Performance of Raspberry Pi microclusters for Edge Machine Learning in Tourism

Andreas Komninos Ioulia Simou Nikolaos Gkorgkolis John Garofalakis

[akomninos, simo, gkorgkolis, garofala]@ceid.upatras.gr
Computer Technology Institute and Press "Diophantus", Rio, Patras, 26504, Greece

Abstract

While a range of computing equipment has been developed or proposed for use to solve machine learning problems in edge computing, one of the least-explored options is the use of clusters of low-resource devices, such as the Raspberry Pi. Although such hardware configurations have been discussed in the past, their performance for ML tasks remains unexplored. In this paper, we discuss the performance of a Raspberry Pi micro-cluster, configured with industry-standard platforms, using Hadoop for distributed file storage and Spark for machine learning. Using the latest Raspberry Pi 4 model (quad core 1.5GHz, 4Gb RAM), we find encouraging results for use of such micro-clusters both for local training of ML models and execution of ML-based predictions. Our aim is to use such computing resources in a distributed architecture to serve tourism applications through the analysis of big data.

CEUR-WS.org/Vol-2492/paper2.pdf

1 Introduction

The rise of machine learning (ML) applications has led to a sharp increase in research and industrial interest in the topic. Paired with the increase of Internet of Things (IoT) deployments, ML is used to deliver batch (off-line) and real-time (streaming) processing of big data, to serve a variety of purposes, including real-time data analytics, recommender systems, forecasting via regressors and classification of numerical, textual and image data [LOD]. Typical deployments involve a distributed architecture, where remote nodes submit data to a central analysis and storage system, which in turn either stores the data for future processing, or responds by returning ML results to the contributing nodes. These central storage and processing systems are sometimes physically co-located, but often are distributed themselves across various datacenters, in a cloud computing configuration, particularly in large scale systems. As data is gathered centrally, remote clients benefit from ML results based on the contribution of all nodes in a system, but suffer from issues such as network latency (which is important for time-critical applications) and reliability (since the central repository becomes a single point of failure in the system). More recently, the concept of edge computing has sought to address some of these problems, by further distributing the storage and processing capabilities of the system, to nodes closer to the end-user [APZ]. These nodes are not as resource-rich as cloud computing datacenters, but are generally more capable than typical IoT devices. By relocating the storage and ML model execution closer to end users, the system becomes more responsive and is more resilient, though one associated drawback is that edge nodes cannot store as much data and thus cannot derive highly accurate models, compared to cloud-computing setups.

Copyright © by the paper's authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

In: E. Calvanese Strinati, D. Charitos, I. Chatzigiannakis, P. Ciampolini, F. Cuomo, P. Di Lorenzo, D. Gavalas, S. Hanke, A. Komninos, G. Mylonas (eds.): Proceedings of the Poster and Workshop Sessions of AmI-2019, the 2019 European Conference on Ambient Intelligence, Rome, Italy, 04-Nov-2019, published at http://ceur-ws.org

Interest in edge deployments is strong, since the drawbacks can be mitigated using alternative deployment approaches. For example, edge nodes can forward data to a cloud server so that complex and powerful ML models can be built. These models can be saved and distributed back to the edge nodes for use. Additionally, edge nodes can pre-process data locally before forwarding them to cloud servers, helping with the distribution of data cleansing and transformation workloads. Edge computing architectures also lend themselves particularly well to certain types of application, where the users might be more interested in locally pertinent data. Edge computing hardware form factors vary, ranging from server-class hardware, to simple desktop-class computers, and even IoT-class devices (e.g. Arduino, Raspberry Pi) can perform edge computing roles. Recently, dedicated edge nodes for ML have been developed (e.g. Google Coral, Nvidia Jetson, Intel Neural Compute), and hardware accelerator add-ons for existing platforms are also on the market (e.g. Raspberry PI AI hat, Intel 101).

One interesting configuration which has not gained much attention, is the ability of IoT devices running Linux operating systems, to work together in a cluster computing configuration. This ability leverages many of the known advantages of cloud computing (e.g. using a distributed file system such as HDFS, running big data analytics engines such as Spark), providing a scalable solution for powerful and resilient local edge components, while keeping deployment costs low. In this paper, we explore the performance of small Raspberry Pi (RPi) clusters in the role of an IoT ML edge server, using the RPi-4 model, which is the latest release in this product line. Although Pi clusters have been reported in previous literature, the RPi-4 model is newly released (Q2 2019) and its significant hardware improvements that make it a more realistic option for this role than before.

