2016 30th International Conference on Advanced Information Networking and Applications Workshops

# Implementing Hardware Decision Tree Prediction: a Scalable Approach

Mario Barbareschi

*DIETI - Department of Electrical Engineering and Information Technologies
University of Naples Federico II.
Email:{firstname.lastname}@unina.it
†CeRICT scrl - Centro Regionale Information Communication Technology

***Abstract*—Performance of data classification systems is one of the most important aspect when involved data volume, combined with classical computing approaches, does not match tight constraints on latency and throughput. Indeed, even though the classification accuracy of modern machine learning tools is very suitable for the adoption in many applications, they require many computational resources and elaboration time. In the literature, a huge effort has been done to define new architectures and several hardware implementations have been introduced. In this paper, we show a hardware implementation for the classification system based on the Decision Tree and we formally give a demonstration of its scalability in terms of required resources. At the end, with a significant amount of experimental evidences, we prove that the occupied area and power consumption have a linear behavior against the classification parameters.**

## I. Introduction

The era of Big Data challenges has been involving architectural design aspects as classical approaches are no more adequate to tackle performance worsening, such as time and energy consumption, caused by unprecedented amount of data that has to be processed by applications. The literature identifies the causes of such challenges through the so called 5 'V's, which are Volume, Velocity, Variety, Veracity and Value.

The research scientific and the industrial attention is focusing on new methodologies and techniques, as well as innovative technologies, to handle Big Data in elaboration systems. In particular, in the case of data classification, machine learning and pattern recognition algorithms have to deal with large data sets and with a very high number of samples per time unit (throughput) that have to be classified [1], [2], [3]. Typical application domains are sensor networks [4], e-health systems [5], [6], [7], network traffic analysis [8], [9], and so on. Main advances in machine learning are not only new learning algorithms to enhance classification accuracy, since even a small percentage of incorrect classifications will affect a huge amount of samples, but also new design and implementation techniques, whose aim is to improve the classification speed, mainly exploiting hardware implementations. Indeed, hardware accelerators are able to guarantee high throughput and low latency compared to software approaches [10], [11], [12]. Other techniques exploits Could implementation and integration of existing large-scale distributed services [13].

As data models constantly evolve, and this is certainty emphasized in the Big Data context, any hardware classification system has to guarantee the possibility to update them. All listed features are realized by the Field Programmable Gate Array (FPGA) technology, since it is able to realize high parallel, high speed and large digital designs which can be configured and reconfigured directly on field. The FPGA technology started the configurable computing science and, recently, its potentiality has been considered to be integrated with traditional computer architecture [14].

Among the multitude of different classification approaches proposed so far, Decision Trees (DTs) are one of the most suited for a hardware implementation, since, contrary to other classification techniques, they do not require arithmetic computations, which are costly in terms of occupied area and time, but only comparisons.

In this paper, we formalize the synthesis of a DT prediction algorithm, demonstrating the scalability for the occupied area for worst cases. Through an extensive test campaign conducted on a Xilinx Virtex-5 XC5VLX110T FPGA, we effectively prove that the maximum area and power consumption are an upper-bound for a significant amount of tests. Moreover, we provide further analyses for maximum frequency and latency.

## II. DT Prediction Architecture

DTs are tree-based predictor models which consist in internal nodes, containing rules, and leaves, labeled with a classification value. Basically, a DT represents a set of rules which classify data according to the conditions specified by the nodes. Generally a condition specifies an attribute (feature) in the dataset, a relational operator and a constant. A condition may contain more complex expressions over features, but here this case is not considered. For the sake of ease, in this paper we focus on binary DT as any n-ary tree can be rearranged in a binary form.

