Towards ANN-based Scalable Hashing Algorithm for Secure Task Processing in Computational Clouds

Jacek Tchórzewski
AGH University of Science and Technology
30-059 Cracow, Poland
Cracow University of Technology
31-155 Cracow, Poland

Agnieszka Jakóbik
Cracow University of Technology
31-155 Cracow, Poland

Daniel Grzonka
Cracow University of Technology
31-155 Cracow, Poland

KEYWORDS

Computational Clouds; Cloud Security; Tasks Scheduling; Artificial Neural Networks; Intelligent Security System; Hash Functions; Scalable Hashing; Metaheuristics.

ABSTRACT

Task processing security is a key challenge in large scale distributed environments such as computational clouds. Methods that guarantee the identification and integrity of the data are hash functions. Their computing load is high. Security models are expected to have an insignificant impact on the performance of the computing environment. In this paper, we have proposed the ANN-based model for scalable hashing of tasks results. The model is named Dynamic ANN-based Light Hashing Function. In the presented approach the hash length may be changed. The length of the hash affects the number of cryptographic operations performed and influences time devoted to the task processing. Therefore, applying hash having longer output allows loading the fast computing resources more. Using shorter hash length enables to unload the slow computing units. Our approach allows selecting such parameters of cryptographic algorithms that will not significantly affect the makespan of a task batch.

I. INTRODUCTION

Assuring the proper cryptographic services results in computational overhead during tasks processing. More challenges arise when data needs to be processed very fast, or the amount of data is very large [11]. Cryptography hash functions are well-recognized methods for verifying the data sets integrity. Most frequently used: SHA-2 and SHA-3 do not allow the change of the output length. They are providing only a small variety of outputs in bits. For example, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256. There is no possibility of obtaining output length that is in between those values. The proposed model enables hashing with the hash length that may be chosen in a flexible way, with a step of 1 bit.

The task scheduling problem is one of the biggest challenges in high distributed computational environments composed of heterogeneous computational units. The task scheduling problem covers both - the problem of assigning tasks to appropriate resources and the problem of managing their execution. Effective scheduling of computational tasks is crucial in achieving high environmental performance [4]. However, the scheduling problem is not limited only by maximizing the efficiency of infrastructure; this problem also affects other areas and aspects of the environment, such as security, energy consumption, environment customization, and others. In our paper we consider both, performance and security.

The paper presents also a model of the flexible hashing method which (for example) can be used in the Cloud systems. It is based on consuming the predicted idle time of the Virtual Machines (VMs) that are waiting for the next batch of tasks to be processed.

The model is based on the two-level scheduler. In order to test the presented model, Independent Batch Scheduler ([14]) was adopted. The intuitive scheme of the model integrated with the environment is shown in Figure 1.

The paper is organized as follows. Section III presents the concept of the Dynamic ANN-based Light Hashing Function. In section IV we described the idea of security-driven task scheduling. In Section V, we describe in details tests of the system and discuss results. In that part, we are presenting the application of the proposed model for an exemplary scheduling problem composed of 5 VMs and 25 tasks. The paper ends with Section VI, which contains a summary, conclusions based on the results of the experiments, and ideas for future work.

II. RELATED WORK

In general, a hash function is a function that is capable of mapping data of arbitrary size onto data of a fixed size. Different types of hashing were introduced in the literature [19]. From among them the simplest Trivial Hash Function, Multiplicative Hashing, hashing

Communications of the ECMS, Volume 33, Issue 1,
Proceedings, ©ECMS Mauro Iacono, Francesco Palmieri,
Marco Gribaudo, Massimo Ficco (Editors)
ISBN: 978-3-937436-65-4/978-3-937436-66-1(CD) ISSN 2522-2414

with checksum functions or Perfect Hashing that maps each input into a different hash value. In this paper we consider cryptographic hash functions having 3 properties:

1. It is computationally fast to calculate a hash for any given data.
2. It is extremely computationally difficult to calculate an alphanumeric text from a given hash.
3. It is extremely unlikely that two different messages will have the same hash.

