Statistic complexity metrics as a basis for formal probabilistic model checking

Irakli RODONAIA*
Vakhtang RODONAIA **

Abstract

The paper proposes a new technique for detecting malware threats in autonomic component ensembles. The technique is based on the statistic complexity metrics, which relates objects to random variables and (unlike other complexity measures considering objects as individual symbol strings) are ensemble based. This transforms the classic problem of assessing the complexity of an object into the realm of statistics. The proposed technique requires the implementation of the process X (which generates 'healthy' flows containing no malware threats) and objects generated by the actual (possible infected) process Y. The component flows files are used as objects of the processes X and Y. The result of the proposed procedure gives us the distribution of probabilities of malware infection among autonomic components. Quantitative verification techniques are implemented within PRISM, a probabilistic model checker, which provides direct support for DTMCs, MDPs and CTMCs. The distribution of malware threats probabilities, obtained by the above-described procedure serves as a starting point for formal reasoning about the behavior of the Autonomic components ensembles.

Keywords: autonomic ensemble, complexity measure, statistic complexity, traffic flows, malware, Markov chains, quantitative verification.

Introduction

The problem of anomaly detection in autonomic component ensembles was considered by Prangishvili et al. (2013, 2014), where the following problem was set. A singleton application currently runs on one of the VMs at an Datacenter. During the session the application experiences consistently high CPU load. This increase may be caused either by legitimate traffic overload or by coordinated attacks (DDOS) launched against the PaaS provider. The latter might be wrongly assumed to be legitimate requests and resources would be scaled up to handle them. This would result in an increase in the cost of running the application (because a provider will be charged by these extra resources) as well as in violation of SLA (due to increased response times). Hence, it is necessary to distinguish between these two cases, the earlier this distinction is made, the higher is the degree of protection of the application from failure and poor performance. To provide this protection, the following security measures were suggested. The traffic flows through the VMi had to be analyzed using Kolmogorov complexity metrics. During the session the constant monitoring of the metric (by the special probe implemented in the separate module), along with measure of CPU load and available memory size was being executed. If the traffic satisfied some pre-formulated criteria (indicated that there exist serious DDOS attack threats) then the application rapidly migrated to some other VMj.

The technique described by (Prangishvili et al., 2013, 2014) implemented Kolmogorov complexity metrics to reveal the possible malware attacks and had to deal with only with DDOS attacks. Despite its usefulness, Kolmogorov complexity does not capture the intuitive notion of the complexity very well. For example, random strings without any regularities , say that , strings that are constructed bitwise by repeated tosses of a fair coin, have very large Kolmogorov complexity. However, those strings are not "complex" from an intuitive point of view — those strings are completely random and do not carry any interesting structure at all. Many approaches have been suggested to define some complexity measures that are closer to the intuitive notion of complexity and overcome the difficulties of Kolmogorov complexity. For example, Kolmogorov complexity is based on algorithmic information theory considering objects as individual symbol strings, whereas the measures *effective measure complexity* (EMC) (Grassberger, 1986), *excess entropy* (Crutchfield et al., 1983), *predictive information* (Bialek et al., 2001)), etc., relate objects to random variables and are ensemble (that is, set of interrelated objects –symbol strings) based.

The Kolmogorov complexity measures M assigns a complexity value to each individual object x' under consideration. Let us denote it as $C_M(x')$. It is assumed that x' corresponds to a string sequence of a certain length and its components assume values from a certain domain. In (Feldman et.al.,1997), (Ziv Bar-Yossef et al, 2003), (Em-

* Prof.Dr.Department of Technologies and Engineering, International Black Sea University, Tbilisi, Georgia
E-mail: irakli.rodonaia@ibsu.edu.ge

** Assist.Prof., Department of Technologies and Engineering, International Black Sea University, Tbilisi, Georgia
E-mail:vakhtang.rodonaia@ibsu.edu.ge

Journal of Technical Science and Technologies; ISSN 2298-0032; Volume 4, Issue 1, 2015

mert-Streib, 2010) *statistic complexity* that is not only different to all other complexity measures introduced so far, but also connects directly to statistics, specifically, to statistical inference, was introduced. More precisely, a complexity measure with the following properties is introduced. First, the measure is bivariate comparing two objects, corresponding to the pattern generating processes, on the basis of the *normalized compression distance*(NCD) (Cilibrasi et al,. 2005), (Terwijn et al, 2011) with each other:

$$NCD(x,y)=\frac{C(xy)-\min\{C(x),C(y)\}}{\max\{C(x),C(y)\}}$$

where C (x) denotes the compression size of string x and C (xy) the compression size of the concatenated stings x and y.

