High-Performance Intrusion Response Planning on Many-Core Architectures

Stefano Iannucci
Distributed Analytics and Security Institute
Mississippi State University
Starkville, Mississippi
Email: stefano@dasi.msstate.edu

Qian Chen
Department of Engineering Technology,
Computer Science Technology
Savannah State University
Savannah, Georgia
Email: chenq@savannahstate.edu

Sherif Abdelwahed
Electrical and Computer Engineering
Mississippi State University
Starkville, Mississippi
Email: sherif@ece.msstate.edu

***Abstract*—The quantity and sophistication of cyber attacks have increased year by year, thus it is infeasible to manually process Intrusion Detection Systems (IDSs) alerts. Intrusion Response Systems (IRSs) extend IDSs by providing automatic protection mechanisms. The core of an IRS is its planning algorithm, in charge of selecting the best response action to counter the detected attacks. However, the planning algorithm has to be carefully designed and implemented in order to exhibit a low overhead and not to compromise the scalability of the protected system. In this paper we present the performance evaluation of an IRS based on Markov Decision Process (MDP), which leverages many-core co-processors. Such an IRS produces optimal long-term response policies evaluated according to a multi-criteria objective function. We show that, despite the complexity of the MDP modeling, the proposed IRS is able to protect large systems while introducing little to no overhead on the protected hosts.**

I. Introduction

Recently, massive attacks directed to computer systems are becoming more sophisticated. According to the Akamai's state of the Internet 2015 Q2 Report [1], the number of recorded attacks is more than doubled compared with a year ago. Half of detected Distributed Denial of Service (DDoS) attacks were launched by multi-vector attacks. The complexity of modern attacks and computing systems attacks targeting at makes it difficult and sometimes infeasible to manage through manual intervention, even if they are readily detected and signaled by an Intrusion Detection System (IDS). Therefore, there is a strong need to develop more effective approaches for countering attacks through an automatic Intrusion Response System (IRS) aimed at detecting and mitigating cyber attacks with little or no human intervention.

In the early 2000s, most IRSs were implemented using a static mapping between the detected attacks and a list of perspective responses [24]. When using a static mapping, the responses used to protect against certain attacks remain the same until intrusions expose the inadequacy of current protection mechanisms. System administrators must therefore manually upgrade the current response mechanisms periodically. This manual upgrading causes a significant delay of protection. Research on automatic IRSs therefore shifted on dynamic IRSs, that are capable of deciding which action to take according to the detected attacks types, to the set of candidate responses and to a list of possibly conflicting criteria (e.g., response time and cost). Several decision models for dynamic IRSs have been proposed (e.g. [17], [24], [21], [9]), but most of them do not consider the notion of *system state*, which is fundamental to compute a measure of the current hazard level of the system. Knowing the system state allows the IRS to either select the best immediate action to take, or to plan for a long-term response policy, that is, a sequence of actions that is able to drive the system back to normal conditions.

Unfortunately, when system states are considered in the decision making process, the IRS has to deal with a state space cardinality exponential in the number of the attributes used to describe the protected system. Since timeliness is always crucial for a defense action to be effective [4], an efficient implementation of the planning algorithm is required for the IRS to protect large systems. In this work we evaluate the planning performance of a Markov Decision Process (MDP) [18] based IRS. We show a comparison between the planning time obtained using the optimal Value Iteration (VI) algorithm [5] and the sub-optimal rollout-based Monte-Carlo planning algorithm named UCT [15], both of them implemented in the state of the art BURLAP [2] library. Furthermore, as the main contribution of the paper, we propose, to the best of our knowledge, the first multi-threaded implementation of VI specifically optimized to run on the Intel MIC architecture [10]. We show that the execution of the VI planner on Intel Xeon Phi provides 2x speedup over its execution on a standard Intel Xeon platform and that the automatic vectorization provided by the `icc` compiler plays a fundamental role in increasing the performances. The evaluation of the effectiveness of the response policies planned by the decision making algorithm is out of the scope of this paper. The interested reader can find a thorough effectiveness evaluation in our previous work [12].

The paper is organized as follows: Section II discusses related works on performance evaluation of IRSs; in Section III we provide a background on MDP and on VI; in Section IV we show how MDP can be used to model a system protected by the IRS; in Section V we briefly introduce the testbed and the Intel MIC architecture and then we compare the performance of (i) the single-threaded VI and UCT algorithms

and (ii) of the multi-threaded VI algorithm executed on Intel Xeon CPUs and on Intel Xeon Phi, comparing the latter with a non-vectorized version. Section VI concludes the paper and outlines future works.