A cryptographic hash function should behave like a random function and is considered "insecure" from a cryptographic point of view, if: it is possible to find a message that matches a given hash value, or two different messages having the same hash value. Most of algorithms introduced so far, including the most popular SHA-2 and SHA-3 offers only the possibility to obtain the certain hash length of either 256, 386 or 512 bits, [1], [2]. Light cryptography hash functions offers smaller hashes, [20], eg. output string equal to 60, 80, 128, or 160 bits. In all mentioned solutions in order to change the output string length, we need to change either the hashing algorithm into another one or use the different version of the algorithm. Dynamic hash functions were introduced in [17] for checking the integrity of the data located in hard drives. But this algorithm is not cryptographically secure. It may not be used for digital signatures. Some flexibility according to the hash function structures offers the algorithm introduced in [6]. The Segmented Hash algorithm presented in this paper enables the use of multiple logical hash tables. The change of the hash function internal structure is possible due to having several hash tables with a different number of buckets each. This method does not offer obtaining a different length of the hash. Only the computational effort of the algorithm may be changed.

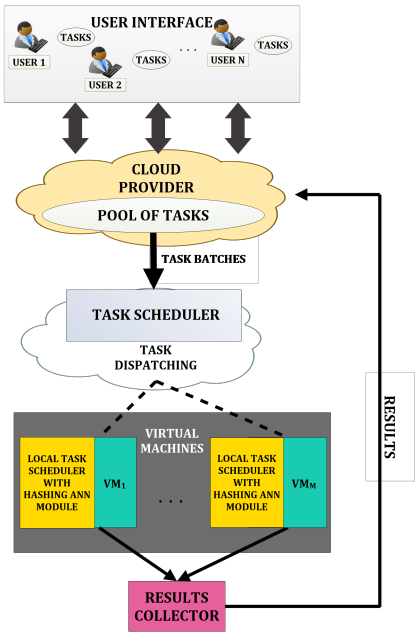

Fig. 1: Model of the two level task scheduling.

TABLE 1: Mackey - Glass time series parameters.

a	0.2
b	0.1
τ	80
x_0	random from range $[0, 2]$

In [15] genetic algorithm is used for producing the hash from the message. The length of the has output is regulated by using population size variable. But authors claimed that this evolutionary method is producing non-cryptographic hash functions. Moreover, only typical length of 512 bit hash were tested.

The main aim of the solution presented in the paper is to obtain the hashing algorithm that may be used in cryptographic procedures and offers the possibility to change the bit length of the fingerprints of the hashing messages.

III. MODEL OF DYNAMIC ANN-BASED LIGHT HASHING FUNCTION

The presented algorithm is based on Artificial Neural Networks (ANN) paradigm. A hashing algorithm is operating on strings of data (inputs), that are compressed into the certain, known length. For the simplicity of the presentation, all considered strings are represented in bits. The following scheme was adopted to obtain the appropriate hashing function:

- choosing the number of bits in hashing function output H;
- training two layered ANN 64-m-n-H to mimic the Mackey - Glass time series, where m is the number of neurons in the ANN hidden layer, and n is the number of neurons in an output layer;
- applying trained ANN as the compression function for the considered data set.

For clarity, H=64 is presented in the paper. For training ANN, input data $INPUT^{train}$ consisted of 100 vectors were selected. Each input string contained 64 bits, (see eq. (1)). Bits were generated randomly. Thus each vector represented a random message, which should be hashed.

$$INPUT^{train}[i] = [b_1, b_2, b_3, ..., b_{64}], i = 1, ..., 100. \quad (1)$$

The testing data $INPUT^{test}$ were generated in the same manner, but the number of vectors were increased to 10000, see eq. (2).

$$INPUT^{test}[i] = [b_1, b_2, b_3, ..., b_{64}], \quad i = 1, ..., 10000. \quad (2)$$

The target data $TARGET^{train}$ used for training were generated in a different way. The Mackey - Glass time series, [5], was used for this purpose. Mackey - Glass equation appears to be chaotic when the τ parameter is equal to or greater than 17. In the paper $\tau = 80$ was chosen, see tab. (1).

The starting point of computation x_0 was chosen randomly. However, all inputs from $INPUT^{train}$ got separate Mackey - Glass equation. The differences were in values of x in the time interval $[-\tau, 0]$. After adjusting the parameters properly, we discretized this time interval into 8 samples. Note that this vector is also an initial condition and can be called *history vector* [5]. Then we gathered input vectors $INPUT^{train}$ from bit into byte form $INPUT'^{train}$. Bytes were represented by numbers from 0 to 255, see eq.(3).

$$\begin{aligned} &INPUT'^{train}[i] = [n_1, n_2, ..., n_8], \\ &n \in [0, 255] \wedge n \in \mathbb{N}. \end{aligned} \tag{3}$$