Each evaluation on nodes returns a boolean value, since the result is ever expressed with positive (the comparison is true) or negative response (the comparison is false). Such result leads to the left or to the right child, according to the visiting algorithm. Algorithm 1 describes a visiting algorithm over a DT model: starting from the root of the tree, the algorithm

978-1-5090-2461-2/16 $31.00 © 2016 IEEE
DOI 10.1109/WAINA.2016.171

**Algorithm 1** Binary Decision Tree predictor algorithm.

**Require:** A feature vector $f_{vector}$ and a classification model
**Ensure:** The predicted Class for the input $f_{vector}$

```
node ← model.root
while !node.isLeaf do
  if node.ρ(f_vector[node.f], node.k)
  then
    node ← node.childLeft
  else
    node ← node.childRight
  end if
end while
return node.classValue
```

fetches the condition of the current node and determines which child node has to be selected for the next iteration. If the node is a leaf, the algorithm returns its label, which is the classification result. The illustrated prediction algorithm is able to visit a binary DT by evaluating the condition in each node and it steps to left if the condition turns out true, otherwise to right.

Figure 1(a) illustrates an example of a binary DT: it is made of 4 nodes evaluating conditions on two features and its leaves are labeled with three different classes. Features represent numerical values, but a DT may include other data types, such as string or enumerated types.

### A. Run DT Prediction In Hardware

Hardware architectures of DTs have to be programmable since DT predictor models are continuously updated by adding new knowledge and by tuning trained trees. As previously illustrated, in the literature some programmable DT accelerator architectures are available. They are designed in the structure, but their behavior is determined through a configuration. The advantage, of course, is the programming at runtime, but they are memory-based, resulting slower or bigger in the area than fixed implementations. Also, since they have to be realized once, resulting accelerators are static in parameters, such as the maximum number of nodes, the maximum number of features, and so on, meant that their values cannot be dynamically changed. For instance, in [15] the tree visiting accelerator unit can be programmed to compute the prediction algorithm on different trees, but it is designed to execute the algorithm over a maximum number of nodes and features. Hence, if an application requires a predictor which nodes amount is bigger than the maximum available on the accelerator, the unit cannot be used.

The SRAM based FPGA technology can mitigate the problem as it can be (re)programmed on field and on demand by providing a new hardware configuration. Hence, a hardware accelerator can be designed on a FPGA without providing additional memory banks for the core configuration, and allowing the updating of it during the time. Of course, the core update problem is not completely solved since the FPGA could potentially not have enough available resources (such as clock distribution circuitry, look-up tables, and so on) to host future versions of the DT predictor accelerators.

### B. Implementing a Fully Parallel DT Predictor

The main performance related issue of the tree visiting algorithm is the inherent sequentiality caused by the selection instruction which decides which child node has to be picked for the next iteration. Hence the procedure cannot be executed in a parallel fashion. To reach high throughput in the DT visiting avoiding sequential computations, a speculative approach can be considered.

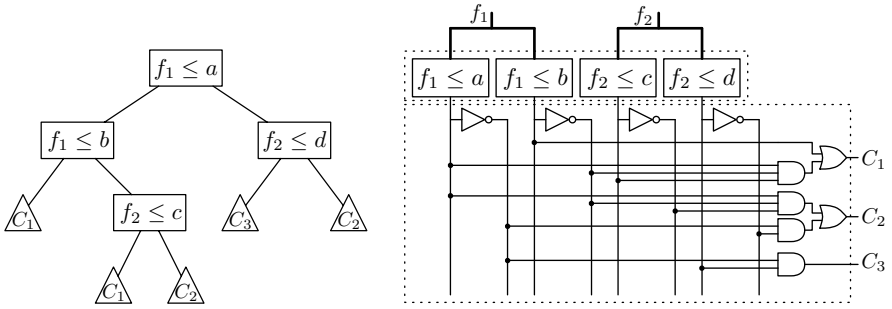
Sequential implementations of the DT prediction algorithm may take advantages from the adoption of well-known computer architectures speculation techniques, such as the branch prediction [15]. Even though they are effective for real applications, sequential approaches do not exploit the high parallelism offered by the hardware.

In [16] a new fully-parallel speculative implementation was introduced. In particular, the approach proposes to evaluate which leaf is reached by computing in parallel all the nodes conditions. Each condition is processed by fast comparators, named Decision Boxes, which work in parallel. Then, a boolean net decides, evaluating the outcomes of the Decision Boxes, which leaf, hence which class, the algorithm returns. In the evaluation performed by the hardware accelerator, the speculation stands in computing some conditions of the tree that are useless for the algorithm result, such that the Algorithm 1 would never evaluate them.