Second, this measure provides the quantification of an error that could have encountered by comparing samples of finite size from the underlying processes. Hence, the statistic complexity provides a statistical quantification of the statement 'X that is similarly complex as Y '. This implies that a fundamental complexity measure needs to be bivariate, C(X, Y), instead of univariate comparing the two processes X and Y.

Next, the desirable property of any complexity measure is that a complexity measure should quantify the uncertainty of the complexity value. As motivation for this property we just want to mention that there is a crucial difference between an observed object x' and its generating process X. If the complexity of X should be assessed, based on the observation x' only, this assessment may be erroneous. This error may stem from the limited (finite) size of observations. Also, the possibility of measurement errors would be another source of wrong assessment. Based on these considerations, the statistic complexity measure, suggested by (Emmert-Streib (2010) is defined by the following procedure:

1. Estimate the empirical distribution function

$\hat{F}_{XX}$ of the normalized compression distance from n_1

$$S_{X,X}^{n_1}=\{x_i=NCD(x',x'')\,|\,x',x''\prec X\}_{i=1}^{n_1}$$

from objects x' and x'' of size m generated by process X (here ' $\prec$ ' means 'is generated by X')

2. Estimate the empirical distribution function $\hat{F}_{XY}$ of the normalized compression distance from n_2 ,

$$S_{X,Y}^{n_2}=\{y_i=NCD(x',y')\,|\,x'\prec X,y'\prec Y\}_{i=1}^{n_2}$$

from objects x' and y' of size m generated by two different processes X and Y.

3. Determine

$$T=\sup_x\left|\hat{F}_{X,X}(x)-\hat{F}_{X,Y}(x)\right|$$

and $p=Prob(T\le t)$.

4. Define

$$C_s(S_{X,X}^{n_1},S_{X,Y}^{n_2}\,|\,X,Y,m,n_1,n_2):=p$$

as *statistic complexity.*

This procedure corresponds to a two-sided, two-sample Kolmogorov-Smirnov (KS) test based on the normalized compression distance (Cilibrasi et al., 2005) obtaining the distances among observed objects.

The statistic complexity corresponds to the p-value of the underlying null hypotheses, $H_0:F_{XX}=F_{XY}$, and, hence, assumes values in [0,1]. The null hypothesis is a statement about the null distribution of the test statistic

$$T=\sup_x\left|\hat{F}_{X,X}(x)-\hat{F}_{X,Y}(x)\right|$$

and because the distribution functions are based on the normalized compression distances among objects x' and x'', drawn from the processes X and Y. This leads to a statement about the distribution of normalized compression distances. Hence, verbally, H0 can be phrased as "on average, the compression distance of objects from X to objects from Y equals the compression distance of the objects only taken from X". If the alternative hypothesis $H_1:F_{XX}\neq F_{XY}$ is true, this equality does no longer hold implying differences in the underlying processes X and Y, leading to the differences in the NCDs.

Applied to the problem of finding malware threats in the flows between autonomic components CPi (Prangishvili et al., 2013, Prangishvili et al., 2014) the above procedure will look as follows. For each autonomic component (AC) of the autonomic-component ensembles (ACEs) the processes X and Y are considered as the processes generating objects represented in the form of strings. The strings, in turn, represent traffic flows through these autonomic components. The specific ways of how flows are transformed into strings are considered later in the paper. The process X ('training process') is the process generating flows in the conditions when there are no malware threats. So, objects (strings) generated by the process X are 'healthy' (they do not contain any patterns of malware). These strings have to be generated preliminary (before actual workload on an autonomic components ensemble). Some fractions of objects (string) have to be generated for the situation with unusual (but not malicious) behavior. For randomly taken pairs x' and x'' (the amount of such pairs is n_1) of the generated strings the metric NCD(x' , x'') is calculated. The size of samples n_1 has to be sufficient to account for various possible situations and conditions that may occur in the specific autonomic ensemble under consideration. Then the empirical distribution function $\hat{F}_{XX}$ is being built and stored in the specific place.

