QuickAdapt: Scalable Adaptation for Big Data Cyber Security Analytics

Faheem Ullah[a, b] and M. Ali Babar [a, b]
[a] CREST – Center for Research on Engineering Software Technologies, The University of Adelaide, Australia
[b] Cyber Security Adelaide, The University of Adelaide, Australia
Email: {faheem.ullah, ali.babar}@adelaide.edu.au

***Abstract*— Big Data Cyber Security Analytics (BDCA) leverages big data technologies for collecting, storing, and analyzing a large volume of security events data to detect cyber-attacks. Accuracy and response time, being the most important quality concerns for BDCA, are impacted by changes in security events data. Whilst it is promising to adapt a BDCA system's architecture to the changes in security events data for optimizing accuracy and response time, it is important to consider large search space of architectural configurations. Searching a large space of configurations for potential adaptation incurs an overwhelming adaptation time, which may cancel the benefits of adaptation. We present an adaptation approach, *QuickAdapt*, to enable quick adaptation of a BDCA system. *QuickAdapt* uses descriptive statistics (e.g., mean and variance) of security events data and fuzzy rules to (re)compose a system with a set of components to ensure optimal accuracy and response time. We have evaluated *QuickAdapt* for a distributed BDCA system using four datasets. Our evaluation shows that on average *QuickAdapt* reduces adaptation time by 105× with a competitive adaptation accuracy of 70% as compared to an existing solution.**

Keywords*— *big data, cyber security, adaptation, accuracy

I. Introduction

Traditional cyber security systems such as Intrusion Detection Systems (IDS) and Malware Detection Systems (MDS) are unable to cope with the increasing volume, velocity, and variety (3Vs) of security events data [1, 2]. That is why an increasing number of enterprises are incorporating big data technologies such as Hadoop and Spark to deal with the 3Vs of security events data. A combination of cyber security systems and big data technologies has led to a new breed of systems called **B**ig **D**ata **C**yber Security **A**nalytics (BDCA) systems. A BDCA system is defined as "*A system that leverages big data technologies for collecting, storing, and analyzing a large volume of security events data to protect organizational networks, computers, and data from unauthorized access, damage, or attack*" [3].

A BDCA reference architecture consists of five phases i.e., data engineering, feature engineering, process engineering, data processing, and data post-processing [3]. Each phase comprises of several components responsible for distinct operations. For example, feature engineering comprises of three components i.e., feature selection, feature generation, and feature transformation. In practice, specific components are selected from several components in a reference architecture to generate a concrete architecture. For instance, the BDCA architecture presented in [4] and [5] consists of two components i.e., signature detection and anomaly detection; another one presented in [6] and [7] consists of three components i.e., duplicates removal, features selection, and anomaly detection. We refer to the combination of components as an *architectural configuration*.

The choice of architectural configuration significantly impacts accuracy and response time, which are the two most important quality attributes of a BDCA system [3]. *Accuracy* measures how accurately a BDCA system detects cyber-attacks and *response time* measures how long a system takes to detect an attack [2, 3]. For example, a configuration with and without feature selection shows different accuracy and response time. The *best* configuration is the one with optimal accuracy and response time. Whilst it is promising to select the *best* configuration during system deployment, such an approach fails to cope with intrinsic changes in an operating environment of a system [8]. This is because the *best* configuration varies with changes in the operating environment. Among such changes, the change in security events data is the most prominent one [1, 2]. Examples of changes include change in redundancy, number of features, and class bias in data. Motivated by the widespread adoption of architecture-driven adaptation, we previously presented an architecture-driven adaptation approach, *ADABTics* [8], that selects architectural configuration at runtime to enable a BDCA system to ensure optimal accuracy and response time. One of the key limitations of *ADABTics* is that it takes around 751.5 sec on average to adapt a system. The adaptation time is high for two reasons: (a) the search space, from which *ADABTics* selects the optimal configuration, is large (i.e., 32 configurations considered in [8]) due to several available components and their subsequent combination and (b) *ADABTics* sequentially executes each configuration within the search space to find its effectiveness and ultimately selecting the most optimal configuration (§IV.B). This situation has stimulated a question to be addressed: "***How to enable a BDCA system to quickly adapt to the changes in the security events data?***"