II. Related Works

In this section we describe relevant works in the field of dynamic IRS with a specific focus on decision making and performance evaluation. Not many works in literature employ a stateful decision making process and very few of them provide a performance evaluation. All the reviewed IRSs focus either on network-based response actions such as blocking or throttling the offending source IP address or on system-based response actions such as system rebooting or data migration. The former typology provides planning times in the order of milliseconds because they use a very limited set of response actions and have to be compatible with a deployment in-line with the network flow. The latter are instead higher-level IRSs, which are not deployed in-line with the network flow. Their planning times are in the order of seconds or minutes because thet must evaluate a possibly very large set of different conflicting response actions deployable on a possibly large set of system components, including eventual firewalls. Moreover, the available response actions in this latter case usually have an execution time of minutes (as in the case of the system reboot) or hours (as in the case of the data migration).

In [24] the authors represent a computer system by modeling: (i) the executed services; (ii) the system users; (iii) the network topology; (iv) the firewall rules. The model is used to find, given a dependency tree between the entities, the best firewall rule to apply in case of a detected attack. The best response is selected considering that very often the application of an additional firewall rule may interfere with the normal system operativeness. Therefore the response which provides the lowest penalty to the normal operativeness is applied. A performance assessment on the realized prototype revealed an optimization time of 34 seconds for a model composed of 35 resources distributed over five subnets.

A Partially Observable MDP (POMDP) is used in [27] to model a IRS able to plan optimal response policies. Since the POMDP is subject to an exponential growth of the states according to the number of the considered attributes, the authors propose a hierarchical decomposition to reduce the computational complexity. This work, however, does not fully exploit the potentialities offered by the MDP framework; the response actions rewards are statically defined (therefore simulating a static attack-response mapping) and long-term plans are not considered. The performance of the proposed approach are evaluated considering spaces characterized by up to 100 attributes. The authors show that the proposed hierarchical decomposition is able to reduce the planning time of 50% in the considered scenario.

In [17] the authors evaluate benefits and risks of a reaction, together with potential damages caused by the attack in case of no reaction. Penalty costs are modeled as Service Level Agreement costs related to the importance of a provided service. The performance of the developed prototype are measured in terms of the time needed to parse the alert generated by the IDS and to select the optimal response action among a set of three (none, rate-limit IP, block IP), for which it takes 0.1 seconds for a system of any size.

ADEPTS [11] focuses on restricting the effect of the intrusion to a subset of the services by maximizing the availability of the system at the expenses of the features compromised by the attack, which are isolated from the rest of the system.

In [23] the authors propose an IRS that takes into consideration the stochastic nature of the detections made by the IDS and the response action is only triggered if the confidence level of the detected attack is greater than a specified threshold. Response actions are manually mapped to known attack patterns and the best response action is chosen based on its impact on the system.

In [26] the authors introduce the concept of system state using an MDP for the response selection process of an autonomic IRS. However, they do not try to solve the MDP in order to obtain the optimal long-term response policy, rather they define an utility function that, given the current state and an action to evaluate, compares each possible future state reacheable by executing the considered response action. This approach can only guarantee local optimal response action selection because the look-ahead is limited to a single action and therefore it does not take full advantage of a stateful model. The work has been then extended in [25] with the concrete implementation of the approach, but the MDP model has been replaced with a stateless utility function.

Neither [11], nor [23], nor [26] provide any performance evaluation of the proposed approach.

III. MDP Background

A Single-Agent Discrete-Time MDP is a stateful and probabilistic approach to model the behavior and the run-time dynamics of a system. An MDP [5] is a tuple $\langle S, A, P, R, \gamma \rangle$.