Formally, let $D_\rho^k(f)$ be a node of DT which compares the feature $f$ with the constant $k$ by the relational operator $\rho$, i.e. $D_\rho^k(f) = \rho(f,k)$, $\rho \in \{<, \leq, >, \geq, =, \neq\}$. Each Decision Box implements $D_\rho^k(f)$ and takes in input the actual value for the feature $f$, returning a boolean result.

The boolean net is composed by different boolean functions as many as the number of classes. Formally, let $BF_c$ be the boolean function that rises logic-1 when the classification result is the class $c$, logic-0 otherwise. Therefore, each $BF_c$ has to rise logic-1 when at least one of the paths that lead to a leaf labeled as $c$ is asserted. So each $BF_c$ can be realized through an *or*-gate with a number of inputs (namely the fan-in) which is equal to the leaves labeled with the class $c$. Hence, to define a $BF_c$ for the given class $c$ it is necessary to find the conditions which establish if one of the paths that lead to a leaf labeled as $c$ has been reached. To this aim, let $P_c(i)$, $i \in \{0, 1, ..., leavesOf(c) - 1\}$ be the i-th path that leads to the i-th leaf which is labeled with the class $c$. Moreover, let the assertion $A_C(i)$, $i \in \{0, 1, ..., leavesOf(c) - 1\}$ be the logical intersection of all decision box outcomes which belong to $P_c(i)$. Each decision box outcome is a literal of the boolean function $A_C(i)$ and can be taken direct or negated, according to the path over the tree that leads to the considering leaf. In particular, the literal associated with a node is considered directed if the path continues to its left child, negated otherwise. Thus, each assertion can be defined as:

$$A_C(i) = \prod_{D_\rho^k(f) \in P_i} \begin{cases} D_\rho^k(f) & \text{when the child is left} \\ \overline{D_\rho^k(f)} & \text{when the child is right} \end{cases}$$




(a) An example of decision tree with four internal nodes, two features and three classes.

(b) The hardware implemention of the illustrated decision tree which includes

Fig. 1. Decision tree model example and its hardware implementation.

Finally, for each class $C$ it is possible to define the boolean function as:

$$BF_C = \sum_{i=0}^{leavesOf(C)-1} A_C(i).$$

As one can notice, the boolean net is composed by boolean functions in form of *sum of product* (SoP). Such construction is functionally equivalent to the Algorithm 1, as one and only one assertion $A_C(i)$, and hence only one $BF_c$, rises logic-1 for each input of the circuit.

Figure 1(b) depicts the hardware accelerator unit for the example of DT prediction model in Figure 1(a). As one can see, decision boxes receive the actual values for both the features and their outcomes are available directed or negated. The boolean net contains three functions for the three classes and they are expressed in SoP forms. As previously stated, the speculation of the approach stands the evaluation of tree nodes which are not involved to accomplish the tree visiting algorithm. For instance, in the reported example, the boolean function associated to the class $C_1$ never takes into account the outcome of the decision box which evaluates $f_2 \leq d$, since in the tree such class does not depend on that node.

### C. Consideration on Area Occupancy and Time

The previously described architecture requires a number of decision boxes equal to the number of internal nodes, hence the impact on the occupied area varies linearly on the number of the tree nodes. As for the boolean net, the required resources are related to the assertions' length (*and*-gates fan-in) and to the number of leaves labeled with the same class (*or*-gates fan-in).

First of all, it is worth to notice that some assertions retrieved from a DT may be redundant. In fact, considering the example given in Figure 1(a), the paths to reach leaves labeled with the class $C_1$, are: $P_{C_1}(0) = \left\{D^a_{\leq}(f_1), D^b_{\leq}(f_1)\right\}$ and $P_{C_1}(1) = \left\{D^a_{\leq}(f_1), \overline{D^b_{\leq}(f_1)}, D^c_{\leq}(f_2)\right\}$. The $P_I(0)$ has the associated assertion: $A_I(0) = D^a_{\leq}(f_1) \cdot D^b_{\leq}(f_1) = D^b_{\leq}(f_1)$. Indeed, if $f_1 \leq b$, surely also $f_1 \leq a$, hence the first comparison is redundant.