We present *QuickAdapt*, a runtime adaptation approach that optimizes accuracy and response time through a quick adaptation of a BDCA system with respect to changes in security events. Similar to *ADABTics* [8], *QuickAdapt* exploits the component-based BDCA architecture to add or drop components at runtime (§II.A). Whilst the previous adaptation approaches (e.g., [8-10]) face the issue of high adaptation time due to searching a large configuration space, *QuickAdapt* takes a direct approach of monitoring and extracting the necessary insights from security events to realize adaptation. Our approach uses Kolmogorov-Smirnov test [11] to monitor security events data and trigger adaptation, which is based on a set of fuzzy rules developed using the Wang-Mendel's fuzzy rule generation method [12]. The rules express and map the statistics with linguistic rules and membership functions, hence, decide whether a component should be operationalized keeping in view the type of input data. Thus, the rules draw a relationship between the input data and a system's architecture. The components selected based on the rules eventually form the configuration to be adopted. We have evaluated *QuickAdapt* using a Hadoop-based BDCA system deployed on a cluster. Our evaluation shows that in comparison to *ADABTics* [8], *QuickAdapt* reduces adaptation time by 105 times with minimal impact on adaptation accuracy.

II. QuickAdapt Overview

We present *QuickAdapt*, which consists of two parts – BDCA component-based architecture and adaptation approach.

978-1-7281-4646-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICECCS.2019.00016

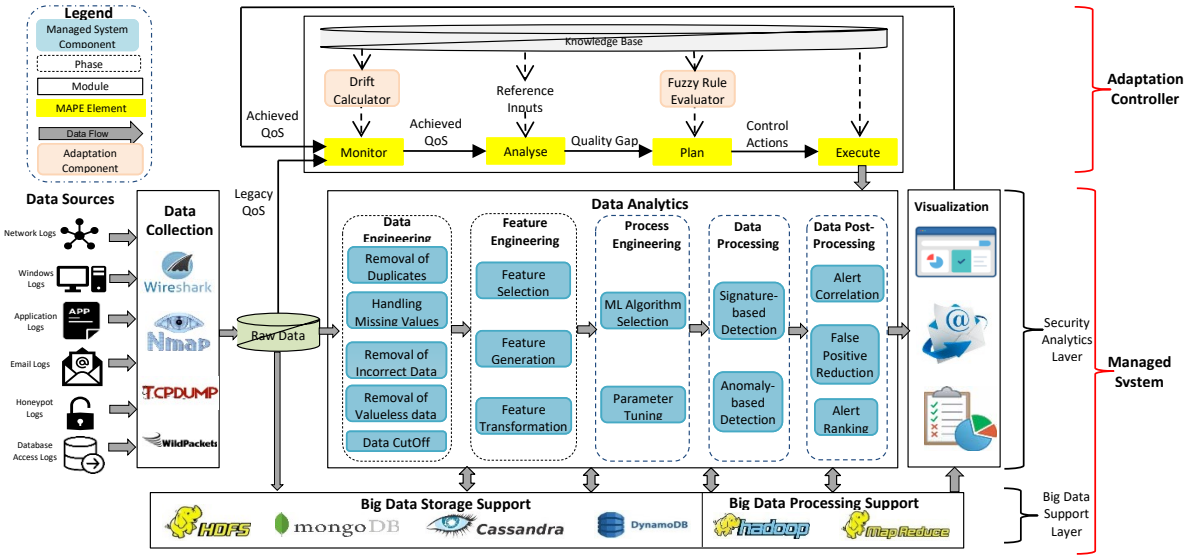

Fig. 1. BDCA Component-based Architecture

A. BDCA Component-based Architecture

A component-based architecture decomposes a system's design into logical components. The components specify units of encapsulated behavior that are used to organize operations and data. We present the component-based architecture of a BDCA system in Fig. 1. The details of the architecture are described in [8]. Here, we briefly outline the various parts of the architecture.