The symbol S represents the state space that the agent can navigate and $s_k \in S$ represents the agent state at discrete time k. A common practice when programming MDPs is to characterize each state with a number of attributes so that a specific attribute configuration univocally maps with a given state. A is the finite set of actions available to the agent to navigate the state space. Specifically, by executing at time k an action $a_k \in A$ in the current state $s_k \in S$, the agent moves to a successor state $s_{k+1} \in S$. The transition dynamics from the current to the next state are given by the transition probability function P. This function specifies, for each source state $s_k \in S$, for each destination state $s_{k+1} \in S$, and for each action $a_k \in A$, the value $P(s_k, a_k, s_{k+1})$, that is, the probability value that by executing the action a in state s at time k, the resulting state will be s_{k+1}. Every time an action is executed, the MDP agent is rewarded with a bonus (or penalized with a cost), according to the reward function R. That is, $R_k = R(s_k, a_k, s_{k+1})$ represents the reward that the agent will earn (or the cost the agent will pay) for executing at time k the action a in state s_k and being taken to some

state s_{k+1}. Some MDP models use a different reward function, based only on s_k and a_k and not considering s_{k+1}. Since the entire model is probabilistic, it usually makes sense to give more reward to actions executed in the short-term period than to actions executed in the long-term period because the former are more likely to be executed, while the execution of the latter is dependant on the achievement of the state in which they are executable. To this end, MDPs are characterized by a γ discount factor, usually defined in the interval $[0, 1]$, which specifies how much short-term rewards should be preferred over long-term rewards.

The overall behavior of the agent is described by a deterministic or stochastic policy π. When π is deterministic it univocally specifies, for each s_k, the action a_k that the agent must execute. When π is probabilistic it specifies a probability distribution such that $\pi : S \times A \rightarrow [0, 1]$. The objective of the agent is to find a policy π^* such that the discounted reward

$$R_k = \sum_{j=0}^{\infty} \gamma^j R_{k+j+1} \tag{1}$$

is maximized.

Several optimal and sub-optimal algorithms for solving MDPs have been proposed (e.g., [5], [15], [20], [16], [14]), but one of the most commonly used remains the *Value Iteration* (VI) algorithm [5] because of its simplicity. It is based on the concept of *state-value* function $V_\pi(s_k) = \mathbb{E}_\pi[R_k|s_k]$, that is, the expected reward achievable by the agent starting from state s_k and then following policy π. The base step of the algorithm is to assign an initial random state-value V^0 to all the states and then to execute the iterative refinement process described in [6]:

$$V^{i+1}(s_k) = \max_{a_k \in A} R(s_k) + \gamma \sum_{s_{k+1} \in S} P(s_k, a_k, s_{k+1}) V^i(s_{k+1}) \tag{2}$$

The sequence of functions V^i converges linearly to the optimal value V^* in the limit and provides thus the expected maximum reward obtainable by following the optimal policy π^* from state s_k.

IV. System Model

The MDP framework can be used to describe a system and its behavior when subject to control actions planned by an IRS. The system is indeed statically described by its corresponding MDP's state space and dynamically described by the MDP's actions and associated transition matrix, that rule how the system evolves over time. The MDP reward, as well as the γ parameter, do not take part in the system description, rather they are used at planning time in order to choose the best response action to execute on a given state.

We model a system state by joining two macro-attributes: (i) the current attack vector **p** and (ii) the system variables **v**. The first contains as many variables as the number of attacks detectable by the attached IDSs and each variable $p_i \in \mathbf{p}$ represents the probability value that the system is currently under attack i, while the second represent the current system status. We characterize each action $a_k \in A$ with pre-conditions and post-conditions: the former are boolean expressions on the state attributes, that is, they identify a subset of the state space in which the actions are executable; the latter define instead a probability distribution over the possible next states reachable by the agent after the execution of the action.

Upon the execution of an action a_k, the MDP agent obtains a reward. We define the reward function as a penalty score on the executed actions. The reward function evaluates the response actions according to the following criteria:

- **Response Time** $T(a_k) \in \mathbb{R}$**.** This criteria represents the time needed to apply the response a.
- **Cost** $C(a_k) \in \mathbb{R}$**.** This criteria represents the economic cost of applying the response a.
- **Impact index** $I(a_k) \in [0, 1]$**.** This criteria represents the impact index of the response a_k on the normal system operativeness. The lower its value is, the lower is the impact on the system.

We define the following reward function:

$$R_k = -w_r \frac{T(a_k)}{T_{max}} - w_c \frac{C(a_k)}{C_{max}} - w_i I(a_k) \tag{3}$$

where $w_r, w_c, w_i \in [0, 1]$ are custom weights used to balance the importance of the criteria in the multicriteria optimization problem. T_{max} and C_{max} represent respectively the maximum response time and the maximum cost over all the considered response actions and are used to normalize their values.