When the ensemble starts actual operation (receives workload), the process Y ('production process') generates objects (strings) y', which represent actual current traffic between ensemble's components. Some of these objects may contain malware patterns. The sample of the size n_2 of objects x' (generated preliminary by the 'training process' X) and objects is being created and the metric NCD (x' , y') is calculated for each pair. Then the empirical distribution function $\hat{F}_{XY}$ is being built. Now, by applying the steps 3 and 4 of the above procedure, the values of the *statistic complexity for each autonomic component can be computed.*

The obtained numerical value of the statistic complexity can be interpreted in the following sense: in the current

Journal of Technical Science and Technologies; ISSN 2298-0032; Volume 4, Issue 1, 2015

conditions the flows of packets through the given autonomic component cannot be regarded as complex flows (with the probability equal to p). That is, the flows may contain some patterns (indicating the possible presence of some malware threats) with the probability p.

It should be pointed out that in production conditions (when the ensemble is under actual workload) the sample size n2 cannot be determined in advance. This size depends on the actual working conditions: traffic intensity, frequency of creation of objects (strings), actual hardware indices (CPU load, available memory, etc.). As a rule, the number n2 is less than the number n1. This fact can somewhat decrease the precision of the metric, but it requires additional technical consideration. In general, the statistic complexity has the very desirable property that the power reaches asymptotically 1 when $n_1 \to \infty$ and $n_2 \to \infty$. This means, for infinite many observations the error of the test to falsely accept the null hypotheses when in fact the alternative is true and becomes zero. Formally, this property can be stated as $p \to 0$ for $n_1 \to \infty$ and $n_2 \to \infty$.

Finally, it should be noted that despite the fact that statistic complexity is a statistical test, it borrows part of its strength from the NCD and, respectively, Kolmogorov complexity on which this is based on. Hence, it unites various properties from very different concepts.

Application of statistic complexity to autonomic components ensembles

In the proposed approach, to anomaly detection in autonomic component ensembles, an attempt to deal with the wide range of malware threats has been made (unlike the techniques described above and in (Prangishvili et al., 2013, Prangishvili et al., 2014), which had to deal only with DDOS attacks).Autonomic cloud computing datacenters can be considered as autonomic-component ensembles (ACEs) and be represented by constructions of SCEL (Software Component Ensemble Language), a kernel language for programming autonomic computing systems (Prangishvili et al., 2013), (ASCENS, 2010), (De Nicola R et al., 2013). Each (virtual) machine is running one instance of the Cloud Platform called Cloud Platform instance (CPi). Each CPi is considered to be a service component. Multiple CPs communicate over the Internet (IP ptotocol), thus forming a cloud and within this cloud one or more service component ensembles. The notions of autonomic components (ACs) and autonomic-component ensembles (ACEs) (ASCENS, 2010), (De Nicola R. et al., 2013) have been put forward as a means to structure a system into well understood, independent and distributed building blocks that interact in specified ways.

The process part of a component (Fig.1) is split into an autonomic manager controlling execution of a managed element. The autonomic manager monitors the state of the component, as well as the execution context, and identifies relevant changes that may affect the achievement of its goals or the fulfillment of its requirements. It also plans adaptations in order to meet the new functional or non-functional requirements, executes them, and monitors that its goals are achieved, possibly without any interruption. A managed element can be seen as an empty “executor” which retrieves from the knowledge repository the process implementing a required functionality id and bounds it to a process variable Z, sends the retrieved process for execution and waits until it terminates. Also actual parameters for the process to be executed can be stored as knowledge items and retrieved by the executor (or by the process itself) when needed.

