AARMS Vol. 16, No. 1 (2017) 37–52.

Processing Intrusion Data with Machine Learning and MapReduce

Csaba BRUNNER[1]

These past years, cyber-attacks became a daily issue for enterprises. A possible defence against this kind of threat is intrusion detection. One of the key challenges is information extraction from this large amount of logged data. My paper aims to identify cyber-attack types as patterns in log files using advanced parallel computing approach and machine learning techniques. The MapReduce programming model is applied to parallel computing, while decision tree algorithms are used from machine learning.

I discuss two research questions in this paper. First, despite parallelization, are machine learning algorithms still able to provide results with acceptable accuracy measured by traditional data mining figures (accuracy, precision, recall, area under receiver operand characteristic [ROC] curve [AUC])? Second, is it possible to achieve significant performance improvement by measuring runtime execution of the algorithm by introducing several measurement points?

I proved that the machine learning model with two categories in the target variable is preferred to the one having five categories. The average performance improvement was 4–5 times faster for the whole algorithm compared to a single core solution. I achieved most of these improvements during the data transfer phase.

Keywords: *intrusion detection, parallel processing, machine learning, network security*

Introduction

With the rapid spread of the internet and related technologies, a new form of crime has appeared. This form evolved together with the technology it was based on. By today, it has grown so big, that it endangers business ventures, especially those that rely on the same technology to deliver value. News of website shutdowns, bank card id thefts and botnet attacks are increasingly common and concerning. What can business ventures do against such threats?

One solution is to stop these attacks before they enter crucial systems, like the internal e-mail server. The first initiatives were DMZs (DeMilitarized Zones) between the internet and the intranet to protect the latter from malicious codes coming from the former. A more advanced form of protection is to use intelligent Intrusion Detection/Intrusion Prevention Systems (IDS/IPS-es), systems that detect TCP/IP packets sent with harmful intent, and prevent the arrival of further packets. This detection/prevention is a complicated task, as the packets are disguised well. They usually follow detectable patterns, which is especially evident with denial of service attacks.

[1] Corvinus University of Budapest, Ph.D. student; e-mail: csaba.brunner@uni-corvinus.hu

According to source [1], IDS-es have several categories and types:

- *Network based IDS:* listen to network activity, when an anomaly is detected, send a warning to the operator. This way they are able to complement the functionality of a firewall.
- *Host based IDS:* configured to the system they were installed on, logging information about resource usage to raise warnings about a potential attack.

The types are the following:

- *Signature based:* These IDS-es protect against detected intrusion patterns stored in the form of signatures.
- *Statistics based:* These systems need a comprehensive definition of the known and expected behaviour of the system.
- *Supported by neural networks:* monitors general activity and traffic of the network, and creates a database. Similar to statistic based IDS-es, but with additional self-learning capabilities.

With signature based intrusion detection, pattern recognition techniques, such as machine learning, are used. „The aim of machine learning is to find a hypothesis best fitting initial observations—with the expectation that the learned pattern or connection could be applied to new observations as well.” [2: 267] For intrusion detection, common pattern mining and classification are the most useful options. One example of classification algorithms are decision trees.

First I talked about processing the data, but storing them is just as important. In what structure is packet data stored on the server, what is the aggregate size of it, and how will this data be accessed again? Of all the questions, the structure and size causes the most trouble. In the case of intrusion detection, structure is moderate as packet information is stored in network logfiles in such a big quantity that it causes problems even to dedicated mainframe architectures. My suggested solution is cheap commodity hardware set up in a parallel processing architecture, and the MapReduce programming model.

In the MapReduce programming model, a function is performed on all observations of a given dataset, often a key assignment to help observation allocation. The observations are then distributed between the nodes and intermediate calculations are performed algorithmically. The final result is generated in the reduce step and is sent to one of the nodes or directly to the user.

The focus of my research consists of machine learning and parallel computing using MapReduce. Intrusion detection is a practical example where the results are used. Research questions which I deal with in this paper are the following:

1. Will the accuracy of models created by machine learning algorithms deteriorate due to parallel computing?
2. Will runtime performance improve by parallelizing the task?

I used a publicly available dataset in the research, and then I have written a customized computer algorithm to test my research questions. Model accuracy was tested with standard data mining figures, such as accuracy, precision, recall, F-score and wherever applicable, AUC. I evaluated performance improvement by inserting several measurement points during the execution of the algorithm.

I structured the article in the following way: first I will give a literature overview. Then, I will elaborate on the theoretical background and research methodology, from Map-

Reduce and the custom-coded program to the sampling of the selected dataset. Next, I will discuss the results of the test runs for both my research questions (accuracy and performance). Finally, I will discuss the findings and propose a trajectory to conduct further research on.

Literature Review

The argument for selecting source [3] was that they had written about the dataset of the SIGKDD '99 data mining competition, the dataset contained data for solving and testing intrusion detection problems, the problem I selected for my research as well. Source [3] introduced the reader to the goals and methods of IDS-es, and provided the most important conclusions of IDS research, such as accuracy, extensibility and adaptability. According to them, several categories of data mining exist that can help in performing intrusion detection: categorization, link and sequential analyses. In their research, they used all three. They called attention to the shortcomings of IDS-es too. In their final model the authors designed basic classifiers for detecting connections between features, then the linked features were grouped and assessed in a final model by an aggregate classifier.