The planning process stops when the agent reaches a state belonging to the subset of the target states $S_{tgt} = \{s|F = true\}$, where F is a termination function expressed as a boolean condition on the state attributes.

V. Performance Evaluation

VI is one of the mostly used algorithms to plan an optimal policy for Single-Agent and Multi-Agent MDPs [6], [8]. It produces successive approximations of the optimal value function until the expected objective value is stable for all the MDP states. Unfortunately, even if each iteration can be performed in $O(|A||S|^2)$ steps [13], the number of states composing the MDP grows exponentially with the number of the defined attributes. For this reason, carefully designing and developing the response selection algorithm is fundamental in order for the IRS to be able to protect large systems. When realizing a system model as described in Section IV, even a relatively small system could require hundreds of attributes, resulting therefore in a hardly feasible optimization problem. However, we observe that not necessarily the entire set of attributes and the entire set of actions must be included when instantiating the MDP problem: while countering any threat, indeed, we only need to consider only the attributes and the actions that, directly or indirectly, help in facing the threat. The rationale is that specific threats are supposed to impact only specific system attributes and therefore the IRS should instantiate only the minimal MDP required to counter the detected attack.

In this section we first describe the testbed on which we ran the experiments and how we generated the experimental MDP; then, we compare the performance of a single-threaded VI planner with the performance of a single-threaded sub-optimal rollout-based Monte-Carlo planning algorithm named UCT [15], both of them implemented in the state of the art BURLAP [2] library; finally we propose, to the best of our knowledge, the first multi-threaded implementation of the VI algorithm specifically optimized to run on the Intel Xeon Phi (Intel MIC) platform. The latter exploits the massive parallelism of the Intel MIC architecture and leverages its Vector Processing Unit (VPU).

A. Testbed

The experiments have been run on a single compute node of the Shadow supercomputer at Mississippi State University. Each compute node of Shadow is characterized by 2 10-core Intel Xeon E5-2680v2 CPU running at 2.8 Ghz, 512 GB of RAM and 2 Intel Xeon Phi accelerators 5100 series. The latter has 61 x86 architecture processor cores running at 1.05 Ghz. Each core contains a VPU, which has 32 512-bit vector registers [10]. One VPU can execute 16 single-precision (SP) or 8 double-precision (DP) operations per cycle. The VPU can also execute 32 SP and 16 DP under fused multiply-add instructions. Because of the new design, the VPUs are very power efficient for HPC workloads [22]. Each core has two-level caches. One 32KB L1 cache for data and instruction, and a 512KB L2 cache. Although Intel Xeon Phi does not provide shared cache between the cores, it supports hardware-based cache coherency. All L2 caches of the 61 cores are interconnected to each other, and these caches are also interconnected with eight GDDR-5 memory controllers by a ring bus. Therefore, the last-level cache of the Intel Xeon Phi is up to 32MB.

Each Intel MIC is equipped with its own Linux-based operating system and it is therefore accessible through SSH like a standard Linux Machine. Two programming paradigms are supported by Intel MIC: *native mode* and *offload mode*. A native mode application is run directly on the Intel MIC: usually the executable is compiled with the `-mmic` option and then copied on the MIC board, where it is executed. In this case only MIC's resources are used by the application, without interfering with the activities on the host. Offload mode, instead, is used to realize hybrid applications able to exploit both the host's CPUs and the installed Intel MICs. Unlike the native mode, in this case the executable is run on the host and only properly annotated code fragments are executed on the MIC. Application code and application data is transferred from the host to the MIC and vice-versa using the PCI-Express bus.

The experimental MDP has been built reflecting a system characterized by up to 1000 boolean state attributes and up to 1000 response actions. Each action is tied to one attribute and it changes its boolean value when executed, in order to generate the full state space. The termination condition is based on an additional $termination$ attribute that can be set to $true$ by any action with probability $1/10$. The reward function assigns the reward -1 to the actions with an even index and -2 to the actions with an odd index.