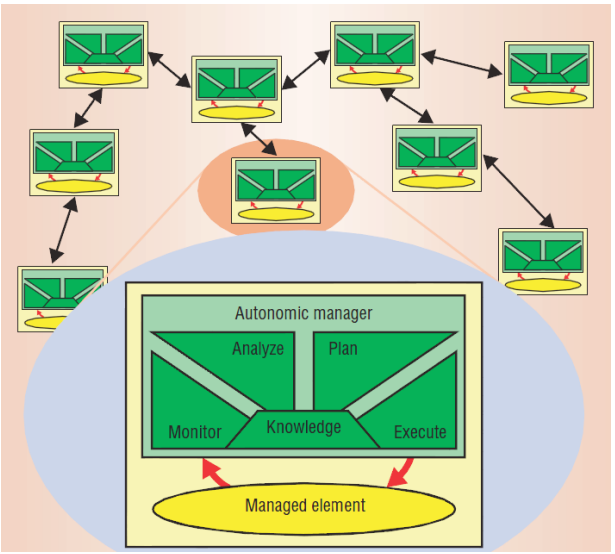

Figure 1. *Functional description of a component*

In our approach the notions of netflows, their informational-theoretical metrics and components’ autonomic manager are essentially leveraged. A network flow can be defined in many ways. In a general sense, a flow is a series of packets with some attribute(s) in common. Each packet that is forwarded within a router or switch is examined for a set of IP packet attributes. These attributes are the IP packet identity or fingerprint of the packet and determine if the packet is unique or similar to the other packets. All packets with the same source/destination IP address, source/destination ports, protocol interface, and class of service are grouped into a flow and then packets and bytes are labeled. This methodology of fingerprinting or determining a flow is scalable because a large amount of network information is condensed into a database of netflow information called the netflow cache.

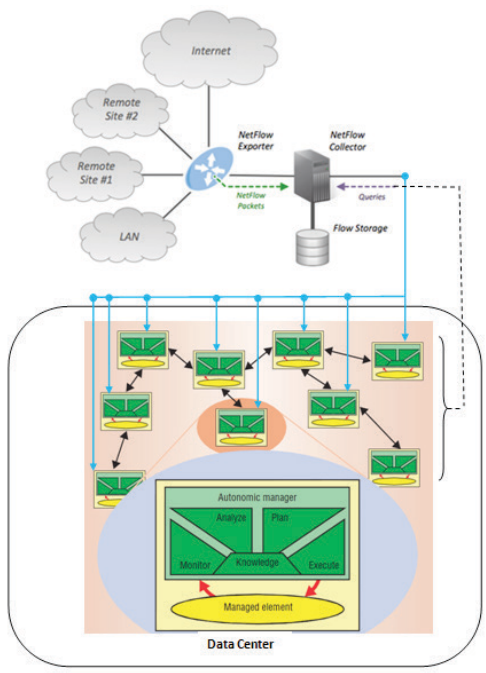

Figure 2. *Interaction between netflow devices and autonomic components*

Journal of Technical Science and Technologies; ISSN 2298-0032; Volume 4, Issue 1, 2015

A netflow-enabled device (netflow exporter: router or switch) (see the Fig.2) sends to the netflow collector single flow as soon as the relative connection expires. This can happen 1) when TCP connection reaches the end of the byte stream (FIN flag or RST flag) are set; 2) when a flow is idle for a specific timeout; 3) if a connection exceeds long live terms (30 minutes by default).

Packets captured by the netflow collector are stored to a flow storage. In our approach the duration of each flow's formation time is unknown in advance and actually is defined by the relevant collectors on the basis of the selected connection expiration time criteria.

Flows accumulated at the flow storage are then subdivided into component flows. That is, flows which have the component's IP address as a destination address are grouped and sent to the corresponding component (more exactly, to the autonomic manager of a component - these flows are marked with blue arrows in the Fig.2).

After receiving their destined flows, the component's autonomic manager can start the processing in order to reveal the abnormal behavior of flows in accordance with the following technique.

Application for collecting and processing NetFlow statistics are defined below (Fig.3).