The key conclusion of source [3] was that different intrusion types are better described by different indicators, and are detected by different models:

- *The traffic model:* for defence against Denial of Service (DOS) and fast probing attacks.
- *The host-based traffic model:* for defence against slow probing.
- *The content model:* to detect R2L and U2R attacks.

Next, the authors evaluated the accuracy of the aggregate classifier. The models found features describing probing and U2R attacks well. On the other hand DOS and R2L attacks had a significant standard deviation, feature generation and machine learning provided less convincing results.

Source [3] were the first who took a dataset of intrusions and attempted to analyse it by using machine learning. They proposed a model, which was able to identify attacks on the network. Their article was the first where the dataset from the KDDCup '99 competition was mentioned. Processing this dataset on a parallel architecture is where I attempt to provide new findings.

One potential machine learning algorithm family supporting IDS-es are decision trees. The idea behind them is that they translate complicated connections to a set of simple decisions. They are capable of detecting both linear and nonlinear connections, they can be considered as universal approximators in that regard. Decision tree algorithms start from a root node, select an appropriate variable from the dataset (based on calculations, the most common measure is information gain based on entropy), then split the dataset in two along a selected variable, so that the two parts are more homogenous, than the whole dataset was before. Repeat these steps until stopping criteria is met. This way, every observation can be assigned one single leaf of the tree. The category that has the most observations on a leaf becomes the prediction for future observations. Further readings on decision trees are found in sources [4] [5] [6] [7] [8].

Decision tree algorithms have many advantages:

- they automatically recognize variables with weak predictive power and omit them from the model. This is why decision trees are often used for preliminary variable selection as well;

- their scalability is good, can be used on large datasets;
- an easy to understand set of decisions could be generated from a path leading from the root to a selected leaf;
- decision tree algorithms perform comparably to many other classification models.

Their notable disadvantages:

- they have a tendency to overlearn, meaning they learn not general trends in the dataset, but specific observations of it. This causes poor accuracy when presented with new observations. This is avoided by combining two techniques: testing the model with new observations that were not part of the training, and by pruning the decision tree (by removing partial trees from the model);
- as a classification algorithm, decision trees perform worse on datasets with unequally distributed target variable. To fix this, stratified sampling could be used in a way that overrepresented values in the target variable are under sampled and underrepresented values are oversampled. This is important, because the problem of intrusion detection also involves unequally distributed datasets.

Taking all the advantages and disadvantages into consideration, I chose decision tree algorithms for pattern recognition on the KDD dataset.

Source [9] developed an IDS/IPS equipped with a machine learning algorithm to protect 802.11 Wi-Fi networks against DOS attacks. The two types of DOS attacks against Wi-Fi networks are authentication and authorization attacks. In the former, the attacker sends a lot of authentication messages, thus overloading the Wi-Fi AP (access point). The latter works similarly, except here the goal is to overload the MAC address table. The authors selected several machine learning algorithms: Bayesian networks, AdaBoost, alternating decision trees, SVM and RIDOR algorithm. The focus of the research was on the performance of the machine learning algorithms, with more emphasis on their precision and recall. From all the algorithms, RIDOR, alternating decision trees and AdaBoost performed best, surpassing 90% for both measures. Taking runtime performance into consideration, AdaBoost turned out to be the best choice.

The article shown several examples for supporting IDS-es with machine learning algorithms. A criticism towards it is that it dealt with network issues more, while potential for parallel machine learning remained mostly unexplored.

Source [10] developed a new decision tree algorithm for processing large datasets in a fast and memory efficient way. Several decision tree algorithms were developed before, but these had two issues: the entire training dataset had to be loaded in memory and parameterizing them was a complicated task. The improvement of source's [10] decision tree over the previous ones was that it did not store every training observation in memory, instead, loaded them one by one, and updated the leaves accordingly. If more than a set number of observations were assigned to a leaf, then a cut and reassignment was performed. The authors compared their new algorithm with the already existing ones. The new decision tree algorithm provided comparable or better results. Memory use was evaluated in separate tests. The new algorithm was very efficient in this regard as well.

Source [10] provided a new memory-efficient algorithm for use on commodity computers. Re-using the primary outcome of their research in a parallel environment is potentially beneficial, but the frequency of read cycles on the slow HDD storage remains a question.

Source [11] took a different approach. The article was about a decision tree algorithm implemented in a parallel architecture based on the Message Passing Interface (MPI) standard and the MapReduce programming model. The tree was built up on a central node, while the worker nodes were responsible for calculating the next cut variable. The algorithm collected (reduced) the information gains on the master node which chose the next cut variable. Both the decision tree and the worker nodes were updated according to this decision. The author used the first 15 observations of the iris dataset. Two tests were performed: the first on one multicore computer, the next involved several.