2 Related Work

Table 1: Raspberry Pi models

	Model 1B	Model 2B	Model 3B	Model 4B
Cores	1	4	4	4
CPU Clock (GHz)	0.7	0.9	1.2-1.4	1.5
RAM (Gb)	0.5	1	1	1-4
Network (Mbps)	100	100	100	1000

The performance of Raspberry Pi clusters has been investigated in the past literature, mostly using Model 1B an 2B devices (see Table 1). The first known paper to report findings on such a deployment is [TWJ$^+$], with a configuration of 56 Model-B RPis. No specific performance evaluations were reported but the advantages of low-cost and the ability to use such clusters for research and educational purposes were highlighted in this paper. An even larger Model-B cluster (300 nodes) is reported in [AHP$^+$], although again no performance evaluation is discussed. A smaller Model-B cluster (64 nodes) is discussed in [CCB$^+$], demonstrating that network bandwidth and limited memory are barriers for such clusters. The performance advantages in computing power depend on the size of the computational task, with smaller problems not benefiting from additional computing resources (nodes) due to communication overheads, and memory size limiting the size of the computational task that can be performed. Similar results demonstrating the computational performance drop from the theoretical linear increase line as nodes are added, are obtained by [CPW] using a 24 unit Model 2B cluster, and also in a 12-node Model 2B [HT] and an 8-node Model 2B cluster [MEM$^+$]. In [KS] the performance using external SSD storage (attached via USB) is evaluated, demonstrating that big data applications on such clusters (20 x Model 2B) is bound by the CPU limitations. In [QJK$^+$], a 20-node RPi Model 2B cluster is investigated for real-time image analysis. Its performance was lower than virtual machines running on traditional PCs, however, the small form factor paired with the relatively good performance, make such clusters ideal for mobile application scenarios.

With regard to RPi cluster performance in the execution of ML algorithms, [SGS$^+$] describe the performance of an 8-node Model 2B cluster. The researchers concluded that RPi clusters offer good tradeoffs between energy consumption and execution time, though better support for parallel execution is needed to improve performance. Such optimisations are demonstrated in [CBLM], with the 28-node Model 2B cluster outperforming a traditional multicore server, in terms of processing power and power consumption, using 12 nodes only.

RPi clusters have been proposed for use in educational settings, to teach distributed computing [DZ, PD, Tot], as servers for container-based applications [PHM$^+$] and as research instruments, e.g. to analyse data from social networks [dABV] or in security research [DAP$^+$].

Overall, while significant interest in the research community has been shown towards RPi clusters, there is presently no work demonstrating their performance in ML roles except in [SGS+]. Hence, our goal for this paper is to investigate the performance of RPi clusters in an edge ML role, using the latest RPi Model 4B model which overcomes some of the previous network and memory constraints.

3 RPi cluster configuration

Our cluster consists of 6 RPi4 Model B devices (Fig. 1), with 4Gb RAM available on each node. Additionally, the devices were equipped with a 64Gb MicroSD card with a V30 write speed rating (30Mb/s). The devices were connected to a Gigabit ethernet switch to take full advantage of the speed improvements in the network card. The cluster was configured with a Hadoop distributed file system (3.2.0) and Apache Spark (2.4.3). As such, we are able to leverage Spark's MLlib algorithms for distributed machine learning.

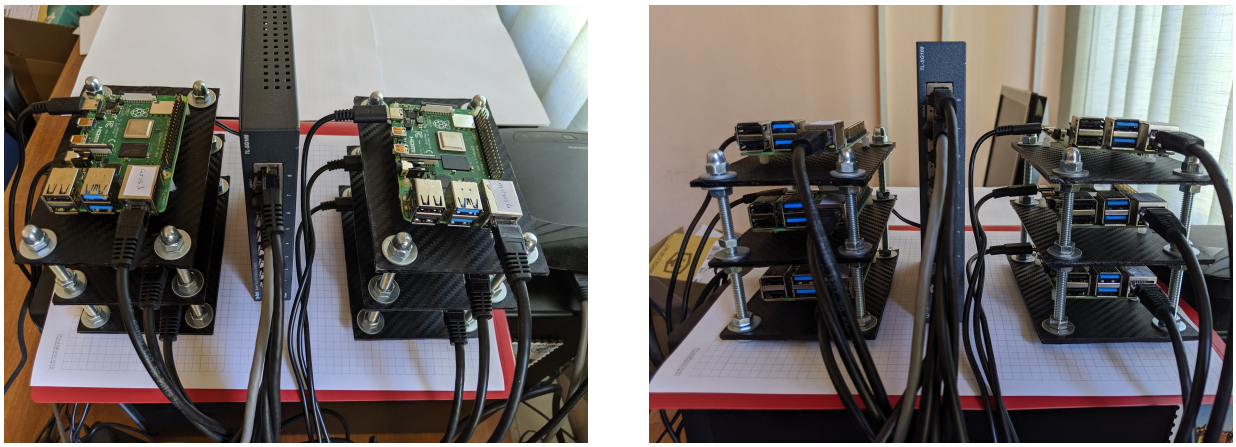


Figure 1: The RPi micro-cluster on 5mm grid paper (A4) demonstrating physical dimensions

3.1 Hadoop & Spark execution environment configuration

For the distributed file storage, we set the number of file replicas to 4. Since Hadoop is installed in the cluster, we use the YARN resource allocator for the execution of Spark jobs. Each YARN container was configured with 1920Mb of RAM (1.5Gb + 384Mb overhead), leaving 1Gb available for YARN master execution and 1Gb of RAM for the operating system (Raspbian 10 - Buster). Although the cluster can be configured to run with multiple Spark executors on each device, we opted for a "fat executor" strategy, meaning one executor per device. Additionally, as a baseline configuration scenario (*S-base*), we opted for reserving one processor core on each device for use by the operating system, thus resulting in 3 cores per executor. Jobs were submitted from within the cluster, therefore one device always played the role of the client, one device played the role of the application manager (YARN), thus up to 4 devices were available as executors, in order to run the Spark jobs.