Thanks to this observation, the number of literals in each assertion can be reduced. Assertions which exploit this rule are defined as essential assertions. Furthermore, considering only essential assertions, it is possible to find an upper limit to the number of literals needed to evaluate them, that is an important result for the scalability of the architecture.

**Theorem 1.** *Essential assertion $A_c(i)$ contains at most $2 \cdot |F|$ literals, being $|F|$ the cardinality of the features set.*

*Proof.* Let us consider the feature set $\{F\}$, the path $P_c(i)$ and the associated essential assertion $A_c(i)$, which is built by adding every literal corresponding to $D^k_\rho(f) \in P_c(i)$. Aiming at evaluating how many literals appear in the assertion $A_c(i)$, we can distinguish three different cases.

***Degeneracy Case***:
$f \in \{F\}$ is not involved in any $D^k_\rho(f) \in P_c(i)$, this means that it has not be chosen by the learner as discriminating feature for the class $c$.

***Case 1***:
$f \in \{F\}$ is a continuous feature that is involved at least once in the path $P_c(i)$. Thus $\rho \in \{<, \leq, >, \geq\}$, $f \in \mathbb{R}$. $D^k_\rho(f)$ establishes an interval for $f$:

- right interval of $k$ ($[k, +\infty[$ or $]k, +\infty[$), if $\rho \in \{>, \geq\}$;
- left interval of $k$ ($]-\infty, k]$ or $]-\infty, k[$), if $\rho \in \{<, \leq\}$.

If in $P_C(i)$ the literal appears complemented, it is possible to simply choose the complemented interval.

If $f$ appears twice or more, $D^k_\rho(f), D^j_\lambda(f) \in P_C(i) \Rightarrow D^k_\rho(f) \cap D^j_\lambda(f) \neq \emptyset$, because the learner generates the conditions by splitting operation over them, consequently $k \neq j$. In other words, the path must be consistent in order to reach the leaf. So the intersection $D^k_\rho(f) \cap D^\lambda_j(f)$ generates:

1) a right interval if both the nodes define a right interval: $[M, +\infty[$ or $]M, +\infty[$ with $M = max(k,\ j)$,
2) a left interval if both the nodes define a left interval: $]-\infty, m]$ or $]-\infty, m[$ with $m = min(k,\ j)$,
3) an interval $[m, M[$ or $]m, M[$ or $]m, M]$ or $[m, M]$, with $m = min(k,\ j)$ and $M = max(k,\ j)$.

This process can be iterated as many times as $f$ appears in nodes along $P_C(i)$. In the first two cases $A_C(i)$ requires only one literal which involves $f$, in the third an intersection (*and*-gate) between two literals.

***Case 2***:
$f$ is a nominal feature $\Rightarrow \rho \in \{==, !=\}$, $f$ can assume finite values. A $D^k_\rho(f)$ appears at most once on any path in the tree. So $A_C(i)$ requires only one literal on $f$. □

Being $DN$ the number of internal nodes of the DT, an implementation of a DT predictor would ideally require only $DN$ *and*-gates and a number of *or*-gates which is maximum equal to $\frac{DN+1}{2}$. Considering the worst case for a $BF_c$, the *and*-gate could have fan-in equal to $2 \cdot |F|$, and an *or*-gate a fan-in equal to $|C| - 1$. The implementation technology is able to realize gates with a maximum fan-in and, with a significant values, a compositional technique is required. On the FPGA, the gates implementation area (and delay) grows with the size of the fan-in like a step function technology, because the FPGA is able to implement each function in a k-input LUT, consequently to realize gates with higher fan-in it combines more that one LUT in a tree scheme.