Source's [11] research brought up more questions than it answered. The usability of the results was reduced by the fact that observation count was too low in the tests. Unlike source's [11] research, I assigned the construction of decision trees to the worker nodes as well, simulating a decision forest algorithm. The selected dataset for my research was far bigger than what the author selected: I decided to test my assumptions on the KDD dataset.

Source [12] introduces and calls attention to the hardships during the evolution of parallel computing in his article. It starts with a historical introduction, and then three logically sounding yet bad ideas were introduced: Amdahl's law, "dusty deck" and attached accelerators.

Amdahl's Law: "If half of a computation cannot use even a second processor working in parallel with the first, then, no matter how many processors one employs, the work will take at least half the uniprocessor compute time. If the fraction of work that must be sequential, the Amdahl fraction, is f, then the speedup from parallelism cannot be more than 1/f." [12: 2] The main driving force behind parallelization is not the speed improvement, but the possibility and capacity of it. As complexity increases so decreases the importance of the Amdahl fraction.

Dusty deck: from time to time in order to improve performance, programs have to be changed as their execution model changes. Automated reprogramming is not possible, as too many physical, mathematical and other theories and ideas lie behind the codes that are not referenced by the code.

The third bad idea was the use of attached accelerators. They indeed were capable of boosting the performance of general use computer hardware, but most attempts did not find a wide enough audience. There were several efforts at developing hybrid solutions, but then the programmer had to harmonize two low-level architectures, compared to clusters created from commodity PCs.

At the end of his article [12] introduced some ideas that were not wrong fundamentally, but for some reason did not become widespread enough. Examples are vector computers and shared memory computers. These two ideas live on in modern multicore processors.

Then he proposed some good ideas that are followed even today:

- it is always better to abandon old code and re-think algorithms if it results in better parallelization;
- data should be distributed to minimize communication and data transfer;
- there is no need for shared memory, only for a standard portable messaging layer;
- cheap commodity hardware is preferable;
- memory is the bottleneck, acquiring more computing capacity is cheaper. being harder to access, after a threshold, memory becomes more important in a parallel architecture;
- internal network should be well established with high bandwidth.

The article helped identify the pitfalls that need to be paid attention to when developing a parallel architecture, which are the proposals that look logical, but misleading, and which ideas work in practice.

Research Methodology

In this section I will detail the MapReduce programming model followed by the selected decision tree machine learning algorithm. After this, I will introduce the flow of the custom test program. Then I will elaborate on the dataset, its specifics and sampling together with the introduction of a conceptual hierarchy and my reasoning behind it.

MapReduce

MapReduce is a programming model first invented and used by Google. It is used to perform operations on large datasets as it allows programs to run on parallel clusters of commodity hardware. The following paragraph is based on the work of J. Dean and S. Ghemawat. [13]

The general parallel architecture based on MapReduce has one or more computer called masters which are responsible for resource management on the rest of the architecture. The remaining computers are called workers, and as the name implies, they do most of the calculations. First, a map function is carried out, which performs a function on all the observations of the dataset, most commonly a key assignment, to provide intermediate results. There might be additional calculations with these intermediate results, or the next step follows immediately. This step is the reduce step which calculates the final result using the intermediate result on the master computer. If more master computers are involved, then a second reduce might be performed, ending the MapReduce cycle.

Parallel Architecture

Parallelism took the number of processor cores of the computers connected as a basis; they are the execution threads of the program and the computing nodes of the parallel architecture. I have set up three configurations. The first involved one processor with two cores. I did this to create a basis for comparison; I carried out four runs in this configuration. I expected that the program would take the most time to run in this configuration. The second and third were the real research executions of the program, one single CPU with 4 cores, and two CPUs with 8 cores together. I changed two key factors, one at a time during the test runs: target variable class count (5 or 2 classes) and sample size (small or large). I repeated each run three times to reduce the chance of error. Altogether, the program executed 24 + 4 times, comparison bases included.

Bagging Algorithm

The MapReduce model is not present in my program in a pure state. Reduction is performed separately in a code snippet reminiscent of a bagging algorithm. Bagging has ties with machine learning; the dataset is separated into sections for training on several machine learning models. The models each then generate predictions on new observations, send these predictions to a master node, then a simple voting is performed to generate the final prediction. It was confirmed that bagging can improve the accuracy of unstable machine learning models, such as decision trees. For additional material, check sources [5] and [7].

The Program

I created my test program in Java using a Java implementation of the MPI standard, called MPJ Express and the Waikato Environment for Knowledge Analysis Application Programming Interface (WEKA API). MPI is a general interface that allows programming of communication between different computers. I used MPI in my program, because it has method support for the MapReduce model. WEKA is an open source Java API for data mining, supporting, among others, decision tree algorithms. More information can be found on MPJ Express in source [14], and on the WEKA API in source [15].

The program executes a standard data mining process, but has some additions to it to make it run on a parallel architecture. First, the master loads a pre-sampled dataset, splits it, and distributes the splits between the workers. The workers then train their own decision trees using the samples they received and send a description of their models back to the master. The test phase is next, similar to training; the master sends slices of the test dataset to the workers to test their models. The workers in turn send back the confusion matrices. Next, as a form of validation, 10 observations are sent to every worker. They each make their own predictions on the observations and send them back to the master, where they will go through a simple voting to determine the composite prediction of the architecture, similar to a bagging algorithm. Finally, as the program ran, performance was measured and collected on the workers, and now is sent back in the final step to the master.