Those byte values were put into a discreet version of history vectors. This allows us to connect particular Mackey - Glass equation with particular input from $INPUT^{train}$. Then 100 times series were generated, each consisting of 24000 samples, see eq. (4).

$$\begin{aligned} &MG[i] = [x_1, x_2, ..., x_{24000}], \\ &i = 1, ..., 100; x \in [0, 2] \wedge x \in \mathbb{R}. \end{aligned} \tag{4}$$

After that we calculated the *step* value, to determine which samples will create potential 64 bit hash. The *step* was equal to $\frac{24000}{64} = 375$. Accordingly to the following computations, target data used for training purposes ($TARGET^{train}$) is presented in eq. (5).

$$\begin{aligned} &TARGET^{train}[i] = \\ &[MG[i][x_{1*step}], ..., MG[i][x_{64*step}]]. \end{aligned} \tag{5}$$

where $i = 1, ..., 100$ denotes particular input $INPUT^{train}[i]$ and corresponding to this input hash (target) $TARGET^{train}[i]$.
A pair $(INPUT^{train}, TRAGET^{train})$ was passed to all 3 ANNs as a training data. The $INPUT^{test}$ was then passed to them as a testing data. The results: $OUTPUT'_1$ for ANN_1, $OUTPUT'_2$ for the ANN_2 and $OUTPUT'_3$ for the ANN_3 were matrices, consisting 10000 vectors, each consisting of 64 real numbers from range $[0, 2]$. Those outputs had to be binarized to form hashes. To do this, we calculated mean values in each column. If considered value from column j of output of ANN o was greater or equal to the average value from this column (AVG^o_j), it was set to 1, otherwise was set to 0 (see eq. (6)).

$$OUTPUT_o[i][j] = \begin{cases} 1 & \text{if } OUTPUT'_o[i][j] \geq AVG^o_j \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Where $o = 1, .., 3$ denotes ANN number ($ANN_{1,...,3}$) which was used to create hashes, $i = 1, ..., 10000$ denotes hash number and $j = 1, ..., 64$ denotes a column number.

IV. MODEL OF SECURITY DRIVEN SCHEDULING

A. *Model of computational unit and task*

In this paper, we have extended our previous security driven model of the computational unit and task presented in [14], [7] which is based on the fuzzy trust model [16]. In this model computational unit $i(i = 1, 2, \ldots, m)$ is described by the following parameters:
- cc_i - computing capacity expressed in floating point operations per second;
- tl_i - trust level in the range of [0, 1].

Similarly, task $j(j = 1, 2, \ldots, n)$ is described by the following parameters:
- wl_j - workload expressed in floating point operations;
- sd_j - security demand in the range of [0, 1].

B. *Optimization criterion*

In the paper, the Independent Batch Scheduling Problem is considered in order to generate the schedule. For estimating the execution times of tasks on the computational units we based on the Expected Time to Compute (ETC) matrix model [3] which can be defined in the following way:

$$ETC = [ETC[j][i]]^{i=1,...,m}_{j=1,...,n} \tag{7}$$

where

$$ETC[j][i] = wl_j / cc_i \tag{8}$$

in which n and m are denoting the number of tasks and the number of VMs (computational units), respectively, and $ETC[i][j] = NaN$ when $sd_j > tl_i$. The full description of this model can be found in [16], [9], [10], [12].

The final schedule is a schedule with the shortest makespan, that can be calculated as follows [16]:

$$C_{\max} = \min_{S \in Schedules} \left\{ \max_{j \in Tasks} C_j \right\}, \tag{9}$$

where C_j is the completion time of the j-th task, $Tasks$ is the set of tasks in the batch, and $Schedules$ is the set of all schedules which can be generated for the tasks from that batch

C. *Security-aware task scheduling*

Based on the above models, we introduced the following cryptography wrapping model:
1. the computational overhead that influences the task scheduling and includes pre and post-processing security operations. In this case, the bias required to deliver tl_i to task j, that requires a security demand of sd_j, may be estimated in the form of additional computational time (sec.) and denoted by [14]:

$$b_{i,j} = b(sd_j, wl_j, tl_i, cc_i, inputSize_j, outputSize_j) \tag{10}$$

where $inputSize_j$ and $outputSize_j$ are the sizes (in bytes) of the files characterizing the task j and storing the corresponding result, respectively. This value can be approximated by the sum of the number of instructions to compute the cryptographic requirements and the workload.
2. the computational overhead for the security protocols that do not influence the scheduling process, such

as: verifying the integrity of tasks results, ciphering outputs from tasks that will be stored in the data center (before sending the task back to the end user), verifying the digital signature of the end user who wants to collect results from the Cloud Computing system. In this case, the bias required to deliver tl_i to the task j, that requires a security demand of sd_j, may be represented in the form of additional computational time (sec.) and denoted by:

$$B_{i,j} = B(sd_j, wl_j, tl_i, cc_i, inputSize_j, outputSize_j) \quad (11)$$

The ANN described in the previous section works as the computational overhead type 2.