As for the area occupation, being $N$ the total fan-in an integer multiple of $k$, and $S$ the depth of the tree, $m_i$ (the number of k-gates at i-th level) is: $m_0 = \lceil \frac{N}{k} \rceil$; $m_1 = \lceil \frac{m_0}{k} \rceil$; $m_i = \lceil \frac{m_{i-1}}{k} \rceil = \lceil \frac{m_{i-2}}{k^2} \rceil = \lceil \frac{N}{k^{i+1}} \rceil$. In the last stage there is only one k-gate: $m_{s-1} = 1 = \frac{N}{k^S}$. So the relation between the tree depth $S$, the original fan-in $N$ and the fixed fan-in $k$ is: $k^S = N$, so $S = \frac{lg(N)}{lg(k)}$, hence the number of k-gates is:

$$\#k-gates = \sum_{i=0}^{S-1} m_i \simeq N \cdot \sum_{i=0}^{S-1} \frac{1}{k^{i+1}} = N \cdot \left( \sum_{i=0}^{S} \frac{1}{k^i} - 1 \right)$$
$$= N \cdot \left( \frac{1 - \left(\frac{1}{k}\right)^{S+1}}{1 - \frac{1}{k}} - 1 \right) = \frac{N-1}{k-1}, k > 1.$$

At the end, considering the worst case, the total number of gates $N_g$ for all $BF_c$ can be defined as:

$$N_g = \sum_{\forall c} \left( \frac{2 \cdot |F| - 1}{k-1} \cdot leavesOf(c) \right) + \sum_{\forall c} \left( \frac{leavesOf(c) - 1}{k-1} \right) = \frac{2 \cdot |F| \cdot (DN+1) - |C|}{k-1}$$

where the first summation term is related to number of k-gates to implement the *and*-gate with the worst fan-in ($N = 2 \cdot |F|$), while the second summation term is related to the or that has a fan-in equal to the number of leaves ($N = leavesOf(c)$). The number of classes $|C|$ varies between 2 (best case) and $DN+1$ (worst case). This result is an upper bound of the area occupancy when the DT model is effectively realized over a technology target. Indeed, by applying the resource sharing techniques, subparts of an assertion could be shared among assertions which are derived from the same subpath.

As for the time, the critical path in the boolean net is given by the highest *or*-gate fan-in and *and*-gate fan-in. The worst case for the delay is generated when the DT has all the leaves labeled with a class $C$ except one leaf, which leads to have a *or*-gate with fan-in equals to $DN$, and one of the path of $C$ has an assertion with $2 \cdot |F|$ literals. To tackle an excessive latency, the tree scheme of the gates can be successfully exploited to define a pipelined structure in which each gate output is registered in a dedicated flip-flop. Such technique requires registers as many as gates, but guarantees a throughput which is close to the delay of a single gate.

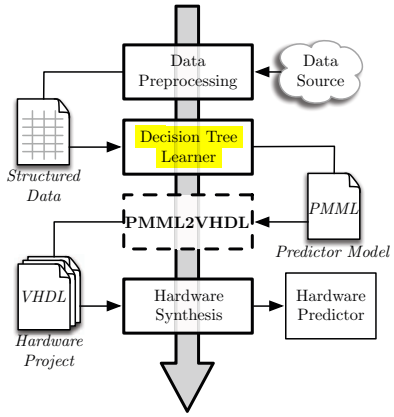



Fig. 2. Automatic flow for the implementation of hardware DT predicion architectures.

### D. Automatic HDL Generation

The demonstration of the Theorem 1 gives a set of transformation rules whereby a DT model can be automatically mapped in hardware description language (HDL). In [17] the authors proposed the automatic flow reported in Figure 2, which includes three main steps. The flow specifies that the model is coded in Predictor Model Mark-up Language (PMML), a standard XML scheme which describes several predictor models [18]. Exploiting such formalism, the model is translated in HDL by means of a tool, the *PMML2VHDL*, which applies the transformation rules previously introduced and implements the DT predictor in VHDL language. This last artifact can be synthesized on the target technology, such as the application specific integrated circuit or the FPGA.

This automatic flow has been successfully adopted to automatically train DT models and obtain hardware accelerator units [19], [20], [21], [22].

## III. Evaluation

In this section, we provide experimental results related to the overhead of implementations of DT predictors on an FPGA. In particular, we adopted a Xilinx Virtex-5 XC5VLX110T FPGA within the XUPV5 development board. This technology is characterized by look-up tables (LUTs) with a fan-in equals to 6. The aim is to experimentally validate the scalability of the approach for area and power consumption. Moreover, we provide also an analysis on latency and throughput for DT predictors.