1) Security Analytics Layer

This layer is responsible for collecting, preparing, and analyzing security data. The layer is divided into three modules. The *data collection* module uses various tools (e.g., Wireshark and nmap) to collect security data such as firewall logs, NetFlow data, and email logs. The *data analysis* module analyzes the data to detect cyber-attacks. The module has five phases. The *data engineering* phase uses following components to clean the data. The *Removal of Duplicates* discards redundant records from the data [6]. The *Handling Missing Values* component either removes or implants new values for missing values of a record. The *Removal of Incorrect Data* removes records with incorrect values for some features (e.g., number of source bytes = -10). The *Removal of Valueless data* removes data (e.g., zero-byte connections used in TCP handshaking) that is irrelevant from security point of view. The *Data CutOff* component applies a customizable limit (e.g., first 15 KB of a network connection) on collecting data to reduce data size. The *feature engineering* phase selects, transforms, and/or generates features from the data. This phase consists of three components. The *feature selection* leverages different techniques (e.g., Info Gain) to select features from data that are most relevant for security analytics. The *feature generation* component generates new features from existing features. The *feature transformation* component transforms the features into a range (e.g., [0,1]) to help ML model to clearly discriminate between benign and malicious activities. The *process engineering* phase is responsible for ensuring effective data processing in the subsequent phase. The phase leverages *ML Algorithm Selection* component for selecting the most suitable ML algorithm for anomaly-based detection and *Parameter Tuning* component to tune various parameters such as the value of K for K-means. The *data processing* phase uses anomaly-based detection and signature-based detection to process data for detecting attacks. The *anomaly-based detection* employs an ML algorithm to detect activities that deviate from normal behavior learnt by a model. On the other hand, *signature-based detection* compares the incoming security data with the predefined attack signatures and generates an alert when a match is found. The *data post-processing* phase process the alerts to remove false alarms and prioritize response to more dangerous alerts. The phase consists of three components. The *False Positive Reduction* uses various techniques to reduce the number of false alerts. The *Alert Correlation* component correlates the alerts to reveal complex attack scenarios. Finally, the *Alert Ranking* component ranks alerts based on their severity and dangerousness. The *visualization* module communicates the results (i.e., alerts) with users.

2) Big Data Support Layer

The layer supports the security analytics layer by managing the distributed storage and processing of data. The layer consists of two modules. The *Big Data Storage* module uses storage solutions (e.g., Cassandra and HDFS) to manage the data being processed. The *Big Data Processing Support* module uses the Hadoop framework and MapReduce for data processing.

B. Adaptation Approach

In this section, we present our adaptation approach that exploits the component-based architecture of the BDCA system.

Adaptation Goal: Our approach aims to "*enable a BDCA system to quickly adapt to the changes in the input data*". We use Quality-of-Service (QoS) [13] as a measure to quantify the accumulated impact on accuracy and prediction time. We measure accuracy through four well-known metrics i.e., F1 score (F1), False Positive Rate (FPR), Detection Rate (DR), and Accuracy (Acc) and Prediction Time (PrT) as the time taken by the system to detect attacks in the testing dataset [8].

Reference Inputs and Measured Outputs: Reference inputs specify the target that our adaptation approach aims to achieve. Similar to most of the research on QoS-aware self-adaptation (e.g., [9, 10]), our reference input is the *target QoS* that our approach attempts to achieve. Similar to [8], we measure QoS using the utility function (Eq. 1), which weighs and sums the values for the considered quality metrics (e.g., accuracy and prediction time). In Eq. 1, *QoS(k)* denotes the QoS for *kth* configuration and $\omega \in [0,1]$ specify the weight set by a user to specify the importance of a quality metric. We use Eq. 2 and Eq. 3 to scale the value functions (e.g., *DR (k)*). Some quality attributes (e.g., FPR) are undesirable (i.e., the higher the value the lower the quality) and some quality attributes (e.g., DR) are desirable (i.e., the higher the value, the higher the quality). Therefore, we use Eq. 2 to scale undesirable quality attributes

and use Eq. 3 to scale desirable quality attributes. The measured outputs are the values monitored as system outputs and then compared with the reference inputs to evaluate whether the system has achieved the target. In our approach, the measured output is the *achieved QoS* that is the QoS system delivers after it has adapted to the change.

$$QoS(k) = \frac{\omega_1 * FPR\ (k) + \omega_2 * PrT\ (k)}{\omega_3 * Acc\ (k) + \omega_4 * DR\ (k) + \omega_5 * F1\ (k)} \quad (1)$$

$$V_N(k) = \begin{cases} \frac{V_{max} - V_k}{V_{max} - V_{min}}, if\ V_{max} - V_{min} \neq 0 \\ 1, \quad if\ V_{max} - V_{min} = 0 \end{cases} \quad (2)$$

$$V_P(k) = \begin{cases} \frac{V_k - V_{min}}{V_{max} - V_{min}}, if\ V_{max} - V_{min} \neq 0 \\ 1, \quad if\ V_{max} - V_{min} = 0 \end{cases} \quad (3)$$

Adaptation Trigger: The trigger specifies the condition upon which the adaptation process is started. We monitor the input data as the changes in input data significantly impact the accuracy and prediction time [8]. For quantifying the change in input data, we use Kolmogorov-Smirnov (KS) test [11] that determines the divergence between two data distributions (i.e., D_{old} and D_{new}) shown in line 1 of Algorithm 1 based on the mean values for each feature. After calculating the divergence, we apply *p-value* test with a significance level 0.05 to determine whether the divergence is statistically significant. In case of a significant divergence, adaptation is triggered.