3.2 Datasets and ML algorithms

We used two datasets for our experiments. First, we perform experiments using a large dataset, in this case the used car classifieds dataset[1]. Secondly, since we aim to apply the RPi cluster in a tourism recommender system for Greece, as part of an ongoing project, we used the global scale checkins dataset [YZQ] and the Greek weather dataset[2]. From the former, we selected all check-ins made in the Attica region of Greece, and we fused the resulting data with the historical weather information from the latter dataset. Both datasets were uploaded to the Hadoop distributed file system in the cluster.

The used cars dataset was used to perform simple linear regression, using car model year and odometer reading, to predict its sale price. The check-ins dataset was used with the decision tree classification algorithm. In this case, by providing the geographical coordinates, month, day, hour and daily mean, high and low temperatures, the target was to determine the type of venue a user might check into (a multi-class classification task). This can be used to recommend types of venue that a user might like to visit depending on their current context.

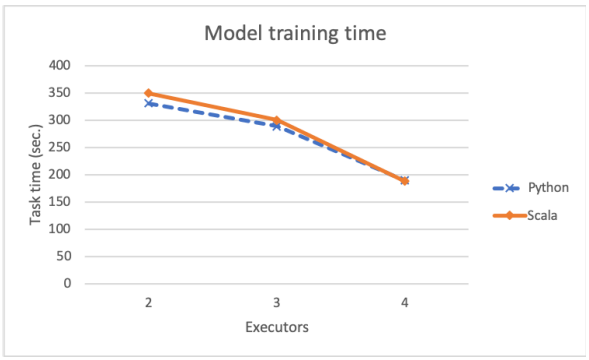
[1] https://www.kaggle.com/austinreese/craigslist-carstrucks-data
[2] https://www.kaggle.com/spirospolitis/greek-weather-data

4 Experiment 1 - programming language performance

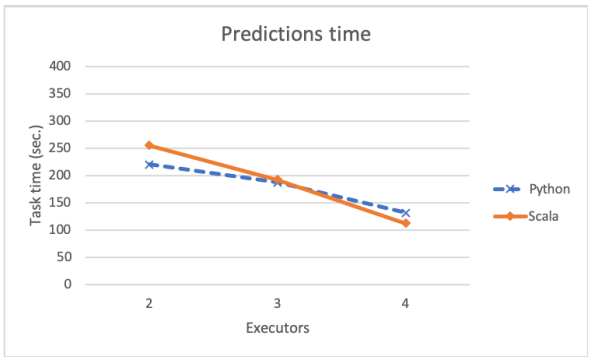
Apache Spark provides programming interfaces for the Python language, popular amongst ML developers, and also uses Scala natively. Since Python is a dynamically typed and interpreted language, where Scala is statically typed and compiled, Scala should provide a performance advantage for applications in our cluster, however, this depends on the type and volume of data used. In this first experiment, we compare the performance of a simple application written in both languages. The application includes the following tasks in sequence:

1. Load a dataset from HDFS into Spark Dataframe
2. Select feature and label columns
3. Filter out samples with NULL values
4. Assemble feature columns into a single feature vector and append to dataset
5. Split dataset to training and test datasets (0.7, 0.3)
6. Train a machine learning model
7. Perform predictions on the test dataset

For this first experiment, we used the cars dataset to train a linear regression model. As a result of the data cleansing process, the final dataset contained 9,585,316 samples. We measured the time taken to complete the data loading, the model training and predictions tasks, as shown in Fig. 2. From these results we can see that the Scala program executes more slowly when the number of executors is small, however, it achieves parity or even outperforms Python in execution time as the number of executors increases. Based on these results, we chose to proceed with the rest of the experimentation using Scala as the programming language.



(a) Model training



(b) Predictions on test set

Figure 2: Programming language performance

5 Experiment 2 - Performance in ML tasks

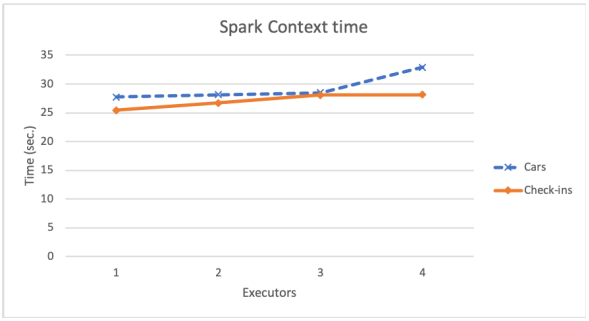
Next, we implemented two ML algorithms to assess the cluster's performance. In this case, we were interested in determining how the number of executors and number of cores per executor affect performance, in the implementation of different ML algorithms. As such, we retain the baseline configuration scenario *S-base* (fixed $n_{cores/executor} = 3$, variable $n_{executors} \in [1, 4]$) and add a further scenario (fixed $n_{executors} = 4$, variable $n_{cores/executor} \in [1, 4]$). This scenario is termed *S-core* henceforth. Note that, despite standard practice, in S-core we allocate up to 4 cores (the maximum available), to investigate the full resource utilisation contesting the OS requirements. The sequence of tasks was identical to the previous experiment, changing of course the type of model to be trained. Additionally, we implemented three extra steps:

8. Write the trained model to distributed storage
9. Load the pre-trained model from distributed storage
10. Perform a single prediction given a random feature vector from the test set

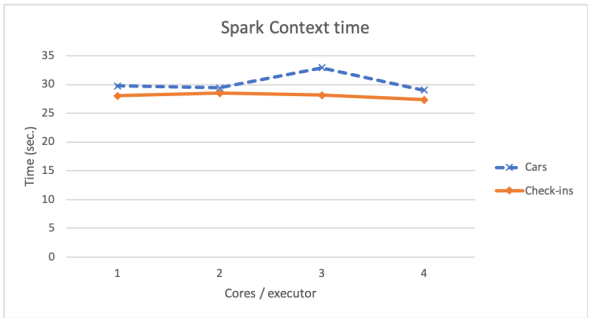
These additional tasks emulate the concept of batch training at regular intervals on the edge node, and using the pre-trained models for application purposes. For the second experiment, we used both datasets. We note that as a result of the data pre-processing task, the final check-ins dataset contains 87,908 samples.

5.1 Application startup overhead

First, we measured the overhead time taken to obtain the necessary SparkContext environment (i.e. assigning a YARN application master and attaching executor nodes to the process). From Fig. 3 we note that the required overhead fluctuates but remains roughly constant in all cases (S-base cars : $\mu = 29.322s, \sigma = 2.419s$, check-ins $\mu = 27.089s, \sigma = 1.278s$; S-core cars : $\mu = 30.271s, \sigma = 1.791s$, check-ins $\mu = 28.013s, \sigma = 0.466s$). Of course this overhead is required only at application startup and is not incurred for every request, when the application is written as a server waiting to receive ML result requests.



(a) Using 3 cores / executor



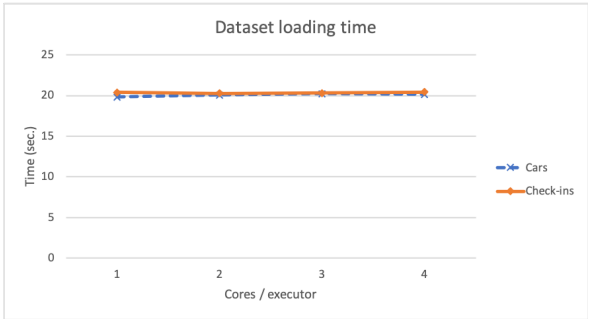
(b) Using 4 executors

Figure 3: Spark context overhead

5.2 Data retrieval from HDFS

Another metric is the time taken to load the dataset from HDFS storage, as a Spark Dataframe. As seen from Fig. 4, the dataset loading time is almost constant, demonstrating that any overhead comes from the HDFS access process and not Spark itself (S-base cars : $\mu = 20.259s, \sigma = 0.243s$, check-ins $\mu = 20.374s, \sigma = 0.263s$; S-core cars : $\mu = 20.077s, \sigma = 0.182s$, check-ins $\mu = 20.372s, \sigma = 0.071s$). Notably both datasets fit comfortably within the memory allocated to each executor container.

(a) Using 3 cores / executor



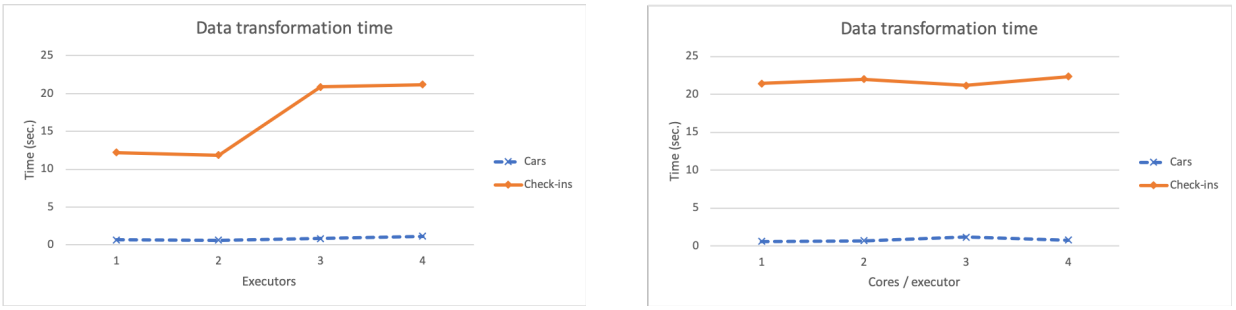
(b) Using 4 executors

Figure 4: Dataset loading time

5.3 Data transformation

After data is loaded into Spark, the first operation on the data is transformation, including cleansing null samples, re-casting data columns to appropriate data types, assembling the feature vector column and encoding the prediction label (for multi-class classification). As seen in Fig. 5, data transformation is more intensive for the check-ins dataset, as a result of the encoding of the prediction label (> 300 labels). Interestingly, while at all other cases the transformation times remain constant, for the S-base configuration we note that the transformation time increases with more than 2 executors. This is the result of the distribution of the mapping operation of

the dataset across multiple nodes and the overhead caused by the communication requirements in reducing and aggregating results across more nodes (S-base cars : $\mu = 0.828s, \sigma = 0.251s$, check-ins $\mu = 16.518s, \sigma = 5.1923s$; S-core cars : $\mu = 0.804s, \sigma = 0.259s$, check-ins $\mu = 21.736s, \sigma = 0.544s$).