The idea of dynamic hashing is based on the usage of the predicted idle time of the particular VM. The makespan, calculated during tasks processing, defines the time of the last task readiness for the VM that will be calculated without being idle. The VMs that will finish their tasks earlier have to wait for the new batch to be scheduled. The main part of the algorithm is to find the longest hash length that may be calculated by them that will not result in makespan change. Instead of being idle, computational units will be forced to do cryptography security operations.

In the proposed model the time needed for computing the hash value of length n on $i-th$ unit for $j-th$ task can be calculated in the following way:

$$B^n_{i,j} = N(sd_j, wl_j, tl_i, cc_i, inputSize_j, outputSize_j, n)/cc_i \quad (12)$$

where function N calculates the number of operations during computing the hash value of length n.

After the schedule is ready each of VM has his own pool of tasks to be computed. Without the loss of the problem generality, we may assume that task for the unit number j was renumbered as $T_1, T_2, ..., T_{n^j}$ with the workloads of $wl_1, wl_2, ..., wl_{n^j}$. The makespan for this batch of tasks is known (it equals C_{max} - see eq. (9)). Then the idle time for this unit is:

$$T_{idle}(i) = C_{max} - \sum_{j=1}^{n^j} wl_j/cc_i \quad (13)$$

Lets assume that the hash length that is offered by the unit i is from n^i_{min} to n^i_{max} with the reasonable equal step between them of Δ^i bits. Then the next offered hash length is $n^i_{min} + \Delta^i$, and the next is $n^i_{min} + 2\Delta^i$,until $n^i_{min} + K\Delta^i = n^i_{max}$, for $K \in \mathbb{N}$. For example if the VM number 2 is very fast we may assume long hashes: from 512 up to 2*512=1024, with any possible hash length generated with the step of length 100 bits, that is: 512, 612, 712, 812, 912, 1024. In this case: n^2_{min} =512, n^2_{max}=1024, Δ^2=100. If the VM number 10 is rather slow (small cc parameter) we may assume light hashes: from 40 up to 3*80=240, with any possible hash length generated with the step of length 10 bits, that is: 40, 50,...,240. In that case: n^{10}_{min} =40, n^{10}_{max}=240, Δ^{10}=10.

That assumption takes into account the fact that some of the resources may offer very poor computational power and very limited security services.

Two scheduling modules were used:
1. Global scheduler for assigning each task into one of available VMs, for ETC matrix, see eq. (7)-(9).
2. Local scheduler located in particular VM that, basing on the idle time, is assigning one of possible hash length to each task.

D. *Adjusting the hash length to the tasks*

The local scheduler is based on the Expected Operations to Compute matrix:

$$EOC = [E[j][k]]^{k=0,...,K}_{j=1,...,n} \quad (14)$$

where

$$EOC[j][k] = N(sd_j, wl_j, tl_i, cc_i, inputSize_j, outputSize_j, n^i_{min} + k\Delta^i) * \frac{outputSize_j}{n^i_{min} + k\Delta^i} \quad (15)$$

The final schedule is maximizing the overall number of operations spend on hashing, which can be defined as follows:

$$HO = \max_{S \in Schedules^i} \Big\{ \max_{j \in Tasks} HO_j \Big\}, \quad (16)$$

where $HO_j = \max_{k=1,..,K} EOC[j][k]$ is the number of hashing operations for the j-th task, This value is equal to the the maximum number of hashing operations for the j-th task that is not extending the idle time of the the VM. $Tasks$ is the set of tasks in the batch and $Schedules^i$ is the set of all schedules which can be generated for the tasks that should be computed by the unit number i considering the security mapping: $EOC[i][j] = NaN$ for i and j such that $sd_j > tl_i$ The scheduling problem is solved using evolutionary algorithm solution proposed in [13] and [14]. An additional condition to be fulfilled is the makespan constrained:

$$HO/cc_i \leq T_{idle}(i) \quad (17)$$

for all $i = 1, 2, ..m$

V. NUMERICAL EVALUATION

In this section, we describe three different Feed Forward ANNs which were used for hashing purposes. The output hashes were equal to 64 bits, so our idea can be classified as light cryptography which is balancing a compromise between cryptography strength and speed of computation in favor of the speed. For all three ANNs the same input, target, and testing data were used, all were train with Bayesian Regularization usage, and all were implemented in MATLAB.