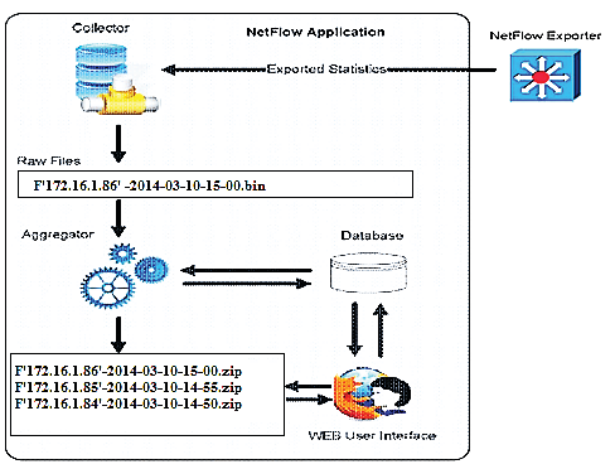

Figure 3. *Components of the NetFlow system for analysis of the statistics*

Once the collector populates the raw file, the file is passed on to the second component in the system, which is called an aggregator. The aggregator receives the file from the collector and processes it using predefined information from the database. The data thus processed (aggregated) is stored in the database. The user interface is a web application that enables us to obtain information on the status of the network, based on the data aggregated in the database. If it is necessary to get more detailed information about a specific communication, the user may open the relevant raw file via the web and filter it according to the desired criteria. The location of the device collecting NetFlow statistics depends on the architecture of the network itself. The amount of NetFlow information exported by network devices is directly dependent on the amount of traffic passing through that device (exporter). Experience has shown that the amount of NetFlow traffic does not exceed 1% of the total amount of traffic through the network, so the "distance" between the server (collector) and the network device exporting the data (exporter) is not relevant. The accessibility and the security of the server are the more important parameters.

In the proposed approach the different files with the particular titles (relevant to the concrete SCPi's IP addresses) to store component flows are used. For example, for the component flow to the SCPi with IP address 172.16.1.86, occurred on 2014/03/16 at 15:00, the files with titles F'171.16.1.86'-2014-03-10-15-00.bin and the F'171.16.1.86'-2014-03-10-15-00.zip will be created. If we look at known threats in data networks from point of unwanted traffic, we can separate the following groups (Ekmanis, 2013):

1. Denial of service attacks.

2. Port scans and remote vulnerability searching and virus spread.

3. P2P files exchange networks.

4. Email spam and web popup.

5. Open resources misuse (open DNS, open mail relay, open proxy, Trojan horse, etc.)

In our approach we observe the following traffic flow attributes (Kołaczek, Juszczyszyn, 2008):

• Source/destination IP address and port number

To measure changes in IP address and port number space we observe a value of Shannon entropy related to these attributes (entropy is used to capture the degree of dispersal or concentration of the distributions for traffic attributes). Entropy values are calculated for separate component flows files (obtained by using the utility nfdump). Different AMs (Autonomic Manager) use various time periods length (see connection expiration time criteria above). The following network variables are used for each component flows files: entropy of source IP address, entropy of destination IP address, entropy of destination port number, entropy of source port number. Duration attributes of each component flow time are different and depend on the traffic conditions and selected connection expiration time criteria

• Number of bytes and packets

These values are: bytes received by a host, bytes sent by a host, packets received by a host, packets sent by a host. Again, duration attributes of each component flow files are different.

• TCP flags

The attribute TCP_FLAG - a difference between number of SYN packets sent and RST and FIN packets received - is measured in the proposed approach. In normal conditions, in long time observation we should get the mean value of TCP_FLAG near zero. Intrusive actions like system scanning, DoS attacks may cause the temporal distortion of the mean value of TCP_FLAG.

• Duration of the connection

During various types of attacks, this value will be affected and so an anomaly may be detected. For example, worm infection will generate a large number of connections with similar duration. We simply use the value of connections' duration attribute contained in the given component flow file.

• Communication Patterns

Fan-in is the number of nodes that originate data exchange with the current CP_i, while Fan-out is the number of

Journal of Technical Science and Technologies; ISSN 2298-0032; Volume 4, Issue 1, 2015

hosts to which CP_i initiates conversations. The above patterns are invariant during most time of normal system activity or change in a predictive way. But while attack appears they will change significantly.