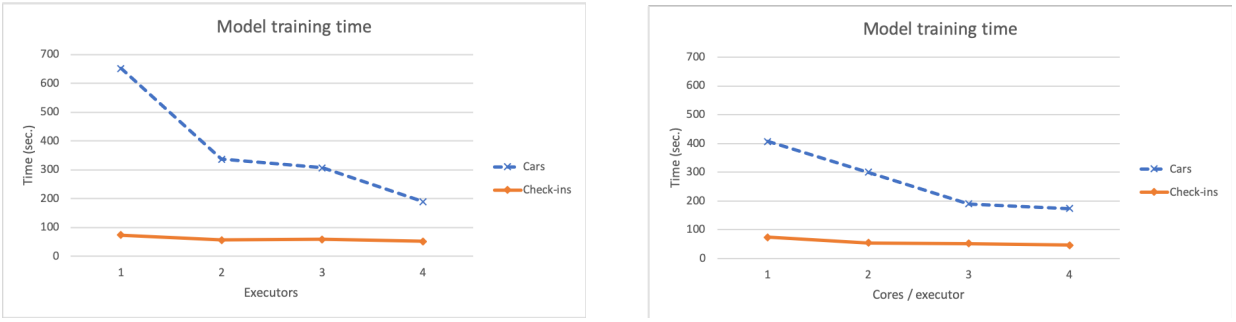
(a) Using 3 cores / executor

(b) Using 4 executors

Figure 5: Data transformation time

5.4 Model training

Next, we investigate the time required to train the models in each scenario. Notably, in all cases, increasing the number of executors or cores per executor yields a performance advantage, even if small. This effect is significantly more pronounced for the cars datasets, which is much larger in size (S-base cars : $\mu = 371.218s, \sigma = 197.369s$, check-ins $\mu = 60.329s, \sigma = 9.636s$; S-core cars : $\mu = 108.332s, \sigma = 108.332s$, check-ins $\mu = 56.893s, \sigma = 11.684s$). A further observation is that allowing access to the 4th core (S-core) that is typically reserved for the operating system, doesn't particularly improve performance. Notably, the average time to train the models is not prohibitive, even in the least favourable conditions, and does not exceed a few minutes of execution time. This demonstrates that periodic training of the edge-based models is feasible and can be performed comfortably in times of low resource demand, even when using large datasets.



(a) Using 3 cores / executor

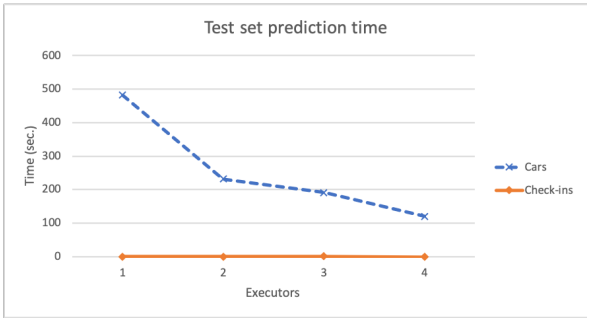
(b) Using 4 executors

Figure 6: Model training time

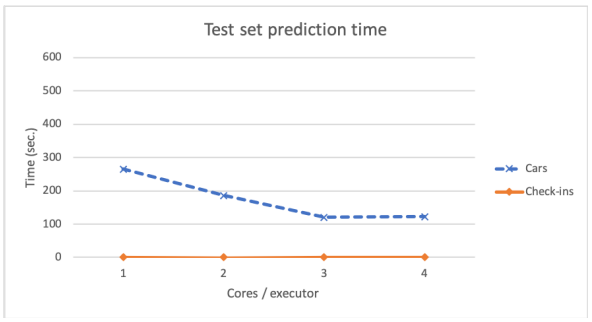
5.5 Performing predictions

In terms of time required to evaluate test sets, again we note that the larger dataset (cars) benefits from multiple executors and number of cores per executor (Fig. 7). As before, allowing access to the additional core in the S-core scenario, doesn't improve performance. Finally, it is noteworthy that for the smaller check-ins dataset, the execution time for the prediction set is sub-second (S-base cars : $\mu = 256.117s, \sigma = 157.592s$, check-ins $\mu = 0.159s, \sigma = 0.114s$; S-core cars : $\mu = 173.103s, \sigma = 68.294s$, check-ins $\mu = 0.176s, \sigma = 0.156s$). These results demonstrate that the cluster is able to resolve predictions on even very large sets, in under 2 minutes.

Related to these results, we report that the prediction of a single feature vector is almost instantanous, across all conditions, often requiring sub-millisecond execution time (S-base cars : $\mu = 0.004s, \sigma = 0.001s$, check-ins $\mu = 0s, \sigma = 0.001s$; S-core cars : $\mu = 0.003s, \sigma = 0.001s$, check-ins $\mu = 0s, \sigma = 0s$). Additionally, the time required to store and load the trained models is very short, as can be seen in Fig. 8 (Saving: S-base cars : $\mu = 5.152s, \sigma = 0.538s$, check-ins $\mu = 6.323s, \sigma = 0.145s$; S-core cars : $\mu = 4.951s, \sigma = 0.489s$, check-ins



(a) Using 3 cores / executor



(b) Using 4 executors

Figure 7: Predictions on test set time

$\mu = 5.924s, \sigma = 0.666s$; Loading: S-base cars : $\mu = 3.928s, \sigma = 0.767s$, check-ins $\mu = 4.394s, \sigma = 0.373s$; S-core cars : $\mu = 3.912s, \sigma = 0.731s$, check-ins $\mu = 4.348s, \sigma = 0.248s$).