Next, results of security-aware task scheduling are presented. In that part, we are presenting the application of the proposed model for an exemplary scheduling problem composed of 5 VMs and 25 tasks.

A. Chosen ANNs

We have tested 3 ANNs listed below:

- 2 layered feed-forward artificial neural network with 16 sigmoid neurons in the hidden layer and 64 linear output neurons in the output layer: 64-16-64-64. (n=16, m=64).

This network was trained for 100 epochs and further will be denoted by ANN_1.

- 2 layered feed-forward artificial neural network with 32 sigmoid neurons in the hidden layer and 64 linear output neurons in the output layer: 64-32-64-64. This network was trained for 50 epochs and further will be denoted by ANN_2.
- 2 layered feed-forward artificial neural network with 48 sigmoid neurons in the hidden layer and 64 linear output neurons in the output layer: 64-48-64-64.

This network was trained for 25 epochs and further will be denoted by ANN_3.

B. Statistical tests of hashing functions

We have done three statistical tests on ANNs outputs. All are described below. To determine whether a test is passed, or not we used Z statistics (see eq. 18) with significance level $\alpha = 5\%$ (one of the most popular in hypothesis testing). Thus when $|Z| >= 1.96$ the test is not passed.

$$|Z| = \left| \frac{AVG - \mu}{std} * \sqrt{n} \right|. \tag{18}$$

where AVG is the average value, μ is the expected value, std is the standard deviation and n is the number of elements. Note, that each ANN got $INPUT^{test}$ described in chapter III as an input. $INPUT^{test}$ consisted of 10000 independent vectors, each containing 64 randomly generated bits. Results and conclusion described below were based on $INPUT^{test}$ and all 3 ANNs outputs.

1. Collision Test. One of the most important and simplest test. We have checked whether any of the chosen ANN produced at least once two same hash values (note that in all cases inputs did not repeat). This test was passed by all ANNs - all hashes were different.

2. Series Test. For each ANN and for each hash we performed Wald Wolfowitz series test. The aim of this test was to indicate whether outputs were produced in a random manner, or not. For each ANN we got a 10000 element vector containing Z statistics values. Each Z statistic value indicated whether a particular hash passed the test (was created randomly) accordingly to the significance level set to the 5%, or not.

In a ANN_1 case 381 samples out of 10000 do not pass the test (3.81%). In a ANN_2 it was 440 samples (4.4%) and in a ANN_3 it was 416 samples (4.16%). In all cases more then 95% of hashes were created in a random manner, so all ANNs passed the test.

3. Hamming Distance Test. In this test, we measured the hamming distance between hashes and its corresponding messages for each ANN. Hamming distance in this case can be considered as a number of ones in vector given by formula: $INPUT^{test}[i]$ XOR $OUTPUT_o[i]$. A result is a vector containing 10000 elements from range $[0, 64]$. Expected value μ is equal to 32 (half of the hash size). Results are presented in fig. (2), fig. (3), and fig. (4).