TABLE I
OVERHEAD IN TERMS OF REQUIRED AREA (LUTS) AND TIME (NS) OF *Floating Point Decision Box*

| Pipe Depth | | $<$ | $\leq$ | $>$ | $\geq$ | $==$ | $!=$ |
|---|---|---|---|---|---|---|---|
| 1 | Delay (ns) | 2,439 | 2,416 | 2,416 | 2,439 | 2,287 | 2,287 |
| | Area (LUTs) | 14 | 14 | 14 | 14 | 7 | 7 |
| 2 | Delay (ns) | 2,039 | 1,955 | 1,995 | 2,039 | 1,880 | 1,880 |
| | Area (LUTs) | 29 | 29 | 29 | 29 | 26 | 26 |
| 4 | Delay (ns) | 1,641 | 1,641 | 1,641 | 1,641 | 1,641 | 1,641 |
| | Area (LUTs) | 59 | 58 | 58 | 59 | 61 | 61 |
| 8 | Delay (ns) | 1,552 | 1,552 | 1,552 | 1,552 | 1,552 | 1,552 |
| | Area (LUTs) | 108 | 108 | 108 | 108 | 108 | 108 |

In order to retrieve DTs with desired characteristics, such as number of nodes, maximum tree depth, number of classes, and so on, authors in [21] developed PMMLGen, namely a Java tool which builds PMML models with desired parameters. This application avoids the learning for actual training sets to obtain DT models with the desired characteristics, as it is unfeasible control them by defining tailored data sets. We collected 400 models whose characteristics are needed for retrieving trends of the hardware-related parameters according to their features. In particular, models were generated by random picking parameters in the following range: number of classes from 2 to 50; number of features from 1 to 40; number of internal nodes from 50 to 3500; tree depth from 2 to 15.

Hardware DT predictor were implemented with decision boxes with the support to main integer types. To guarantee a good balancing in the pipeline architecture, even the decision boxes were realized by means of the pipeline mechanism. The Table I summarized the overhead introduced by each floating point decision box configuration, varying the pipeline depth. As one can notice, the delay decreases with the number of pipeline stages, but the area overhead significantly increases. To guarantee a good trade-off between area and delay, we configured the decision boxes to work with a depth equals to 2.

### A. Area and Power Consumption

Every DT generated by means of PMMLGen was synthesized by involving Xilinx PlanAhead 14.5 design tool. We collected the occupied area in terms of LUTs, slices and registers at the end of the place and route phase. As for the power consumption, we first evaluated for each DT predictor the maximum clock frequency, which is automatically generated by the synthesis tool considering the worst logic delay in the circuit. Then we retrieved for each experiment the Switching Activity Interchange Format (SAIF) file, which contains the switching activity of the DT predictor. This file enables to calculate, together with environmental parameters, such as the ambient temperature, the airflow, board temperature, and so on, both the static and dynamic power consumption.

Figure 3 illustrates observed occupied area and power consumption against the upper-bound value for each experiment. For smaller predictors, i.e. predictor with a lower impact in terms of area and energy, evaluated upper-bounds are closer to observed values. Conversely, bigger configurations exhibit values which are significant smaller that the maximum value. This behavior can be explained considering optimization of the resource sharing accomplished by the synthesis tool. Moreover, there are some singularity associated with configuration with more that 10k LUTs, which result very close to the upper-bound value.

It has been observed that they are associated with experiments in which the tree density (number of nodes on $2^{\textbf{tree depth}}$) was close to 1.

### B. Time Performance

To evaluate the latency and the throughput, we considered the same maximum clock frequency adopted to evaluate the power consumption. Being designed with a pipeline, the maximum frequency of DT predictors should be equal to worst (minimum) frequency value of the pipeline stages. This value is ideally constant and does not depend on the number of stages. Unfortunately, the FPGA routing introduces additional delay among pipeline stages, which causes frequency worsening when DT predictors have an area occupancy close to the device limit.