Sampling: Observations

To perform a machine learning task, like intrusion detection, data is needed first. The dataset of the KDDCup ’99 data mining competition was chosen, being the most common for solving IDS problems with data mining. The dataset contained ~7 million observations of 41 variables divided into a training set (~5 million observations) and a test set (~2 million observations). This amount of data was more than what the program could handle. For this, first, I tried to use 10% samples, instead of the full datasets. These 10% samples were also provided for the competition. [16]

The 10% samples were still too much, in order to reduce memory load, I used a stratified split. This stratified split was done four times to provide a small and a large sample for both testing and training purposes. Another defining characteristic of sample usage was the number of categories in the target variable (5 or 2 categories). For a short summary, see Table 1.

Table 1. Sampling overview. Sampling was determined by two factors: their intended purpose and their size. [Edited by the author.]

Target variable	Training	Test	Sample size
5 class	3,000	5,000	S
2 class	5,000	3,000	
5 class	6,000	10,000	L
2 class	10,000	6,000	

Sampling: Target variables

Another, smaller issue was with the target variable, it had too many categories. To reduce the number of them, I used a conceptual hierarchy. This way, I could reduce complexity first to a 5 class variable, then to a 2 class variable. Table 2 shows the conceptual hierarchy I constructed.

Table 2. Conceptual Hierarchy. [Edited by the author.]

2 class	**5 class**	**original**
NO	*DOS*	back
		land
		neptune
		pod
		smurf
		teardrop
	norm	normal
YES	*probe*	ipsweep
		nmap
		portsweep
		satan
	r2l	ftp write
		guess passwd
		imap
		multihop
		phf
		spy
		warez-client
		warez-master
	u2r	buffer overflow
		loadmodule
		perl
		rootkit

There is one aspect of the conceptual hierarchy that needs explanation: DOS was chosen to be a “NO” category. What I wanted to achieve with the machine learning model, was to find the rarer attack types first, such as probe, R2L and U2R. Later, by creating a different conceptual hierarchy for the 5 class to 2 class cases, the machine learning model can be altered to detect DOS attacks specifically.

Results

Model Accuracy

The results were evaluated to answer the two hypotheses. My presuppositions were that the number of additional cores does not decrease model accuracy, and that sample size played no role either. The results confirm these, for details, see Table 3 and 4. The comparison basis is included for each set of tests. There are some abbreviations, for example 1p4c means the table is about the 1 processor, 4 cores architecture setup.

Table 3. Data mining model performance with 5 classed target variable, 4 processor cores.
[Edited by the author.]

1p4c	**Small sample (3–5,000 obs.)**				**Large sample (6–10,000 obs.)**			
	1. run	**2. run**	**3. run**	**1p2c**	**1. run**	**2. run**	**3. run**	**1p2c**
Accuracy	0.978	0.964	0.981	*0.984*	0.985	0.985	0.980	*0.987*
Precision	0.477	0.449	0.513	*0.511*	0.532	0.558	0.489	*0.576*
Recall	0.438	N/A	N/A	*N/A*	N/A	N/A	N/A	*N/A*
F-score	0.456	N/A	N/A	*N/A*	N/A	N/A	N/A	*N/A*

Table 4. Data mining model performance with 5 classed target variable, 8 processor cores.
[Edited by the author.]

2p8c	**Small sample (3–5,000 obs.)**				**Large sample (6–10,000 obs.)**			
	1. run	**2. run**	**3. run**	**1p2c**	**1. run**	**2. run**	**3. run**	**1p2c**
Accuracy	0.970	0.977	0.976	*0.984*	0.981	0.981	0.980	*0.987*
Precision	0.397	0.467	0.476	*0.511*	0.513	0.445	0.470	*0.576*
Recall	N/A	N/A	N/A	*N/A*	N/A	N/A	N/A	*N/A*
F-score	N/A	N/A	N/A	*N/A*	N/A	N/A	N/A	*N/A*

Table 3 and 4 show a good fit of the model based on accuracy. The number of cores or the sample size did not alter the outcome. Intrusion detection works better if the number of false positives remains low, as well as the number of correctly identified attacks remains high. This requirement is best described in the precision and recall (and F-score) of a selected model. With the target variable having 5 classes, the model was showing a mediocre precision (~0.5 or less) and recall, apart from one exceptional case, remained incalculable. The reason behind this was the low representation of some categories in the original dataset, R2L and U2R attacks were highly under-represented.

This required a second program execution, this time with stratified samples and a target variable having only 2 classes. The results of these runs are shown on Table 5 and 6. With only 2 categories in the target variable, recall and F-score became calculable, and a new measure, the area under the Receiver Operand Characteristic (ROC) curve was added.

Table 5. Data mining model performance with 2 classed target variable, 4 processor cores.
[Edited by the author.]