Algorithm. 1. Adaptation Algorithm

Input: C = {C_1, C_2, C_3,…, C_N} // *Set of components*
D_{old} = Old data distribution
D_{new} = New data distribution
R = {R_1, R_2, R_3,…, R_N} // *Set of fuzzy rules*
Output: AC_{opt} = Optimal Architectural Configuration
Steps:
1. Calculate divergence D (D_{old}, D_{new})
2. Calculate *p-value* for divergence D
3. **If** *p-value* <0.05 **then**
4. $AC_{opt} \leftarrow \emptyset$ // *Initialize to empty set*
5. Calculate V_1 to V_{13} for D_{new} //*Input Variable V*
6. Normalize V_1 to V_{13}
7. **for** i = 0 to i = N **do**
8. Execute R_i
9. **If** R_i == SELECT **then**
10. AC_{opt}.Append (C_i) // *Add component*
11. **end**
12. **end**
13. **end**

Control Actions: Upon triggering an adaptation, the control actions are initialized to enable a system to achieve the target state. Given our component-based architecture, *QuickAdapt* relies on architecture reconfiguration for adapting a system. The architecture configuration is specified by the components operationalized (Table III). Eq. 4 is used to calculate the total number of possible configurations in the search space. Eq. 4 uses mathematical combinations to calculate the number of configurations, where C_N denotes the total number of configurations, *a* specifies the number of components before and *b* specifies the number of components after the Anomaly-based Detection component. For the set of components shown in Fig. 1, a=11 and b=3, which generates a search space of C_N = 16376.

$$C_N = \sum_{g=1}^{a} \binom{a}{g} + \sum_{g=1}^{a} \binom{a}{g} \times \sum_{h=1}^{b} \binom{b}{h} \quad (4)$$

***Component Selection*:** The selection of components for creating an effective architectural configuration is a challenging task. This is because the effectiveness of a component depends upon several factors such as the size of input data, the number of classes in input data, the number of features in each instance, and the number of duplicate instances [2, 3]. Given several factors that are considered in component selection, it is challenging to make a binary decision about component selection. Hence, we cannot be 100% sure about the selection of a component. Instead of binary logic, we therefore leverage fuzzy logic in our decision-making process about component selection. Fuzzy logic is a powerful tool for decision making that involves some degree of subjectivity and impreciseness [10]. For example, a component should be selected with a 75% confidence. Fuzzy logic maps *input variables* to *output variables* through a *membership function*. *Input Variables:* We consider 13 *input variables* that are extracted through descriptive analytics [14] of the input data (line 5 of Algorithm 1). The descriptive analytics analyses data to extract information such as mean, variance, and number of features etc. The input variables are described in Table I. The input variables are evaluated as LOW, MEDIUM-LOW (MED-LOW), MEDIUM, MEDIUM-HIGH (MED-HIGH), and HIGH as shown in Fig. 2b. The values for all input variables are scaled in the range [0,1]. *Output Variable:* The *output variable* is the decision, i.e., whether a component should be selected. The decision is denoted as SELECT or NOT SELECT. The membership function is applied on *input variables* to determine the degree of membership for *output variable*. We used triangular membership function for all the rules with 50% membership for each subset e.g., LOW and MED (Fig 2b). *Fuzzy Rules:* Finally, we formulate fuzzy rules presented in Table II for each component that describe the relationship between observation (input variable) and conclusion (output variable). The symbols '∧' and '∨' denote AND and OR operator respectively. We used the well-known Wang-Mendel's method [12] for generation of the fuzzy rules. The main advantage of Wang-Mendel's method is that it is fast and combines the importance of qualitative information and numerical (quantitative) information for generating the rules. For qualitative information, we used the insight gathered through our literature review [3]. For numerical information, we leveraged the information generated through our large-scale experimentation on evaluating the impact of the BDCA components reported in [15]. We briefly outline the steps followed to generate the rules. *Step 1:* divide the input and output spaces of the numerical data into fuzzy subsets *Step 2:* generate fuzzy rules from the data *Step 3:* Assign a degree of importance to each rule to resolve conflicts among the rules *Step 4:* refine the rules based on qualitative insight gathered from the literature review [3] *Step 5:* map the input variables to the output variables using the refined rules. *Fuzzy Rule Execution:* We illustrate the execution of fuzzy rules (line 7-12 of Algorithm 1) by *QuickAdapt* with the help of a decision about the selection of a component i.e., signature- based detection for CIDDS dataset. The fuzzy rule for the selection of signature-based detection consists of two input variables i.e., *test sample size* and *number of features*. The scaled values of *test sample size* and *number of features* for CIDDS is 0.37 and 0.45 respectively deducted through descriptive statistics and shown in Fig. 2b. The rule