(a) Using 3 cores / executor

(b) Using 4 executors

Figure 8: Save and load model time

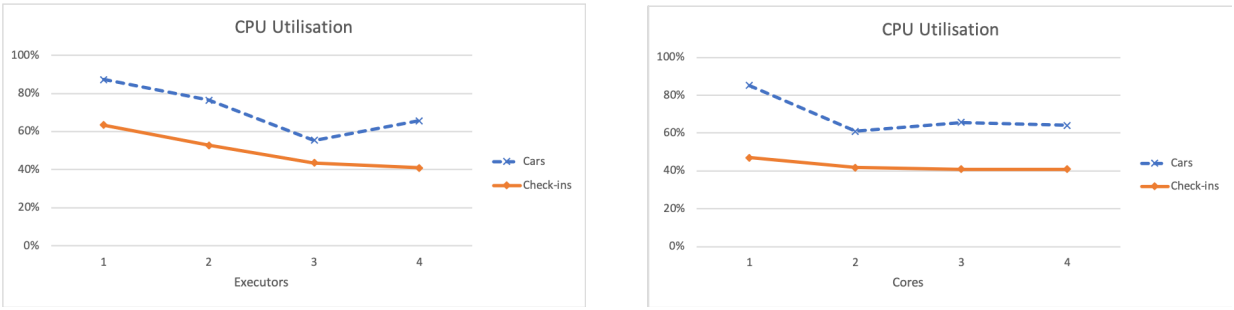
5.6 CPU loads

Finally, we used the SparkLint[3] package to gather statistics from job execution history about CPU utilisation. From these results we note that as the number of executors increases (S-base) the level of CPU utilisation across the entire job decreases, meaning that an increased number of executors affords the cluster more capacity to run additional parallel tasks, as can be expected (Fig. 9). However, with the maximum number of executors (S-core), additional core allocation does not affect CPU utilisation when more than 2 cores are allocated, showing that the additional resources are indeed utilised to decrease overall execution time. Further, plotting the two extremes of this scenario (1 core/executor and 4 cores/executor) we see that the resource utilisation varies significantly. In the former case, most work is carried out using 1 or 2 cores in the cluster (i.e. 1 - 2 executors), while in the latter case, the majority of task execution is separated across all available 16 cores, leading to the reduced execution time (Fig. 10). In this figure, the grey area is idle time (data has been transferred to the driver node), yellow is node-local (data and code resides on the same node), orange is rack-local (processing where data needs to be fetched from another node) and finally green represents local (in-memory) execution time. As part of the job analysis, the main aim here is to minimize the grey area which means that the cluster resources are not being utilised, and as can be seen, the greater number of cores per executor achieves this goal.

6 Training vs. Accuracy tradeoffs

In the preceding analyses, the ML model parameters used were the default values set by Spark's MLlib. Specifically for the decision tree (checkins dataset) case, the generated model is quite simple (max depth: 5, min information gain: 0, min instances per node: 1, information gain measure: gini index). To assess the cluster's performance we considered a scenario where types of venue to check-in could be recommended for a large number of cases, roughly correspondent to the number of venues in a typical city center. We took 5% of the dataset for this purpose (4395 cases) and trained the decision tree on the remaining 95% of the data, using another ML