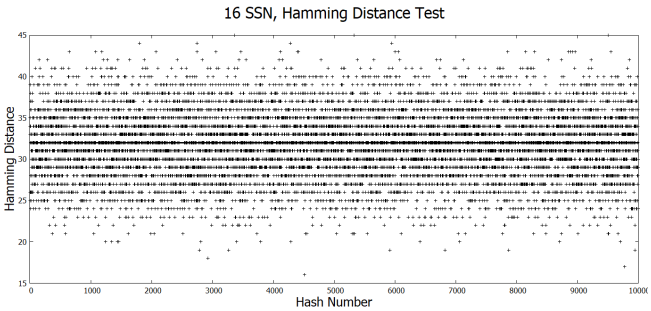

Fig. 2: ANN1 Hamming Distance Test

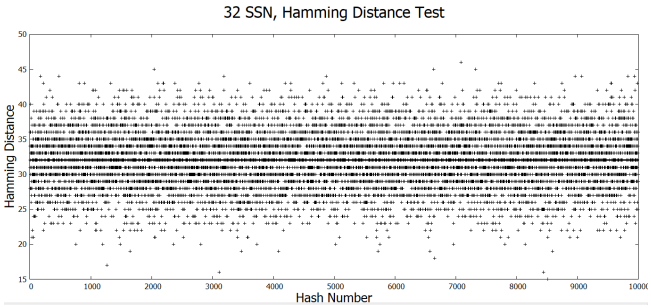

Fig. 3: ANN2 Hamming Distance Test

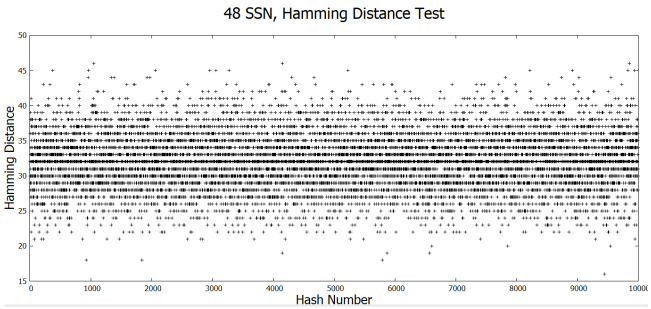

Fig. 4: ANN3 Hamming Distance Test

The $|Z|$ values were equal to 6.49 for ANN_1, 1.14 for ANN_2 and 0.73 for ANN_3. ANN_1 do not passed this test, so is disqualified. Z statistics showed, that hash corresponding to the input message don't differ enough.

4. Bit Prediction Test. The aim of this test was to measure whether a particular bit of output hash can be predicted, or not. We calculated the probability of '1' on a particular hash position accordingly to eq. (19).

$$P_j^o(1) = \frac{\sum_{i=1,...,10000} OUTPUT_o[i][j]}{10000}, \quad j = 1, ..., 64, \tag{19}$$

where o denotes chosen ANN. In every case, we reached vector containing 64 elements indicating the probability

of '1' on the particular position of the hash. Expected value is equal to 0.5. Results are presented in fig. (5) and fig. (6).

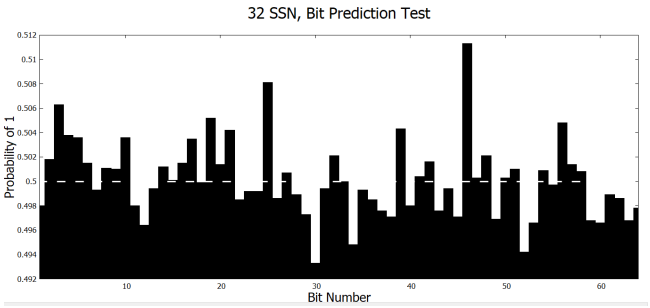

Fig. 5: ANN2 Bit Prediction Test

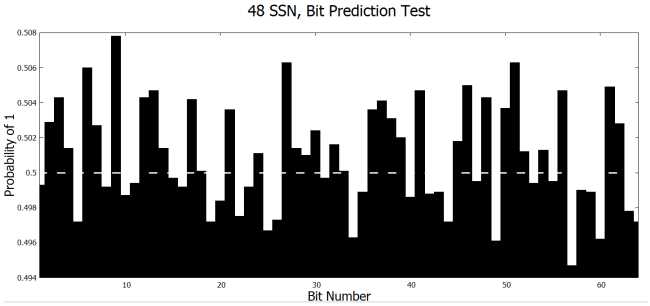

Fig. 6: ANN3 Bit Prediction Test

The $|Z|$ values were equal to 0.3 for ANN_2 and 2.33 for ANN_3. ANN_3 do not pass the test. The Z statistic indicates that particular bits of ANN_3 can be predicted, thus it is not a good candidate for a hashing function.

C. Security aware scheduling results

The last stage of the research is security aware task scheduling combined with the dynamic ANN-based Light Hashing Function. First, we generated an example schedule for 5 VMs and 25 tasks. Tab. 2 presents ready to schedule for batch composed of independent tasks, characteristics of tasks and VMs, main objective - makespan, and the idle time for each VM. The table omits information about the security demand and trust level parameters - the generated schedule meets the requirements imposed by these parameters.

To illustrate the local scheduling, let's consider the first VM. Considered hashing parameters was the following: $n^1_{min} = 64$, n^1_{max}=128, Δ^1=64. the number of hashing operation was measured experimentally. Hashing of the basic 64-bit string using 64 length hash consumes 0.554920282[GFLO] and hashing of the basic 128-bit string using 128 length hash consumes 0.746556553[GFLO].