Table I. Input Variables and their Descriptions

Input Variable	Description
% of duplicates instances	Percentage of duplicates instances i.e., instances having exactly same values for all features
% of missing values	Percentage of instances with values missing for at least one feature
% of incorrect values	Percentage of instances with incorrect value (e.g. number of source bytes = -20) for at least one feature
% of valueless data	Percentage of instances having no relevance for security analytics e.g., instances associated with zero-bytes
Test sample size	Number of instances in the testing dataset
Train sample size	Number of instances in the training dataset
% of identifier and timing information	% of instances having unique identifier and time window e.g., *ID#1 22:30 instance_1, ID#2 22:31*
Number of features	Number of features in each instance of the data
Feature variance	Variance among the features of the data
Feature correlation	Correlation among the features of the data
% of categorical features	Percentage of categorical features in the data
Source bytes variance	Variance among the values of the source byte feature
Destination bytes variance	Variance among the values of the destination bytes features in the data

Table II. Fuzzy Rules for the Selection of Components

Component	Fuzzy Rule
Removal of Duplicates	**If** (% *of duplicate instances* is MED ∨ % *of duplicate instances* is MED-HIGH ∨ % *of duplicate instances* is HIGH) ∧ (test *sample size* is MED ∨ *test sample size* is MED-HIGH ∨ *test sample size* is HIGH) → **SELECT**
Handling Missing Values	**If** (% *of instances with missing values* is MED ∨ % *of instances with missing values* is MED-HIGH ∨ % *of instances with missing values* is HIGH) ∧ (*test sample size* is MED ∨ *test sample size* is MED-HIGH ∨ *test sample size* is HIGH) → **SELECT**
Removal of Incorrect Data	**If** (% *of instances with incorrect values* is MED ∨ % *of instances with incorrect values* is MED-HIGH ∨ % *of instances with incorrect values* is HIGH) ∧ (*test sample size* is MED ∨ *test sample size* is MED-HIGH ∨ *test sample size* is HIGH) → **SELECT**
Removal of Valueless Data	**If** (% *of instances with valueless data* is LOW-MED ∨ % *of instances with valueless data* is MED ∨ % *of instances with valueless data* is HIGH) ∧ (*test sample size* is MED ∨ *test sample size* is MED-HIGH ∨ *test sample size* is HIGH) → **SELECT**
Data CutOff	**If** (*sample size* is HIGH) ∧ (quantity of *identifier and timing information* is MED ∨ quantity of *identifier and timing information* is MED-HIGH ∨ quantity of *identifier and timing information* is HIGH) → **SELECT**
Feature Selection	**If** (# *of features* is MEDIUM ∨ # *of features* is MED-HIGH ∨ # *of features* is HIGH) ∧ (*correlation among features* is MEDIUM ∨ *correlation among features* is MED-HIGH ∨ *correlation among features* is HIGH) → **SELECT**
Feature Generation	**If** (# *of features* is MED ∨ # *of features* is MED-HIGH ∨ # *of features* is HIGH) ∧ (% of categorical features is LOW ∨ % of categorical features is MED-LOW) → **SELECT**
Feature Transformation	**If** (*variance among features* is MED-HIGH ∨ *variance among features* is HIGH) ∧ (% of categorical features is LOW ∨ % of categorical features is MED-LOW) → **SELECT**
ML Algorithm Selection	**If** (*train sample size* is LOW ∨ *train sample size* is MED-LOW ∨ *train sample size* is MED) ∧ (# *of features* is LOW ∨ # *of features* is MED-LOW) → **SELECT**
Parameter Tuning	**If** (*variance among features* is MED ∨ *variance among features* is MED-HIGH ∨ *variance among features* is HIGH) ∧ (*train sample size* is LOW ∨ *train sample size* is MED-LOW ∨ *train sample size* is MED) → **SELECT**
Signature-based Detection	**If** (# *of features* is LOW ∨ # *of features* is MED-LOW ∨ # *of features* is MED) ∧ (*test sample size* is LOW ∨ *test sample size* is MED-LOW ∨ *test sample size* is MED) → **SELECT**
False Positive Reduction	**If** (*test sample size* is LOW ∨ *test sample size* is MED-LOW) ∧ *variance among instances* is MED-HIGH ∨ *variance among instances* is HIGH) → **SELECT**
Alert Correlation	**If** (*test sample size* is MED-HIGH ∨ *test sample size* is HIGH) ∧ (# of features is MED-HIGH ∨ # of features is MED-HIGH) → **SELECT**
Alert Ranking	**If** (*variance for source bytes* is MED-HIGH ∨ *variance of source bytes* is HIGH) ∧ (*variance for destination bytes* is MED-HIGH ∨ *variance for destination bytes* is HIGH) → **SELECT**