1p4c	**Small sample (3–5,000 obs.)**				**Large sample (6–10,000 obs.)**			
	1. run	**2. run**	**3. run**	**1p2c**	**1. run**	**2. run**	**3. run**	**1p2c**
Accuracy	0.785	0.791	0.772	*0.796*	0.795	0.792	0.809	*0.799*
Precision	0.959	0.975	0.866	*0.967*	0.965	0.902	0.903	*0.969*
Recall	0.483	0.490	0.508	*0.506*	0.507	0.539	0.585	*0.513*
F-score	0.642	0.652	0.641	*0.664*	0.664	0.675	0.710	*0.671*
AUC	0.735	0.766	0.783	*0.793*	0.811	0.789	0.776	*0.777*

Table 6. Data mining model performance with 2 classed target variable, 8 processor cores.
[Edited by the author.]

2p8c	**Small sample (3–5,000 obs.)**				**Large sample (6–10,000 obs.)**			
	1. run	**2. run**	**3. run**	**1p2c**	**1. run**	**2. run**	**3. run**	**1p2c**
Accuracy	0.793	0.782	0.788	*0.796*	0.797	0.756	0.777	*0.799*
Precision	0.893	0.931	0.903	*0.967*	0.936	0.895	0.881	*0.969*
Recall	0.546	0.493	0.525	*0.506*	0.529	0.442	0.512	*0.513*
F-score	0.678	0.644	0.664	*0.664*	0.676	0.592	0.648	*0.671*
AUC	0.789	0.719	0.784	*0.793*	0.772	0.771	0.757	*0.777*

By going from 2 classes to 5 classes, accuracy decreased by approximately 0.15, while precision increased to ~0.9. Recall and F scores became available, showing worse, but still acceptable results. The built model had a very low false positive rate, so it did not detect “NO” activities as rare attacks. However, it had more trouble detecting actual rare attacks as attacks and not as “not a rare attack” behaviour. Finally, AUC shown a good fit of the model, values were around 0.75.

Runtime Performance

The number of cores and sample size played a key role in determining runtime performance. My presumption here was that the bigger the sample size was, the longer it took the algorithm to handle the observations. Conversely, as the number of processor cores increased, the algorithm became faster, as more and more parallel threads could run. This is what the overall performance shows on Figure 1.

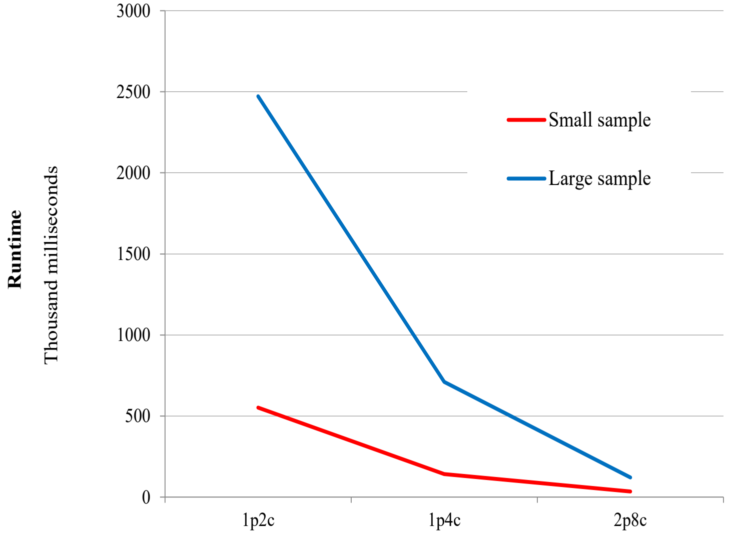

Figure 1. Change of overall performance. [Edited by the author.]

The overall performance shows the runtime of the entire algorithm from start to finish. The speed increase from 2 to 4 cores, as well as from 4 to 8 cores was as high as 4–5 times.

This is not the only result; a detailed picture can be acquired by looking at the different parts of the algorithm. Two received special attention, one dealt with data transfer, the other with the execution time of the decision tree algorithm. Results for data transfer are shown on Figure 2, 3, 4 and 5.

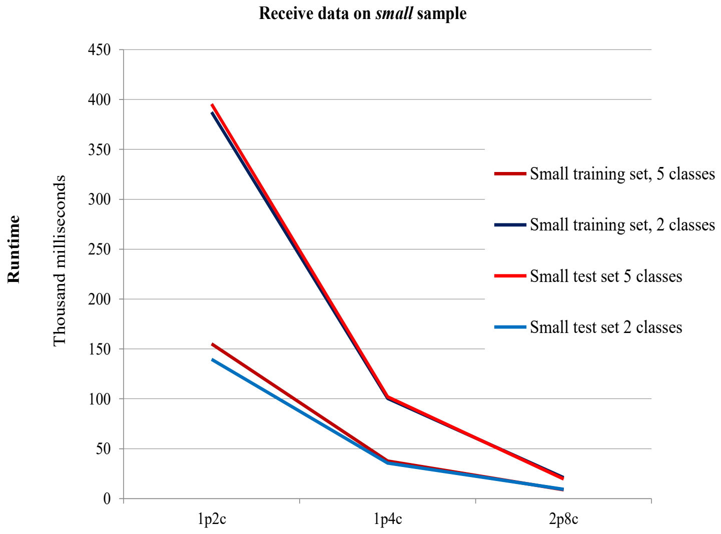

Figure 2. Data transfer measurements over number of cores per sample size (small sample). [Edited by the author.]