3 tasks were scheduled into for VM number 1, with outputs of $outputSize_1 = 18819$, $outputSize_2 = 90782$, $outputSize_3 = 67113$ [FLOP]. Therefore using shorter hashing, assuming task 1 characteristics, for k=1, j=1 we obtain:

$$N(sd_1, wl_1, tl_1, 2.26, 18819, 18819, 64) * \frac{18819}{64} =$$

$$0.554920282[GFLO] * 294.046875 = 163.1725[GFLO]$$

Using longer hashing,for k=2, we obtain:

$$N(sd_1, wl_1, tl_1, 2.261, 18819, 18819, 128) * \frac{18819}{128} =$$

$$0.746556553[GFLO] * 147.0234 = 109.7612[GFLO]$$

Taking into account the rest of tasks, assigned into VM no. 1 , we may calculate the Expected Operations to Compute matrix:

$$EOC = \begin{bmatrix} 163.17 & 109.76 \\ 787.13 & 529.48 \\ 581.91 & 391.43 \end{bmatrix}$$

assigning the strongest hash into all three tasks gives 109.76+529.48+391.43=1030.67[GFLO] of additional computing operations. Assuming, $cc_1 = 2,26[GFLOPS]$, it results with 456.04[sec.] spend on hashing. This VM have a margin time of 4140.25[sec.], see tab. 2, therefore even stronger hashing may be used.

VI. SUMMARY

In this paper, we presented an intelligent system for a generation of the cryptographic hashes enabling to compute the fingerprints with different lengths.

All ANNs produced hashes in a random manner and no collisions were found. However, the hamming distance test eliminated ANN_1 and the bit prediction test eliminated ANN_3. ANN_2 seems to have very good results and can be used in our model for hashing purposes. We proposed a very scalable algorithm which can produce a hash of any length.

Additionally, we presented the method for incorporating such dynamic hash functions during predicted idle time of the VMs. The algorithm is build of two levels of scheduling. One is the global level, assigning the tasks into VMs. The second one is a local scheduler. It is operating for each of the VMs separately. It enables to match the task number with the desired hash length so that not to increase the makespan value of the considered batch of tasks.

Further investigations can cover more ANNs types (NARX networks, Recurrent Networks, Time Delayed Networks, Deep Networks), more ANNs configurations (layers, neuron numbers, training methods, epochs), and more tests. The aim will be to improve security, to improve speed, or both at the same time. The biggest challenge in the future may be testing a first and a second preimage resistances of possible hashing functions.

ACKNOWLEDGEMENT

Research presented in the paper was partly supported by the Polish Ministry of Science and Higher Education through the Faculty of Physics, Mathematics and Computer Science (Cracow University of Technology), grant no. F3/597/2018/DS–M.

TABLE 2: Exemplary schedule for 5 VMs and 25 tasks.

VM ID (CC [GFLOPS])	Schedule - tasks [GFLO]	Idle time (sec.)
2 (2.26)	1903.01, 3502.5, 1043.63 (3 tasks)	4140.35
1 (6.27)	1797.06, 6470.2, 19845.5, 2543.63 (4 tasks)	2104.58
0 (5.71)	3219.49, 7995.67, 3961.75, 17347.6 (4 tasks)	1297.89
3 (6.92)	1190.81, 9009.31, 5718.46, 8000.06, 17162.7, 1248.94 (6 tasks)	876.86
4 (5.57)	1842.05, 10536.4, 7979.65, 1762.2, 2241.56, 5104.97, 2741.67, 6747.83 (8 tasks)	0.0
Makespan: 6993.96 sec.		

References

[1] SECURE HASH STANDARD . `https://csrc.nist.gov`.

[2] SHA-3 Standard . https://nvlpubs.nist.gov.

[3] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali. Task execution time modeling for heterogeneous computing systems. In *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556)*, pages 185–199, 2000.

[4] E. Barbierato, M. Gribaudo, M. Iacono, and A. Jakóbik. Exploiting cloudsim in a multiformalism modeling approach for cloud based systems. *Simulation Modelling Practice and Theory*, 93:133 – 147, 2019. Modeling and Simulation of Cloud Computing and Big Data.

[5] M. Cococcioni. Mackey-glass time series generator. `https://www.mathworks.com/matlabcentral/fileexchange/24390-mackey-glass-time-series-generator`. Accessed: 04.2019.