In Figure 3(c) we report maximum clock frequency value varying the number of tree nodes. Mostly, the frequency linearly decreases with the number of nodes. Indeed maximum speeds corresponds to trees with about 100 nodes. Figure 3(d) graphs the latency of the hardware DT prediction varying the number of tree nodes. Latency indicates how much time input data take to flows through all pipeline stages, hence longer is the pipeline, bigger is the latency. High latency values are associated to design with a significant number of nodes and with DT model with a significant tree depth.

## IV. CONCLUSION

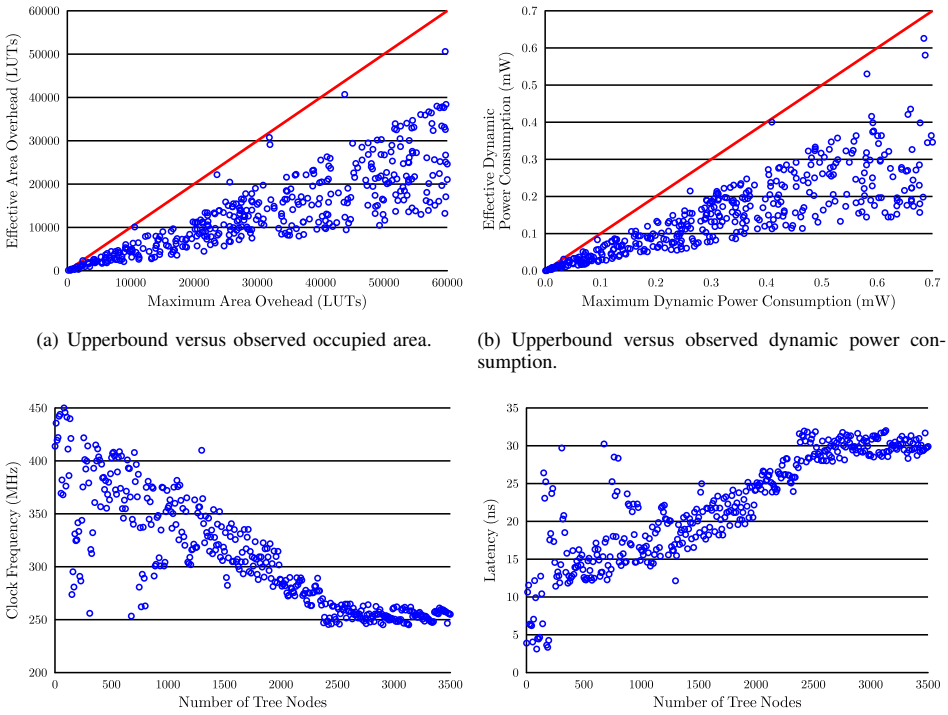
In this paper we presented a hardware based approach to implement a DT prediction algorithm. We introduced the methodology to obtain the hardware accelerator from a DT trained model, formally giving a demonstration of scalability. In particular, we demonstrated that the number of gates needed to synthesize a DT prediction algorithm is at most proportional to the product of number of internal nodes and the number of features. Through an extensive experimental campaign, we also gave an empirical view on the effective occupied area and effective power consumption by implementing a set of 400 DTs on the Xilinx Virtex-5 XC5VLX110T device.


## ACKNOWLEDGMENT

This research work was partially supported by CeRICT for the project NEMBO (Studio e sperimentazione di sistemi iNnovativi “EMBedded” caratterizzati da elevata efficienza per applicazioni ferrOviarie) - PONPE_00159.


## REFERENCES


[1] N. Japkowicz and J. Stefanowski, “A machine learning perspective on big data analysis,” in *Big Data Analysis: New Algorithms for a New Society*. Springer, 2016, pp. 1–31.




(a) Upperbound versus observed occupied area.

(b) Upperbound versus observed dynamic power consumption.

(c) Maximum clock frequency varying the number of tree nodes.

(d) Latency of hardware DT prediction varying the number of tree nodes.

Fig. 3. Experiments conducted over artificial built DT models.


[2] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 1, pp. 97–107, 2014.

[3] P. Zikopoulos, C. Eaton *et al.*, *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[4] Y. Gadallah, M. el Tager, and E. Elalamy, "A framework for cooperative intranet of things wireless sensor network applications," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on*. IEEE, 2015, pp. 147–154.