As one can see, the component flows files contain the same volume of information (they contain the same amount of attributes of the same size). Hence, we can assume that the size m of a component flow file represents the object (in terms of the statistic complexity procedure) of size m. In general, component flows files are regarded as objects $x', x'', x''', \ldots\ldots$ generated by the process X ('training process') and $y', y'', y''' \ldots\ldots\ldots\ldots$, objects generated by the process Y ('production process').

As it was described, the proposed procedure requires implementation of the 'training process' X (which generates 'healthy' flows containing no malware threats) before starting real 'production' (real-time) process Y. In order to decrease overheads, this process is executed just once with as large value of the sample size n1 as it is possible. The obtained results (the empirical distribution function $\hat{F}_{XX}$) is stored to each CPi which can run applications subsequently. When applications are executed on the CPs, the objects $y', y'', y''' \ldots\ldots\ldots\ldots$, (corresponding component flows files) are created and the empirical distribution functions $\hat{F}_{XY}$ are calculated on each CP_i . Then, according to the steps 3 and 4 of the procedure, the value of statistic complexity for each autonomic component is calculated.

The result of the proposed procedure gives us the distribution of probabilities of malware infection among autonomic components of the datacenter.

To estimate the statistic complexity's value, which practically indicates real malware threat, numerous simulation experiments were carried out. The well-known simulation tool CloudSim - a framework for modeling and simulation of cloud computing infrastructures and services – has been used. As a result of simulation experiments, we determined that the statistic complexity's value less than 0.25 can be practically regarded as serious malware threat. In this condition the immediate migration of the application from the VM (where the application is being run currently) to another VM (which is to be selected by using the ensemble's components autonomic managers' knowledge base and issuing the special SCEL statement qry) is required.

It should be pointed out that detection of malware threats and consequent migration are being executed in real-time scale and thus minimize damage from possible malware threats. This also contributes to maintaining the required SLA.

The time of migration must be taken into account when determining the response time. In general, streams of requests generated by each client (application) may be decomposed into a number of different VMs. In case of more than one VM serving the i^{th} client, requests are assigned probabilistically, i.e., α_{ij} portion of the incoming requests are forwarded to the j^{th} server (host of a VM) for execution.

The exponential distribution function is used to model the service time of the clients in this system. Based on this model, the response time distribution of a VM (placed on server j) is an exponential distribution with mean:

$$\bar{R}_{ij} = \frac{1}{C_j^p \phi_{ij} \mu_{ij} - \alpha_{ij} \lambda_i} \qquad (1)$$

where μij denotes the service rate of the i-th client on the j-th server when a unit of processing capacity is allocated to the VM of this client. The VM unit is defined as the basic unit of virtual resource, which is associated with a set of physical resources such as CPU time, main memory, storage space, electricity etc. In real cloud systems, any virtual resource a customer can apply should be a multiple of the VM unit.

Migrating a VM between servers causes a downtime in the client's application. Duration of the downtime is related to the migration technique used in the datacenter. The downtime also is the function of the link speed and VM memory size.

Let us assume that an application *i* had to migrate n_i times during its execution cycle. We introduce the following notations:

n_i - amount of migration of the i-th application during its execution cycle;

m_k - the number (index) of VM (CP) on which the application runs in k-th migration period;

SC_{ip} - the value of the statistic complexity obtained for the i-th application running on the p-th VM in the given time period

$\bar{R}_{ij}$ - see (1)

Then the formula (1) must be updated by adding the term representing the expected downtime of the VMij:

$$\begin{cases} \bar{R}_{im_1} & if \ n_i = 0 \\ \sum_{k=1}^{n_i} (SC_{im_k} * (\bar{R}_{im_k} + DT_{im_k}(LinkSpeed))) & otherwise \end{cases}$$

The obtained estimation of response times is much closer to the actual response times (observed in real operational conditions) and thereby contributes to maintaining the required SLA.