[6] Y. Du, G. He, and D. Yu. Efficient hashing technique based on bloom filter for high-speed network. In *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, volume 01, pages 58–63, Aug 2016.

[7] D. Fernndez-Cerero, A. Jakbik, D. Grzonka, J. Koodziej, and A. Fernndez-Montes. Security supportive energy-aware scheduling and energy policies for cloud environments. *Journal of Parallel and Distributed Computing*, 119:191 – 202, 2018.

[8] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[9] D. Grzonka. *Intelligent Agent-based Monitoring Systems of Task Scheduling for Distributed High-Performance Environments*. PhD thesis, Institute of Fundamental Technological Research Polish Academy of Sciences, 2018. (in Polish).

[10] D. Grzonka, A. Jakbik, J. Koodziej, and S. Pllana. Using a multi-agent system and artificial intelligence for monitoring and improving the cloud performance and security. *Future Generation Computer Systems*, 86:1106 – 1117, 2018.

[11] A. Jakóbik. Big data security. In F. Pop, J. Kołodziej, and B. Di Martino, editors, *Resource Management for Big Data Platforms: Algorithms, Modelling, and High-Performance Computing Techniques*, pages 241–261, Cham, 2016. Springer International Publishing.

[12] A. Jakóbik, D. Grzonka, and J. Kołodziej. Security supportive energy aware scheduling and scaling for cloud environments. In *European Conference on Modelling and Simulation, ECMS 2017, Budapest, Hungary, May 23-26, 2017, Proceedings.*, pages 583–590, 2017.

[13] A. Jakóbik, D. Grzonka, J. Kołodziej, and H. González-Vélez. Towards secure non-deterministic meta-scheduling for clouds. In *30th European Conference on Modelling and Simulation, ECMS 2016, Regensburg, Germany, May 31 - June 3, 2016, Proceedings.*, pages 596–602, 2016.

[14] A. Jakóbik, D. Grzonka, and F. Palmieri. Non-deterministic security driven meta scheduler for distributed cloud organizations. *Simulation Modelling Practice and Theory*, 76:67 – 81, 2017.

[15] M. Kido and R. Dobai. Evolutionary design of hash functions for ip address hashing using genetic programming. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1720–1727, June 2017.

[16] J. Kołodziej. *Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems*. Springer Publishing Company, Incorporated, 2012.

[17] M. Singh and D. Garg. Choosing best hashing strategies and hash functions. In *2009 IEEE International Advance Computing Conference*, pages 50–55, March 2009.

[18] J. Tchórzewski and A. Jakóbik. Theoretical and experimental analysis of cryptographic hash functions. Journal of Telecommunications and Information Technology, 2019.

[19] J. Wang, T. Zhang, j. song, N. Sebe, and H. T. Shen. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):769–790, April 2018.

[20] G. Z. Survey on lightweight hash functions. , 3(1):1, 2016.

AUTHOR BIOGRAPHIES

JACEK TCHÓRZEWSKI received his B.Sc. and M.Sc. degrees with distinctions in Computer Science at Cracow University of Technology, Poland, in 2016 and 2017, respectively. Currently he is Research and Teaching Assistant at Cracow University of Technology and Ph.D. student at AGH Cracow Univeristy of Science and Technology. His e-mail address is: jacek.tchorzewski@onet.pl

AGNIESZKA JAKÓBIK (KROK) received her M.Sc. in the field of stochastic processes at the Jagiellonian University, Cracow, Poland and Ph.D. degree in the field of neural networks at Tadeusz Kosciuszko Cracow University of Technology, Poland, in 2003 and 2007, respectively. From 2009 she is an Assistant Professor at Faculty of Physics, Mathematics and Computer Science, Tadeusz Kosciuszko Cracow University of Technology. Her e-mail address is: ajakobik@pk.edu.pl

DANIEL GRZONKA received his B.Eng. and M.Sc. degrees from Cracow University of Technology, Poland, in 2012 and 2013, respectively. In 2019 he received his Ph.D. degree from the Polish Academy of Sciences (in cooperation with Jagiellonian University). All degrees (with distinctions) are in Computer Science. Currently, he is an Assistant Professor at the Cracow University of Technology. The main topics of his research are monitoring systems, grid and cloud computing, multi-agent systems, task scheduling problems, data mining and high-performance computing. For more information, please visit: www.grzonka.eu