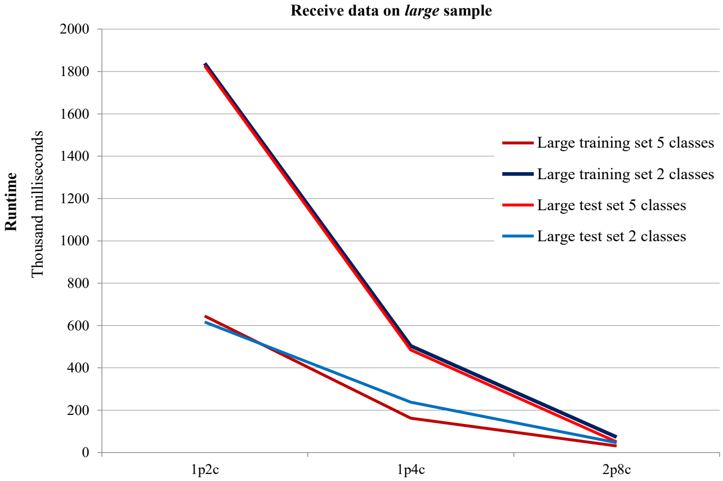

Figure 3. Data transfer measurements over number of cores per sample size (large sample).
[Edited by the author.]

Data transfer charts show a similar tendency to the overall runtime, except the scale is smaller. The charts also indicate a workaround of an error caused by the WEKA API, one that involved sample sizes. As a workaround, the training and test sets between the 2 class and 5 class executions have been switched around. Figure 4 and 5 provides a better insight into this. The charts also show that the data transfer took the most time to complete of all activities, more than 90% of total runtime.

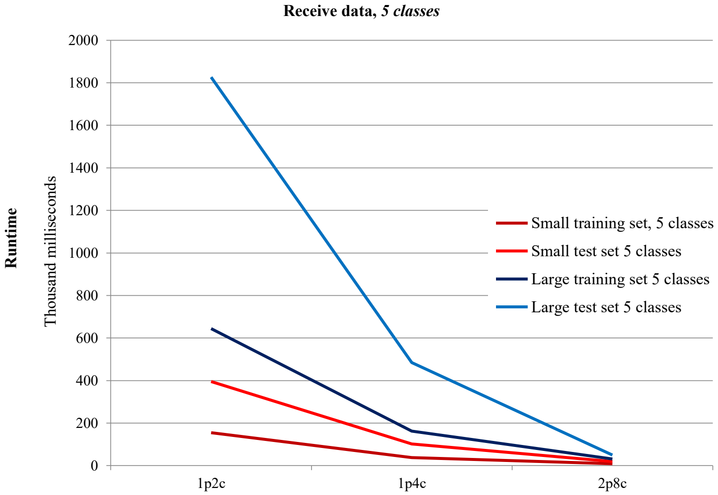

Figure 4. Data transfer measurements over number of cores per target variable classes (5 classes).
[Edited by the author.]

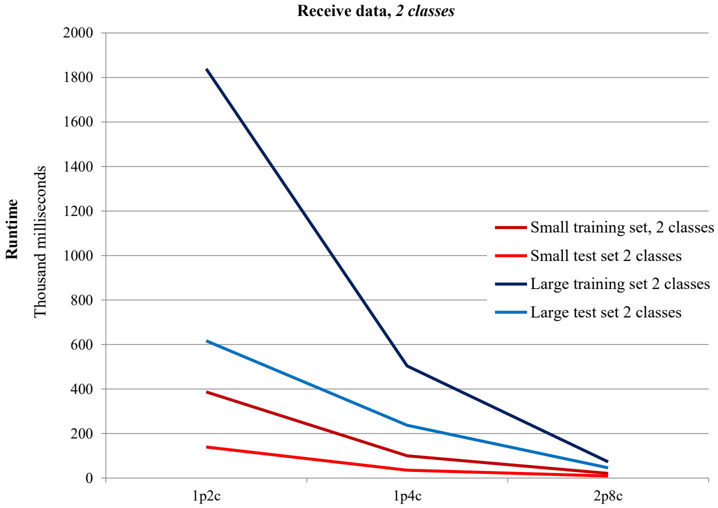

Figure 5. Data transfer measurements over number of cores per target variable classes (2 classes). [Edited by the author.]

The performance measurement of the machine learning algorithm involved the training and testing of the model. The results of model training are shown on Figure 6 and 7 and for model testing on Figure 8 and 9.