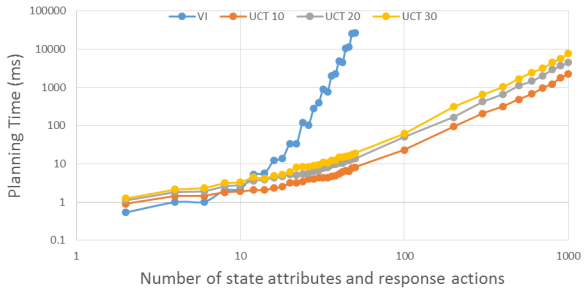

Fig. 1. VI and UCT Planning Time Comparison

Boolean attributes are managed as a special case of numeric attributes, with only two possible values (i.e., 0 and 1). Our approach is based on a discrete-space MDP and therefore it is possible to use both bounded and unbounded, natural and real numeric attributes as long as the values assigned to them belong to a finite set. Otherwise, the state space of the MDP problem would be infinite and the resulting MDP would be a continuous state MDP. As a consequence, different planning algorithms that require the computation of continuous integrals over the state space such as the Point-Based Value Iteration introduced in [19] must be used. However, an exact optimal solution can be computed only when a closed form of the integrals can be found, and this only happens for some special problems, e.g., if all the α-functions of the Partially Observable MDP (POMDP) are linear combinations of Gaussian distributions [7], [19]. In the other cases, only approximated optimal solutions can be computed.

B. State of the Art Performance Evaluation

Figure 1 compares the planning time of the VI algorithm configured with $\gamma = 0.9$ with the UCT algorithm configured to perform 10, 20 or 30 rollouts and with a look-ahead of 10. Results highlight that VI's planning time for systems with 50 state attributes and 50 response actions is 71 seconds. By contrast, the UCT algorithm configured with 10 rollouts is able to plan a policy in less than 2 seconds for a system with 1000 state attributes and 1000 response actions. However, the response policies computed by UCT are not always optimal. Figure 2 compares the obtained rewards. The average reward obtained by VI is close to 10, specifically 10.07. This happens because it always chooses the best response actions and on the average 10 response actions executions are needed to take the modeled system to the terminal state. By contrast, the UCT algorithm with 30 rollouts produces an average reward of 10.86, exhibiting therefore a reward gap of 8.6% in the average. Since such a reward gap is not always tolerable when planning policies for security purposes, we study in the next section the performance of a parallel implementation of the optimal VI algorithm.

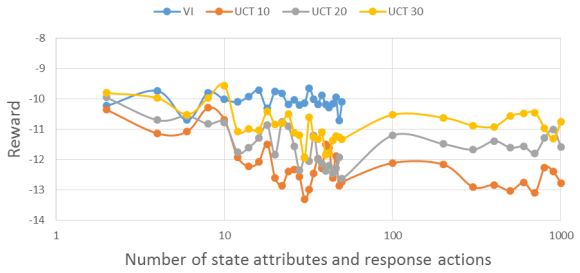

Fig. 2. VI and UCT Rewards Comparison

C. Intel Xeon and Xeon Phi Performance Comparison

We discuss the performance of a VI planner written in C and compiled with the `icc` compiler using the `-O3` option in order to leverage all the available optimizations. The core of the application is the implementation of the right sum of Equation 2, which is in part realized as shown in Figure 3.

```
1: for (i = 0; i < next_state_size; i++) {
2:   __assume_aligned(action.probs, 64);
3:   __assume_aligned(action.rewards, 64);
4:   qs[i] = action.probs[i]*
     (action.rewards[i] + GAMMA * v[i]);
5: }
```

Fig. 3. Implementation of part of the left sum of Equation 2

The particularity of this code fragment resides in the usage of the `__assume_aligned()` directives, that inform the `icc` compiler that the base addresses of the variables `action.probs` and `action.rewards` are aligned to a multiple of 64 bytes, which is required in order to exploit vectorization. Such variables are indeed allocated with the Intel's specific routine `_mm_malloc()`, which allocates memory starting with a base address multiple of a specified parameter.

Figure 4 shows the performance comparison of the multi-threaded VI solver executed on the CPUs of a Shadow's compute node and on one of its Intel MICs in native mode with an input problem consisting of 70 attributes and 70 response actions. This is the biggest problem instance solvable with the 8GB of RAM installed on the Phi 5110. The CPU-based