Conclusions and future works

In the paper we presented a new technique for detecting malware threats in autonomic component ensembles. The technique is based on the statistic complexity metrics. Unlike the Kolmogorov complexity, which is based on algorithmic information theory considering objects as individual symbol strings, the statistic complexity relates objects to random variables and are ensemble based. It is a bivariate measure that compares two objects, corresponding to the pattern generating processes, on the basis of the normalized compression distance with each other. Besides, this measure provides the quantification of an error that could have been encountered by comparing samples of finite size from the underlying processes. The approach transforms the classic problem of assessing the complexity of an object into the realm of statistics. This may open a wider applicability of this complexity measure to diverse application areas. In particular, the statistic complexity is applied to the problem of detecting malware threats in autonomic component ensembles. The proposed procedure requires implementation of the 'training process' X (which generates 'healthy'

flows containing no malware threats) and objects generated by the actual (possible infected) process Y ('production process'). The component flows files are used as objects of the processes X and Y. The result of the proposed procedure gives us the distribution of probabilities of malware infection among autonomic components of the datacenter. The proposed procedure of detecting malware threats and consequent migration are being executed in real-time scale and thus minimizes damage from possible malware threats. This also contributes to maintaining the required SLA.

Quantitative verification based on the statistic complexity estimates

Model checking represents a formal technique for verifying whether a system satisfies its specification. The technique involves building a mathematically-based model of the system behavior and checking that system properties specified formally in a temporal logic hold within this model. The result is based on an exhaustive analysis of the state space of the considered model - a characteristic that sets model checking apart from complementary techniques such as *testing and simulation*.

Quantitative verification techniques (Kwiatkowska, 2013), (Calinescu et al., 2008) are implemented within PRISM, a probabilistic model checker, which provides direct support for discrete-time Markov chains(DTMCs), Markov decision processes (MDPs) and continuous- time Markov chains (CTMCs).

The *discrete-time Markov chain* (DTMC), is defined by a set of states S and a transition proba-bility matrix P: S × S → [0, 1], where P(s, s') is the probability of making a transition from one state s to another state s'. *Markov decision processes* (MDPs) extend DTMCs by allowing both probabilistic and nondeterministic behavior. More formally, in any state there is a nondeterministic choice between anumber of discrete probability distributions over states. Non-determinism enables the modeling of asynchronous parallel composition of probabilistic systems. It also permits under-specification of certain aspects of a system. A *continuous-time Markov chain* (CTMC), on the other hand, is defined by a set of states S and a transition rate matrix R : S × S → $IR_{\geq 0}$, where R(s, s') is the rate of making a transition from state s to s'. The interpretation is that the probability of moving from state s' within t time units (for positive, real-valued t) is $1-e^{-R(s,s')\cdot t}$

MDP properties are typically expressed in temporal logic PCTL (Probabilistic Computation Tree Logic). Examples of PCTL properties are: "What is the maximum probability over all possible strategies (variants of processes progress) of migration of an application being executed on VMi?" or "What is the minimum probability over all possible strategies, of SLA violation within 10 time steps? " or "What is the long-time probability of the ACE's being operated without SLA violation at least 0.99?" and etc. The distribution of malware threats probabilities obtained by the above-described procedure, serves as a starting point for formal reasoning about the behavior of the ACEs.

Quantitative verification techniques is implemented within PRISM, a probabilistic model checker developed at the Universities of Birmingham and Oxford. PRISM provides the direct support for DTMCs, MDPs and CTMCs. Two key aspects of autonomous cloud computing ensembles are: autonomous behavior and adaptivity. Each aspect can be explored by the use of PRISM. The temporal PCTL to specify autonomous behavior goals can be employed and the above goal may be as follows: "What is the probability that the autonomous ensemble will remain in safe (from the SLA requirements' standpoints) state until finishing all assigned tasks successfully?".The problem can be stated as follows. Given a PCTL formula Φ that specifies the mission goal determines a control strategy that optimizes the probability of satisfying *Φ*. Clearly, this problem can be solved by applying quantitative verification, namely, computing the minimum/ maximum probability or expectation, and then synthesizing the optimal strategy.