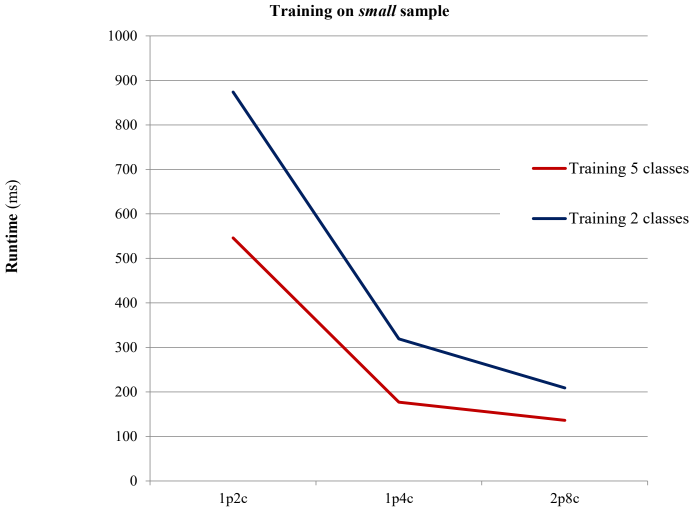

Figure 6. Training performance (small sample). [Edited by the author.]

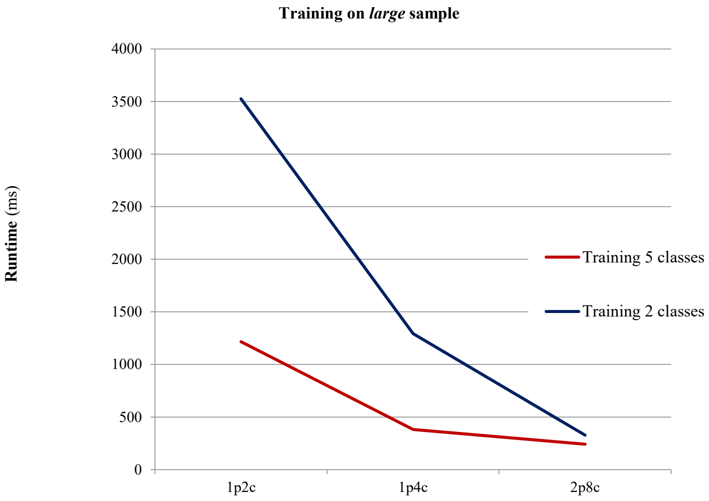

Figure 7. Training performance (large sample). [Edited by the author.]

Training using a 2 class target variable has taken more time, than for a 5 class target variable. The fact that training and test sets were switched around played a key role here as well. Training sets on 5 classes contained 3 and 6 thousand observations, while on 2 classes it contained 5 and 10 thousand. This suggests that the opposite is true for testing performance, where 5 classes will take longer, and 2 classes will be faster.

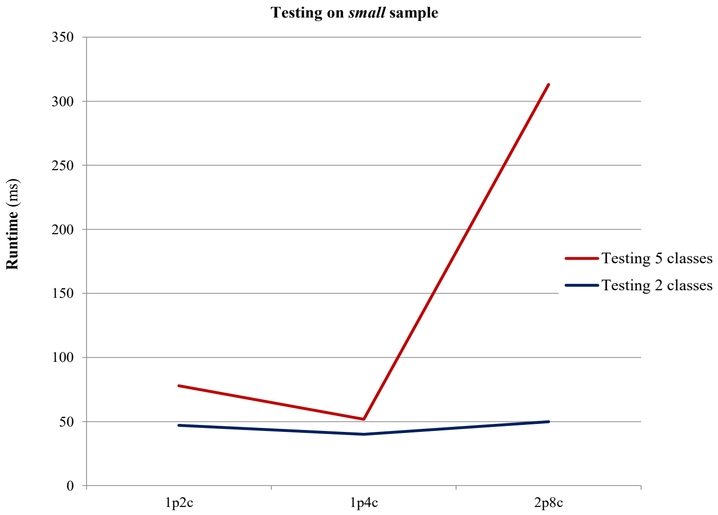

Figure 8. Test performance (small sample). [Edited by the author.]

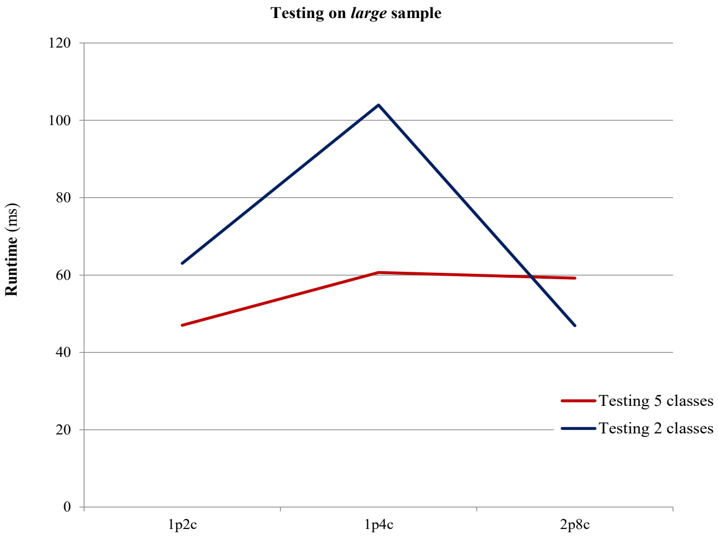

Figure 9. Test performance (large sample). [Edited by the author.]

