models are run in this study, we display the results in the following several tables. Regression results are shown in Table 3, 4, 5. It is shown in the data that linear regression can do performance counter prediction with unstable accuracy since the coefficient of determination fluxes a lot on each test. The only parameters which have P values that are bigger 0.05 are ’mem_inst_retired.all_stores’ and ’itlb_misses.walk_pending’, meaning the rest are all significant. But this doesn’t give out much information. Anomalies often appear in linear regression test too. It is also hard to find rationales through the data. Take the parameter ‘mem_load_retired.l3 miss’ as an example, it has a large negative coefficient which seems that it has a positive effect on the performance but it does not make sense since cache miss always slows down the processor. Nonetheless, the mean squared root, which is given by the following formula, is relatively low (0.0578) which means that linear regression does great on predicting. The reason why linear regression is not stable in prediction is that there is a lot of non-linear relationship parameters which linear regression could not capture.

\[
MSE = \frac{1}{n} \cdot \sum_{i=1}^{n} (y_i - y_0')^2
\]

where \(y_i = Actual\ performance\ result, y_0' = Prediction, n = Number\ of\ examples [20].

Table 3. linear regression results
| R squared in training data | R squared in testing data |
|---------------------------|--------------------------|
| 0.921                     | 0.88                     |
| 0.927                     | 0.059                    |
| 0.919                     | 0.921                    |
| 0.921                     | 0.901                    |
| 0.935                     | 0.303                    |
| 0.921                     | 0.899                    |
| 0.926                     | 0.78                     |
| 0.92                      | 0.905                    |
| 0.922                     | 0.891                    |
| 0.918                     | 0.918                    |

Average of R squared: 0.89
Variance of R squared: 0.00431
Average time used (seconds): 0.09066

Table 4. Comparison of actual and prediction results

| Prediction | Actual performance | Prediction | Actual performance |
|------------|--------------------|------------|--------------------|
| 1.551      | 1.578              | 1.374       | 1.319              |
| 0.787      | 0.983              | 2.256       | 2.429              |
| 2.452      | 2.355              | 0.973       | 0.872              |
| 2.238      | 2.384              | 1.271       | 1.37               |
| 2.429      | 2.453              | 2.61        | 2.381              |
| 1.204      | 1.066              | 3.528       | 2.602              |
| 2.459      | 2.443              | 1.007       | 0.79               |
| 0.909      | 0.766              | 1.018       | 0.851              |
| 4.048      | 4.443              | 1.291       | 1.305              |
| 2.596      | 2.89               | 2.139       | 2.246              |

Table 5. Coefficient of each feature in the linear model

| Feature                                    | Coefficient | Standard error |
|--------------------------------------------|-------------|----------------|
| branch-instructions                         | 62.84       | 1.36           |
| l2_rqsts.miss                              | 19.099      | 3.68           |
| dtlb_load_misses.walk_pending              | 15.87       | 1.034          |
| l2_rqsts.references                        | 12.4        | 1.25           |
| mem_inst_retired.all_stores                | 3.76        | 1.93           |
| l1d_pend_miss.pending                      | 2.3         | 0.033          |
| itlb_misses.walk_pending                   | -0.84       | 2.56           |
| mem_inst_retired.all_loads                 | -2.9        | 0.34           |
| mem-stores                                 | -5.73       | 1.14           |
| dtlb_store_misses.walk_pending             | -51.45      | 21.76          |
| frontend_retired.l1i_miss                  | -111.23     | 22.089         |
| branch-misses                              | -183.61     | 8.045          |
| mem_load_retired.l3_hit                    | -216.046    | 47.91          |
| mem_load_retired.l3_miss                   | -516.31     | 87.45          |
The results of random forest models are shown in Table 6, 7, 8. Feature importance in the random forest shows the weights of each parameter. The sum of all the parameters’ feature importance is one. These performance counter parameters slow the processor, so the greater feature importance it has, the larger negative influence it has. It is shown in the data that random forests can do performance counter prediction with higher accuracy than linear regression since Random forest has smaller MSE (0.006725231). Random forest is overall more accurate compared to linear regression since random forest is able to catch nonlinear relationships between input variables x and output variable y. The features which affect the performance most are the number of branch instructions, last level cache misses, L1 data cache misses and branch instruction misses according to the feature importance. (The sequence is from the most important one to the least important one.)