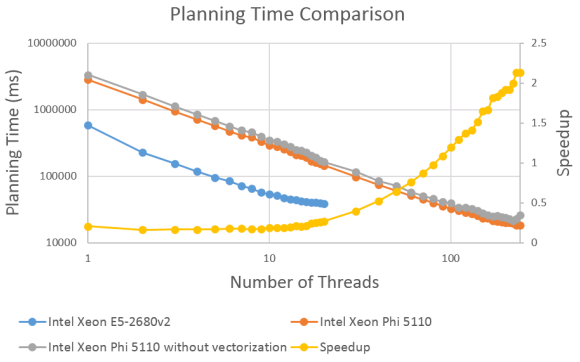

Fig. 4. Intel Xeon and Xeon Phi Planning Time Comparison

```
remark #15399: vectorization support: unroll factor set to 4
remark #15301: FUSED LOOP WAS VECTORIZED
remark #15475: --- begin vector loop cost summary ---
remark #15476: scalar loop cost: 20
remark #15477: vector loop cost: 10.000
remark #15478: estimated potential speedup: 7.270
remark #15479: lightweight vector operations: 14
remark #15488: --- end vector loop cost summary ---
```

Fig. 5. Vectorization Report

execution has been run with a number of threads ranging from 1 to 20 because 20 cores are available on a Shadow's compute node, while the Phi-based execution has been run with a number of threads ranging from 1 to 240 because each Intel Xeon Phi 5110 board is characterized by 61 cores featuring 4 threads each. Intel's best practices suggest to use up to 60 cores, so that a single core can be reserved for the operating system. The speedup has been computed as CPU_t/PHI_t, where CPU_t is the planning time obtained by executing the application on the CPU with t threads and PHI_t is its Intel Xeon Phi's counterpart with vectorization. For $t > 20$, we used the formula CPU_{20}/PHI_t. It is easy to see that CPU always provides better performance in its range of operativeness and it is able to compute an optimal response policy in almost 40 seconds when executed with 20 threads, while it requires 579 seconds when executed with 1 thread. This is due to its higher clock frequency (2.8 Ghz versus 1.05 Ghz) and to its support to Single Instruction Multiple Data (SIMD) with the 256-bit Advanced Vector Extensions (AVX).

The breakeven point (corresponding to 1x speedup) between the two architectures is reached with 80 threads on the Xeon Phi, but the latter provides a 2x speedup over the 20-thread execution on CPU, resulting in a planning time of 18 seconds, when the application is executed with 240 threads.

Figure 5 shows a portion of the vectorization report generated by `icc`, regarding the code of Figure 3. The compiler computed a scalar loop cost of 20, a vector loop cost of 10 and an estimated potential speedup obtainable vectorization speedup of 7.27. However, the used version of `icc` has a bug related to the speedup estimation: instead of computing it as the ratio between the scalar cost and the vector cost, the latter is multiplied by the unroll factor [3]. Therefore, the estimation of the speedup would be 1.82 in our case. In order to verify the estimation, we compiled the application for Intel MIC explicitly disabling the vectorization features. Results shown in Figure 4 show that the vectorized Phi implementation is in the average 20% faster than the non vectorized implementation. Such a gap with the predicted speedup, as explained by Intel in [3], is due to the fact that the speedup prediction is a very rough estimate, aimed only at deciding whether it is useful to vectorize or not.

VI. Conclusion and Future Work

During the last decade we saw the evolution of two generations of IRSs. The first generation was based on a static mapping between attacks and responses, while the second generation aimed at dynamically selecting the best response

given the current attack and a set of possibly conflicting criteria. However, with the growth of Internet, the first generation approaches immediately resulted to be impracticable because of their inherent inflexibility. There is instead still some work ongoing on the second generation approaches, even if they have a limited applicability because they do not consider the concept of system state.

In this paper we presented a performance evaluation of a third-generation, stateful approach to IRS decision making based on MDP, with specific focus on the offloading of the planning algorithm to Intel MIC co-processors. We have shown that the state of the art implementation of common MDP solvers is not adequate to solve large MDP instances and that co-processor offloading offers an effective way to increase the performance and at the same time to reduce the load on the system's CPU, that can be completely dedicated to the traditional workloads. Moreover, we have shown that the `icc` automatic vectorization feature improved the performance of the planner in the order of 20%. The proposed approach is able to plan optimal response policies in a reasonable amount of time for attacks impacting up to 70 system attributes and that require a combination of up 70 response actions in order to be handled.

As a future work we plan to extend the planner by developing an hybrid version able to exploit both the host's CPU and up to two concurrent Intel MICs installed on the same host. Furthermore, we plan to compare the obtained results to GPU-based many-core co-processors and we plan to implement a parallel version of the VI algorithm for Multi-Agent MDPs. Finally, in order to deal with the exponential growth of the state space, a complementar approach is to hierarchically decompose the MDP in sub-problems like in [27].