This assumption is quickly disproven by looking at Figure 8 and 9. In fact, no tendency could be detected during model tests.

Discussion

With my research, I have tried to consider parallel processing and the application of machine learning on parallel processing architectures, asking two main questions, which could be interpreted as hypothesis:

1. Will the accuracy of models created by the decision tree algorithm deteriorate due to parallel computing?
2. Will runtime performance improve by parallelizing the task?

Looking from a strictly top-down perspective, both hypotheses were confirmed. On average, model accuracy did not decrease, not to parallelization. Looking at model accuracy from a closer perspective, one must choose how many classes their target variable should have. With 5 classes, accuracy reached high levels, but two more important measurements, precision and recall, shown sub-par results: one was dangerously low, and the other incalculable. With 2 classes, precision reached high levels and recall became calculable. A requirement towards an intrusion detection system is to have a low false positive and a low false negative rate, which is best described by high precision and recall values, therefore choosing a 2 class variable with the right stratified sampling may be preferable.

A potential topic to continue my research on would be a comparison between the different machine learning algorithms to see which performs best on the selected intrusion dataset. The idea is to take several algorithms, run them on several cores and then compare their performance either with the measures used in this research, or by using a different method. A different idea comes from a drawback of the program developed: individual machine learning model accuracy was easily determined, but a combined accuracy remained largely unexplored, my research only estimated it based on the individual results.

Runtime performance improved by 4–5 times on average. Taking a look at the detailed picture again, we can find where the majority of improvements came from. The time it takes to transfer data between processor cores improved the most, almost exclusively. Machine learning parts also indicated change, but compared to data transfer, it remained negligible. This confirms source's [12] statement about the importance of a high bandwidth network and the need for large amounts of memory to keep data on a storage with fast response times. This is an area worthy of further research.

References

[1] ISACA: *CISA Review Manual 2010.* Rolling Meadows: ISACA, 2010.

[2] SÁNTÁNÉ-TÓTH E., BÍRÓ M., GÁBOR A., KŐ A., LOVRICS L.: *Döntéstámogató Rendszerek.* Budapest: Panem Gazdaságinformatika, 2008.

[3] LEE, W., STOLFO S. J., MOK, K. W.: A data mining framework for building intrusion detection models. *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on IEEE.* 120–132. 1999. DOI: 10.1109/SECPRI.1999.766909

[4] BODON F., BUZA K.: *Adatbányászat.* Budapest: BME, 2013. www.cs.bme.hu/nagyadat/bodon.pdf (Downloaded: 29 12 2015)

[5] OMITAOMU, O. A.: Decision Trees. In. BERRY, M. W., BROWNE, M. (Eds.), *Lecture Notes in Data Mining*. 39–51. Singapore: World Scientific Publishing, 2006.

[6] SHAHNAZ, F.: Decision Trees. In. BERRY, M. W., BROWNE, M. (Eds.), *Lecture Notes in Data Mining*. 79–85. Singapore: World Scientific Publishing, 2006.

[7] LIU, B.: Web Data Mining: *Exploring Hyperlinks, Contents, and Usage Data.* Berlin: Springer Science & Business Media, 2007.

[8] HAN, J., JIAN, P., KAMBER, M.: *Data mining: concepts and techniques.* Amsterdam: Elsevier, 2004.

[9] AGARWAL, M., PASUMARTHI, D., BISWAS, S., NANDI, S.: Machine learning approach for detection of flooding DoS attacks in 802.11 networks and attacker localization. *International Journal of Machine learning and Cybernetics,* 7 6 (2014), 1–17.

[10] FRANCO-ARCEGA, A., CARRASCO-OCHOA, J. A., SÁNCHEZ-DÍAZ, G., MARTÍNEZ-TRINIDAD, J.: Building fast decision trees from large training sets. *Intelligent Data Analysis,* 16 4 (2012), 649–664.

[11] NINAMA, H.: Distributed data mining using message passing interface. *Review of Research,* 2 9 (2013). http://ror.isrj.org/UploadedData/361.pdf (Downloaded: 23 11 2015)

[12] ROBERT, S.: A few bad ideas on the way to the triumph of parallel computing. *Journal of Parallel and Distributed Computing,* 74 7 (2014), 2544–2547.

[13] DEAN, J., GHEMAWAT, S.: *MapReduce: simplified data processing on large clusters.* San Francisco: Google, Inc., 2004. www.usenix.org/legacy/event/osdi04/tech/full_papers/dean/dean.pdf (Downloaded: 23 11 2015)

[14] SHAFI, A., AKTAR, A., JAMEEL, M., CARPENTER, B., ANSAR, J. M., QAMAR, B.: *MPJ Express Project.* 2015. www.mpj-express.org (Downloaded: 27 2 2016)

[15] HALL, M., FRANK, E., HOLMES, G., PFARINGER, B., REUTEMANN, P., WITTEN, I. H.: The WEKA Data Mining Software: An Update. *SIGKDD Explorations,* 11 1 (2009), 10–18.

[16] HETTICH, S., BAY, S. D.: *The UCI KDD Archive.* 1999. http://kdd.ics.uci.edu (Downloaded: 23 11 2015)