[5] M. Barbareschi, S. Romano, and A. Mazzeo, "A cloud based architecture for massive sensor data analysis in health monitoring systems," in *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2015 10th International Conference on*. IEEE, 2015, pp. 521–526.

[6] S. Salas-Vega, A. Haimann, and E. Mossialos, "Big data and healthcare: Challenges and opportunities for coordinated policy development in the eu," *Health Systems & Reform*, no. just-accepted, pp. 00–00, 2015.

[7] F. Amato, V. Casola, N. Mazzocca, and S. Romano, "A semantic approach for fine-grain access control of e-health documents," *Logic Journal of the IGPL*, vol. 21, no. 4, pp. 692–701, 2013.

[8] M. Barbareschi, A. Mazzeo, and A. Vespoli, "Network traffic analysis using android on a hybrid computing architecture," in *Algorithms and Architectures for Parallel Processing*. Springer International Publishing, 2013, pp. 141–148.

[9] R. Yuan, Z. Li, X. Guan, and L. Xu, "An svm-based machine learning method for accurate internet traffic classification," *Information Systems Frontiers*, vol. 12, no. 2, pp. 149–156, 2010.

[10] A. Cilardo, "New techniques and tools for application-dependent testing of fpga-based components," *Industrial Informatics, IEEE Transactions on*, vol. 11, no. 1, pp. 94–103, Feb 2015.

[11] A. Cilardo and N. Mazzocca, "Exploiting vulnerabilities in cryptographic hash functions based on reconfigurable hardware," *Information Forensics and Security, IEEE Transactions on*, vol. 8, no. 5, pp. 810–820, May 2013.

[12] A. Cilardo, L. Gallo, and N. Mazzocca, "Design space exploration for high-level synthesis of multi-threaded applications," *Journal of Systems Architecture*, vol. 59, no. 10, Part D, pp. 1171 – 1183, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762113001537

[13] F. Moscato, F. Amato, A. Amato, and R. Aversa, "Model-driven engineering of cloud components in metamorp(h)osy," *International Journal of Grid and Utility Computing*, vol. 5, no. 2, pp. 107–122, 2014.

[14] M. Rogoway, "Done deal: Intel buys altera for $16.7b," 2015.

[15] M. Barbareschi, E. Battista, N. Mazzocca, and S. Venkatesan, "A hardware accelerator for data classification within the sensing infrastructure," in *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on*. IEEE, 2014, pp. 400–405.

[16] F. Amato, M. Barbareschi, V. Casola, and A. Mazzeo, "An fpga-based smart classifier for decision support systems," in *Intelligent Distributed Computing VII*. Springer, 2014, pp. 289–299.

[17] F. Amato, M. Barbareschi, V. Casola, A. Mazzeo, and S. Romano, "Towards automatic generation of hardware classifiers," in *Algorithms and Architectures for Parallel Processing*. Springer, 2013, pp. 125–132.

[18] A. Guazzelli, "What is pmml? explore the power of predictive analytics and open standards," *IBM developerWorks*, 2010.

[19] M. Barbareschi, A. Mazzeo, and A. Vespoli, "Malicious traffic analysis on mobile devices: a hardware solution," *International Journal of Big Data Intelligence*, vol. 2, no. 2, pp. 117–126, 2015.

[20] M. Barbareschi, A. De Benedictis, A. Mazzeo, and A. Vespoli, "Mobile traffic analysis exploiting a cloud infrastructure and hardware accelerators," in *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on*. IEEE, 2014, pp. 414–419.

[21] M. Barbareschi, S. Del Prete, F. Gargiulo, A. Mazzeo, and C. Sansone, "Decision tree-based multiple classifier systems: An fpga perspective," in *Multiple Classifier Systems*. Springer International Publishing, 2015, pp. 194–205.

[22] M. Barbareschi, A. De Benedictis, A. Mazzeo, and A. Vespoli, "Providing mobile traffic analysis as-a-service: Design of a service-based infrastructure to offer high-accuracy traffic classifiers based on hardware accelerators." *Journal of Digital Information Management*, vol. 13, no. 4, p. 257, 2015.