ACKNOWLEDGEMENT

Funding for this work was (partially) provided by the Pacific Northwest National Laboratory, under U.S. Department of Energy Contract DE-AC05-76RL01830.

REFERENCES

[1] Akamai's state of the internet: Q2 2015 report. https://www.akamai.com/us/en/our-thinking/state-of-the-internet-report.

[2] Brown-umbc reinforcement learning and planning (burlap). http://burlap.cs.brown.edu/.

[3] Vectorization potential speedup calculation. https://software.intel.com/forums/intel-c-compiler/topic/560247.

[4] N. B. Anuar, M. Papadaki, S. Furnell, and N. Clarke. An investigation and survey of response options for intrusion response systems (irss). In *Information Security for South Africa (ISSA), 2010*, pages 1–8. IEEE, 2010.

[5] R. Bellman. Dynamic programming. princeton, nj: Princeton universitypress. *BellmanDynamic Programming1957*, 1957.

[6] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210. Morgan Kaufmann Publishers Inc., 1996.

[7] S. Brechtel, T. Gindele, et al. Solving continuous pomdps: Value iteration with incremental learning of an efficient space representation. In *Proceedings of the 30th International conference on machine learning*, pages 370–378, 2013.

[8] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172, 2008.

[9] Q. Chen, S. Abdelwahed, and A. Erradi. A model-based validated autonomic approach to self-protect computing systems. *Internet of Things Journal, IEEE*, 1(5):446–460, 2014.

[10] J. Fang, H. Sips, L. Zhang, C. Xu, Y. Che, and A. L. Varbanescu. Test-driving intel xeon phi. In *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*, pages 137–148. ACM, 2014.

[11] B. Foo, Y.-S. Wu, Y.-C. Mao, S. Bagchi, and E. Spafford. Adepts: adaptive intrusion response using attack graphs in an e-commerce environment. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 508–517. IEEE, 2005.

[12] S. Iannucci and S. Abdelwahed. A probabilistic approach to autonomic security management. In *Proceedings of the 13th IEEE International Conference on Autonomic Computing (ICAC)*, 2016.

[13] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, pages 237–285, 1996.

[14] M. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.

[15] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.

[16] L. Li, M. L. Littman, and L. Littman. Prioritized sweeping converges to the optimal value function, 2008.

[17] S. Ossenbuhl, J. Steinberger, and H. Baier. Towards automated incident handling: How to select an appropriate response against a network-based attack? In *IT Security Incident Management & IT Forensics (IMF), 2015 Ninth International Conference on*, pages 51–67. IEEE, 2015.

[18] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[19] J. M. Porta, N. Vlassis, M. T. Spaan, and P. Poupart. Point-based value iteration for continuous pomdps. *The Journal of Machine Learning Research*, 7:2329–2367, 2006.

[20] M. L. Puterman and M. C. Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

[21] A. Shameli-Sendi and M. Dagenais. Orcef: Online response cost evaluation framework for intrusion response system. *Journal of Network and Computer Applications*, 2015.

[22] Y. S. Shao and D. Brooks. Energy characterization and instruction-level energy model of intel's xeon phi processor. In *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, pages 389–394. IEEE Press, 2013.

[23] N. Stakhanova, S. Basu, and J. Wong. A cost-sensitive model for preemptive intrusion response systems. In *AINA*, volume 7, pages 428–435, 2007.

[24] T. Toth and C. Kruegel. Evaluating the impact of automated intrusion response mechanisms. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 301–310. IEEE, 2002.

[25] K. M. Vieira, D. S. M. Pascal Filho, C. B. Westphall, J. B. M. Sobral, and J. Werner. Providing response to security incidents in the cloud computing with autonomic systems and big data.

[26] K. M. Vieira, F. Schubert, G. A. Geronimo, R. de Souza Mendes, and C. B. Westphall. Autonomic intrusion detection system in cloud computing with big data. In *Proceedings of the International Conference on Security and Management (SAM)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2014.

[27] X. Zan, F. Gao, J. Han, X. Liu, and J. Zhou. A hierarchical and factored pomdp based automated intrusion response framework. In *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, volume 2, pages V2–410. IEEE, 2010.