Autonomous cloud ensembles dynamically adapt behaviors to the changing requirements and on texts. It has been argued that the need to continuously provide reliability, dependability and performance guarantees for adaptive systems calls for *quantitative runtime verification*. This is different from offline quantitative verification performed at the design stage where a model is developed and analyzed pre-deployment in order to improve the design. Runtime verification, in contrast, is invoked as the system is being executed, intercepting and steering its execution to ensure that given requirements are continuously satisfied in spite of adaptation. The framework proceeds *autonomically,* repeatedly invoking the monitoring, analysis, planning and execution stages as follows:

- *monitor* the reliability, workload and response time of services, to derive an operational model;

- *analyze* performance and QoS requirements, utilizing the values of parameters obtained from the monitoring phase;

- *plan* adaptation of the system based on the results of analysis, which may involve changing the resource allocation or selection of optimal service;

- *execute* the adaptation of the system

The models used are DTMCs and CTMCs, and the following are example requirements:

- $P_{\leq 0.20}$ [F failed alarm]-"the probability that at an alarm (need to migrate to another VM due to the high probability of malware threat) failure ever occurs during the lifetime of the system is less than 0.20" (PCTL property);

- $R_{\leq 0.05}$ [$F^{[service\ \#35\ being\ executed]}$ dropped] -"the probability of a change VM request being dropped due to the request queue being full during a day of operation is less than 0.05" (CSL property).

The QoS framework implements the analysis stage using quantitative verification with PRISM. This involves executing PRISM verification tasks at run-time.

It is necessary to point out again that the DTMC, CTMC and MDP procedures run within PRISM by using the distribution of malware threats probabilities obtained by the statistic complexity procedure which serves as a starting point for formal reasoning about the behavior of the ACEs.

In the future work, we plan to develop a hybrid technique which combines aspects of symbolic and explicit approaches to overcome some performance problems. We are planning to add a MTBDDs (multi-terminal binary decision diagrams) to a purely symbolic framework of PRISM.

Journal of Technical Science and Technologies; ISSN 2298-0032; Volume 4, Issue 1, 2015

References

ASCENS, (2010): http://www.ascens-ist.eu

Bialek W, Nemenman I, Tishby N (2001) Predictability, complexity, and learning. Neural Computation 13: 2409–2463.

David P. Feldman, James P. Crutchfield. (1997). Measures of Statistical Complexity: Why?. Department of Physics, University of California, Davis, CA 95616, Electronic Address: dpf@santafe.edu, 1997

De Nicola R., Loreti M., Pugliese R., Tiezzi F. (2013). SCEL - a Language for Autonomic Computing. ASCENS project, Technical report.

Cilibrasi R., Vitanyi P., (2005). Clustering by compression. IEEE Transactions Information Theory 51: 1523–1545

Ekmanis, M. (2013). Unwanted traffic identification problems. Martins Department of Telecommunications, Riga Technical University, Azenes iela 12, LV-1048, Riga, Latvia.

Emmert-Streib F. (2010). Statistic Complexity: Combining Kolmogorov Complexity with an Ensemble Approach, Queen's University, Belfast, United Kingdom.

Grassberger P. (1986). Toward a quantitative theory of self-generated complexity. Int J Theor Phys 25: 907–938

Kołaczek G., Juszczyszyn K. (2008). Attack pattern analysis framework for multi-agent intrusion detection system. International Journal of Computational Intelligence Systems, Vol.1, No. 3 pp. 215 - 224

Kwiatkowska M.(2013). Advances in Quantitative Verification for Ubiquitous Computing, In Proc. 11th International Colloquium on Theoretical Aspects of Computing (ICTAC 2013), volume 8049 of LNCS, pages 42-58, Springer, Heidelberg.

Prangishvili, A., Shonia, O., Rodonaia, I., Rodonaia, V. (2013). Formal security modeling in autonomic cloud computing environment. WSEAS / NAUN International Conferences, Valencia, Spain.

Prangishvili, A., Shonia, O, Rodonaia, I., Mousa M (2014). Formal verification in autonomic component ensembles, WSEAS / NAUN International Conferences, Salerno, Italy.

S.A. Terwijn, S. A., Torenvliet, L. & Vitanyi, P.M.B. (2011). Nonapproximability of the Normalized Information Distance, J. Comput. System Sciences, 77:4, 738–742

Bar-Yossef, Z., Jayram, T.S. & Sivakumar, D.R. (2003). An information statistics approach to data stream and communication complexity. IBM Almaden Research Center, San Jose, CA 95120.