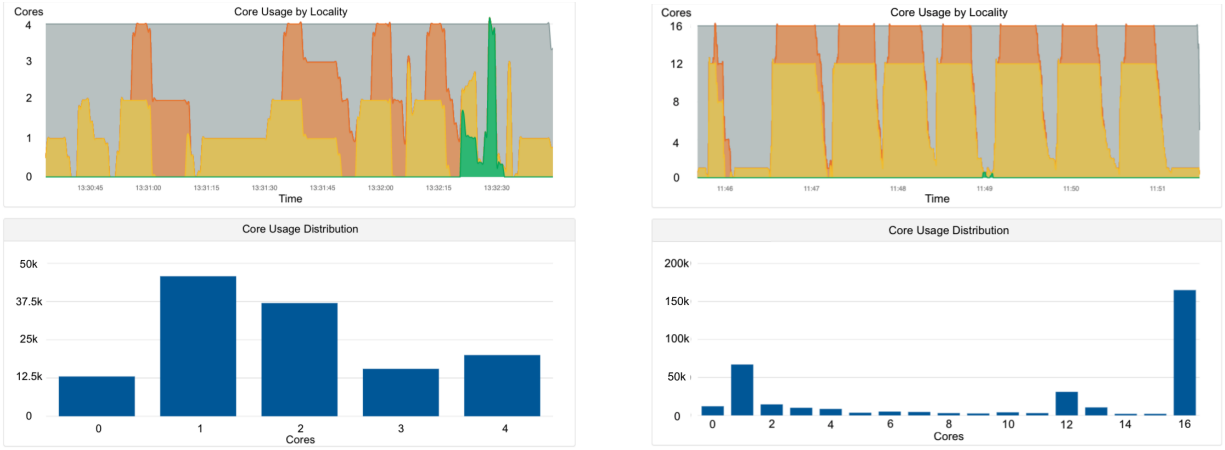
[3]https://github.com/groupon/sparklint



(a) Using 3 cores / executor

(b) Using 4 executors

Figure 9: CPU utilisation



(a) Using 1 cores / executor

(b) Using 4 cores / executor

Figure 10: CPU utilisation distribution

environment (RapidMiner Studio) for convenience, and found (using random parameter search) that a very good performance of 89.15% accuracy can be achieved (max depth: 30, min information gain: 0.01, min instances per node: 2, information gain measure: entropy). Running this experiment on the RPi cluster however yielded an unexpected surprise. While on the Rapidminer environment training took a few seconds, on the RPi cluster the process failed due to insufficient memory on the Java heap, after several minutes of processing. As a reference, YARN containers on the cluster consume up to 2.5G RAM, including 2G for Spark executors and the related overhead (384M). To lighten the load, we experimented to find a smaller tree complexity (depth) and training set size that would yield comparable performance using RapidMiner. We found that a max depth of 15 and sample size at 40% of the original (33405 samples) yielded a good compromise (see Fig. 11). Thus, to assess cluster performance in a more realistic scenario, we ran the experiment again for different sizes of the training dataset between 10 and 40% and predicting on the same number of cases, to assess training time and performance tradeoffs. As shown in Fig.12, an increase of the training set size leads to expectable increases in training time, but not necessarily accuracy. For reference, a fair performance of 72.33% accuracy is attainable with 5m48s of training time using 20% of the original training set (16702 samples).

7 Discussion

In the preceding sections, we have demonstrated the ability to run a small RPi cluster as an edge computing resource, using industry standards such as the Hadoop distributed file system, and Apache Spark for machine learning and data analytics. To the best of our knowledge, this is the first work to present an analytical evaluation of the RPi Model 4B in a cluster configuration for ML tasks. We have demonstrated that the performance of this cluster is sufficient for the purposes of both training and executing ML models in an edge computing context. One of the most encouraging results from our analysis is the short time required to load pre-trained ML models and execute predictions with very fast speed. However, significant additional work remains.

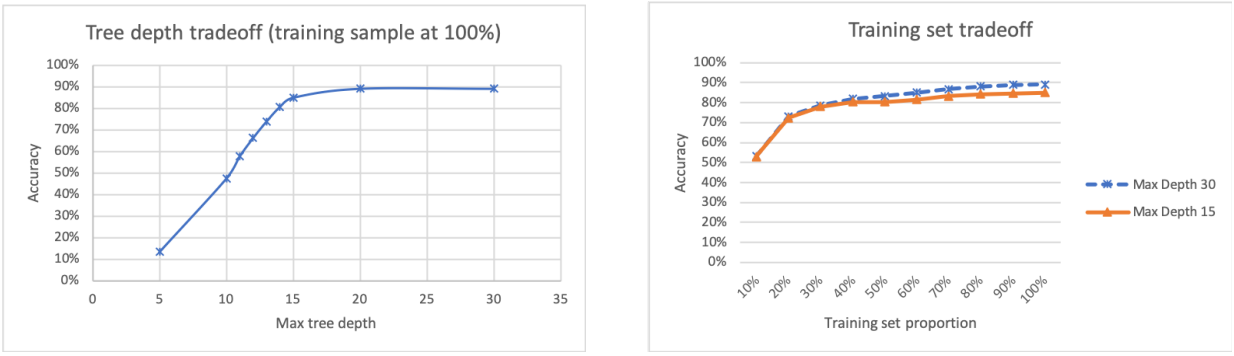


Figure 11: Decision tree model complexity vs. accuracy tradeoffs

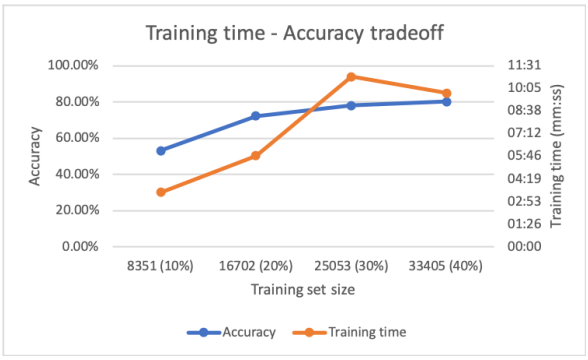


Figure 12: Training time vs. accuracy tradeoff

Firstly, we have run two popular, albeit "lightweight" ML algorithms (linear regression and decision trees). The performance of the cluster should be evaluated using more complex models supported by Spark's MLlib (e.g. gradient boosted trees). Additionally, third party ML libraries such as DeepLearning4J and Tensorflow should be tested for performance, since they support various implementations of artificial neural networks which are better suited for heavier tasks (e.g. image classification and NLP tasks). Another aspect to examine is the size of dataset that can be handled by the cluster. Even though we have tested with a relatively large and a smaller dataset, more analysis is required to identify the performance tradeoffs between dataset size and speed of model training. Additionally, we need to test the cluster's capacity to serve under various request loads. We have noted that single feature vectors can be regressed or classified with sub-millisecond timing, but a real on-line application processing multiple simultaneous user/device requests or handling streaming data (e.g. from IoT devices or social network feeds), will place strain on the cluster's ability to respond in real-time. As a final note, we highlight that our cluster is quite a small setup. This is intentional in our setup, since applications requiring edge computing infrastructures may have strict form factor and physical size limitations. However, it would be interesting to see how performance scales with additional nodes in the cluster.