The number of branch instructions(branch-instructions) is the most influential feature since every taken branch requires installing pipelines. Although branch target predictor(BTB) could reduce branches by predicting whether the branch is conditional or not before being computed by the execution unit of the processor, the size of BTB is limited. Because of the size, BTB cannot cache every target address of branches. [26] So, unavoidably, branches are frequently executed and result in large latency.

Last level cache misses(mem_load_retired.l3_miss) influence the performance a lot since once a cache miss on the last level, it has to find the data in the memory which is out of the cache hierarchy and takes way more computing resources. The off-chip memory latency is enormous compared to CPU cycles.

L1 data cache misses(l1d_pend_miss.pending) affect the performance quite remarkable because it is the number of cache misses in the first level cache. L1 is the smallest among the data caches and stores the hottest data which means that cache miss happens really frequently.

Branch instruction misses(branch-misses) cause a bunch of latency since the number of branch instructions is large and branch misprediction happens often.

### Table 6. Random forest results

| R squared in training data | R squared in testing data |
|---------------------------|---------------------------|
| 0.996                     | 0.980                     |
| 0.997                     | 0.989                     |
| 0.993                     | 0.969                     |
| 0.996                     | 0.968                     |
| 0.998                     | 0.985                     |
| 0.998                     | 0.982                     |
| 0.997                     | 0.976                     |
| 0.996                     | 0.980                     |
| 0.997                     | 0.989                     |
| 0.997                     | 0.974                     |

Average of R squared: 0.985
Variance of R squared: 0.0000716
Average time used (seconds): 0.16895

### Table 7. Comparison of actual and prediction results in random forest model

| Prediction | Actual performance | Prediction | Actual performance |
|------------|--------------------|------------|--------------------|
| 2.494      | 2.553              | 1.874      | 1.762              |
| 1.0128     | 1.0155             | 2.408      | 2.372              |
| 2.418      | 2.474              | 1.531      | 1.365              |
| 2.292      | 2.269              | 2.244      | 2.147              |
| 1.335      | 1.326              | 2.582      | 2.833              |
| Feature | Feature importance |
|---------|--------------------|
| branch-instructions | 0.286 |
| mem_load_retired.l3_miss | 0.244 |
| l1d_pend_miss.pending | 0.179 |
| branch-misses | 0.0933 |
| itlb_misses.walk_pending | 0.058 |
| l2_rqsts.references | 0.0466 |
| mem-stores | 0.0333 |
| mem_inst_retired.all_stores | 0.022 |
| frontend_retired.l1i_miss | 0.0183 |
| dtlb_load_misses.walk_pending | 0.00863 |
| l2_rqsts.miss | 0.00503 |
| dtlb_store_misses.walk_pending | 0.0034 |
| mem_load_retired.l3_hit | 0.00183 |
| mem_inst_retired.all_loads | 0.00112 |

By comparing linear regression and random forest, random forest has less standard deviation than linear regression which means it is more stable. Also, random forest has greater coefficient of determination than linear regression which means random forest has higher accuracy too. However, linear regression takes less time to train on data than random forest.

Figure 3. Comparison of linear regression and random forest

6. Results
Only linear regression and random forest are used since they already have high accuracy on the data, less possible on overfitting and they are faster on training and clearer on comparing the weight of the
different parameters. From the experiment, random forest is proved to be a precise, stable and effective tool to do performance counter prediction.

The features which affect the performance most is the number of branch instructions(branch-instructions), last level cache misses(mem_load_retired.l3_miss), L1 data cache misses(l1d_pend_miss.pending) and branch instruction misses(branch-misses). The reason why the number of branch instructions is the most influential feature is that there is only limited BTB and every taken branch requires installing pipelines. Due to the off-chip memory latency is enormous compared to CPU cycles, last level cache misses(mem_load_retired.l3_miss) is also one of the main factors of slowing the server. L1 data cache misses(l1d_pend_miss.pending) affect the performance quite a bit because cache misses in L1 happens really frequently. Branch instruction misses(branch-misses) cause much latency since the number of branch instructions is large and branch misprediction happens often.