7.0.1 Acknowledgements

Mr. Antonis Frengkou and Spyros Drimalas helped with the resources necessary for this experiment. Research in this paper was funded by the Hellenic Government NSRF 2014-2020 (Filoxeno 2.0 project, T1EDK-00966)

References

[AHP+] P. Abrahamsson, S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, M. Angriman, J. Rikkilä, X. Wang, K. Hamily, and S. Bugoloni. Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, volume 2, pages 170–175.

[APZ] Yuan Ai, Mugen Peng, and Kecheng Zhang. Edge computing technologies for Internet of Things: A primer. 4(2):77–86.

[CBLM] K. Candelario, C. Booth, A. S. Leger, and S. J. Matthews. Investigating a Raspberry Pi cluster for detecting anomalies in the smart grid. In *2017 IEEE MIT Undergraduate Research Technology Conference (URTC)*, pages 1–4.

[CCB+] Simon J. Cox, James T. Cox, Richard P. Boardman, Steven J. Johnston, Mark Scott, and Neil S. O'Brien. Iridis-pi: A low-cost, compact demonstration cluster. 17(2):349–358.

[CPW] Michael F. Cloutier, Chad Paradis, and Vincent M. Weaver. A Raspberry Pi Cluster Instrumented for Fine-Grained Power Measurement. 5(4):61.

[dABV] Mariano d' Amore, Rodolfo Baggio, and Enrico Valdani. A Practical Approach to Big Data in Tourism: A Low Cost Raspberry Pi Cluster. In Iis Tussyadiah and Alessandro Inversini, editors, *Information and Communication Technologies in Tourism 2015*, pages 169–181. Springer International Publishing.

[DAP+] S. Djanali, F. Arunanto, B. A. Pratomo, H. Studiawan, and S. G. Nugraha. SQL injection detection and prevention system with raspberry Pi honeypot cluster for trapping attacker. In *2014 International Symposium on Technology Management and Emerging Technologies*, pages 163–166.

[DZ] Kevin Doucet and Jian Zhang. Learning Cluster Computing by Creating a Raspberry Pi Cluster. In *Proceedings of the SouthEast Conference*, ACM SE '17, pages 191–194. ACM.

[HT] Wajdi Hajji and Fung Po Tso. Understanding the Performance of Low Power Raspberry Pi Cloud for Big Data. 5(2):29.

[KS] C. Kaewkasi and W. Srisuruk. A study of big data processing constraints on a low-power Hadoop cluster. In *2014 International Computer Science and Engineering Conference (ICSEC)*, pages 267–272.

[LOD] H. Li, K. Ota, and M. Dong. Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing. 32(1):96–101.

[MEM+] A. Mappuji, N. Effendy, M. Mustaghfirin, F. Sondok, R. P. Yuniar, and S. P. Pangesti. Study of Raspberry Pi 2 quad-core Cortex-A7 CPU cluster as a mini supercomputer. In *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 1–4.

[PD] A. M. Pfalzgraf and J. A. Driscoll. A low-cost computer cluster for high-performance computing education. In *IEEE International Conference on Electro/Information Technology*, pages 362–366.

[PHM+] C. Pahl, S. Helmer, L. Miori, J. Sanin, and B. Lee. A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pages 117–124.

[QJK+] Basit Qureshi, Yasir Javed, Anis Koubâa, Mohamed-Foued Sriti, and Maram Alajlan. Performance of a Low Cost Hadoop Cluster for Image Analysis in Cloud Robotics Environment. 82:90–98.

[SGS+] João Saffran, Gabriel Garcia, Matheus A. Souza, Pedro H. Penna, Márcio Castro, Luís F. W. Góes, and Henrique C. Freitas. A Low-Cost Energy-Efficient Raspberry Pi Cluster for Data Mining Algorithms. In Frédéric Desprez, Pierre-François Dutot, Christos Kaklamanis, Loris Marchal, Korbinian Molitorisz, Laura Ricci, Vittorio Scarano, Miguel A. Vega-Rodríguez, Ana Lucia Varbanescu, Sascha Hunold, Stephen L. Scott, Stefan Lankes, and Josef Weidendorfer, editors, *Euro-Par 2016: Parallel Processing Workshops*, Lecture Notes in Computer Science, pages 788–799. Springer International Publishing.

[Tot] D. Toth. A Portable Cluster for Each Student. In *2014 IEEE International Parallel Distributed Processing Symposium Workshops*, pages 1130–1134.

[TWJ+] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros. The Glasgow Raspberry Pi Cloud: A Scale Model for Cloud Computing Infrastructures. In *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, pages 108–112.

[YZQ] Dingqi Yang, Daqing Zhang, and Bingqing Qu. Participatory Cultural Mapping Based on Collective Behavior Data in Location-Based Social Networks. 7(3):30:1–30:23.