The defects of the experiment are that there are still a lot of algorithms that were not used to examine if they could predict better such as SVM or other more advanced neural networks. The outlook of the paper is that when companies want to speed up cloud servers by adjusting idle resources, they could focus on investigating, adjusting and ameliorating the four most important features the paper discovered.

References
[1] Batten, C. ECE 4750 Computer Architecture, Fall 2018 T01 Fundamental Processor Concepts.
[2] Shen, Y., Ferdman, M., Tapti P., (2016), Demystifying Cloud Benchmarking. In: 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).
[3] Niao, G. (2010) Linux Private Kitchen Basis Learning Part. Posts & Telecom Press, Beijing
[4] Ferdman, M., Adileh, A., Kocberber, O., Volos, S., Alisafee, M., Jevdijc, D., Kaynak, C., Popescu, A.D., Ailamaki, A., Falsafi B., (2012) Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In: 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).
[5] What is Web server? - Definition from WhatIs.com Margaret Rouse, March 2019.
[6] Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., ... & Stoica, I. (2009). Above the clouds: A berkeley view of cloud computing. Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, 28(13), 2009.
[7] Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., ... & Hu, K. (2019). An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 3-18). ACM.
[8] 'Architecture of the central processing unit (CPU)- Computer Science Wiki' 2019. https://computersciencewiki.org/index.php/Architecture_of_the_central_processing_unit_(CPU)
[9] Liaw, A., Wiener, M. (2002). Classification and regression by randomForest. R news, 2(3), 18-22.
[10] Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.
[11] Patterson, D.A.; Hennessy, J.L. Computer Organization and Design: The Hardware/Software Interface.
[12] Ctufits, Tufts.(2015).Classification Model Pros and Cons. https://github.com/ctufits/Cheat_Sheets/wiki/Classi fiction-Model-Pros-and-Cons.
[13] Intel® Microarchitectur Code Named Skylake Events. n.d.https://download.01.org/perfmon/index/skylake.html
[14] Darshan Jain. (2018) What are the advantages and disadvantages of linear regression? - Quora. https://www.quora.com/What-are-the-advantages-and-disadvantages-of-linear-regression
[15] Lin, Y.; Jeon, Y. (2002). Random forests and adaptive nearest neighbors. Technical Report No. 1055. University of Wisconsin.

[16] Yan, X., Su, X. (2009). Linear regression analysis: theory and computing. World Scientific.

[17] Batten, C. ECE 4750 Computer Architecture, Fall 2018 T02 Fundamental Processor Microarchitecture.

[18] Companies Using Infor CloudSuite HCM, Market Share, Customers and Competitors n.d., viewed 20 June 2019, https://discovery.hgdata.com/product/infor-cloudsuite-hcm.

[19] Stackoverflow https://stackoverflow.com/questions/44495667/calculate-p-value-in-sklearn-using-python

[20] Prevost, J. J., Nagothu, K., Kelley, B., Jamshidi, M. (2011). Prediction of cloud data center networks loads using stochastic and neural models. In: 6th International Conference on System of Systems Engineering (pp. 276-281). IEEE.

[21] Josep, A.D., Katz, R., Konwinski, A., Gunho, L.E.E., Patterson, D., Rabkin, A. (2010). A view of cloud computing. Communications of the ACM, 53(4).

[22] Newman, Sam. Building microservices: designing fine-grained systems. " O'Reilly Media, Inc.", 2015.

[23] Mars, J., et al. (2011) Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture. ACM.

[24] Yuan, J., Yu, S. (2013). Privacy preserving back-propagation neural network learning made practical with cloud computing. IEEE Transactions on Parallel and Distributed Systems, 25(1), 212-221.

[25] Duy, T.V.T., Sato, Y., Inoguchi, Y. (2010). Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In: IEEE international symposium on parallel & distributed processing, workshops and Phd forum (IPDPSW) (pp. 1-8).

[26] Fog, A. (2016) The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers, http://www. agner. org/optimize/microarchitecture. Pdf.

[27] Chen, J., Li, K., Tang, Z., Bilal, K., Yu, S., Weng, C., Li, K. (2016). 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.