matrix generated from the rule that leads to the decision of SELECT (S) or NOT SELECT (NS) is shown in Fig. 2a. The degree of membership of *test sample size* is 1 for MED- LOW and 0 for other subsets as shown in Fig 2b. The membership of *number of features* is 0.2 for MED-LOW, 0.8 for MED, and 0 for other subsets. Using membership values and rule matrix, we draw fuzzy inferences among which only two are non-zero shown in Fig. 3. One none zero inference is for SELECT and one for NOT SELECT. We used ROOT-SUM-SQUARE method (Eq. 5) to combine strengths of non-zero inferences, which gives us a value of 0.2 for SELECT and 0.82 for NOT SELECT. Finally, we use Eq. 6 to defuzzify the inferences' strengths with centroid values of 0 and 1 for SELECT and NOT SELECT respectively. Eq 6 gives us output value 0.81, which indicates that the signature detection component should not be selected with a confidence level of 81% for CIDDS.

$$Membership,\ \mu(X) = \sqrt{rule1^2 + rule2^2 + --- + ruleN^2} \quad (5)$$

$$Output = \frac{\sum_{i=1}^{N}(Center_i.Strength_i)}{\sum_{i}^{N} Strength_i} \quad (6)$$

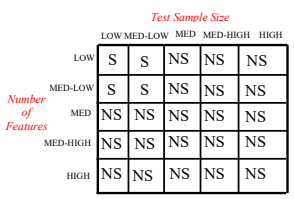

Fig. 2a. Rule Matrix

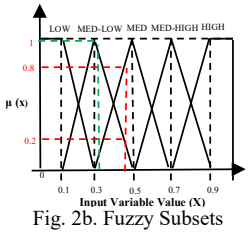

Fig. 2b. Fuzzy Subsets

Test Sample Size — Number of Features — Use Minimum
If MED-LOW and MED-LOW → **SELECT** → min (1, 0.2) = 0.2
If MED-LOW and MED → **NOT SELECT** → min (1, 0.8) = 0.8

Fig. 3. Non-zero Inferences for Signature Detection Component

System Architecture: The system comprises of two parts – managed system and adaptation controller. The managed system has already been discussed in §A. The adaptation controller leverages MAPE-K pattern [13] as shown in Fig. 1. The *monitor* element uses drift calculator to monitor and compute the divergence D in the incoming data. The divergence values are passed on to the *analyzer*, which applies the *p-value* test to determine the significance of divergence. If the divergence is significant, the *analyzer* triggers the *planner* element indicating to the *planner* that the system requires adaptation. The *planner* element uses fuzzy rule evaluator to first compute the values for the 14 *input variables* from the input data. Based on the values for input variables, the *planner* then applies the predefined fuzzy rules to decide which components should be selected for the new configuration. After selecting components, the information is forwarded to the *executor* that realizes the new configuration.

III. Implementation

Instrumentation: We implemented our approach using Apache Hadoop [16]. We installed a Hadoop cluster on an OpenStack cloud. The cluster consisted of one master and 16 slave nodes each running CentOS 6.4 operating system. We allocated 20 GB disk space and 2 GM RAM to each slave node while 80 GB disk space and 8 GB RAM is allocated to the master node. Since the managed system can be implemented using a variety of components as shown in Fig. 1, we only implemented six components for the evaluation of our approach. The components include (C1) Removal of Duplicates (C2) Handling Missing Values (C3) Feature Selection (C4) Signature-based Detection (C5) Anomaly-based Detection and (C6) Alert Ranking. Using Eq. 4 (where a=4 and b=1), these six components generate 32 potential configurations shown in Table III. The implementation details of components are reported in [8] and the source code is available at https://bit.ly/2KQgkTc. We used Apache Mahout [17] for distributed implementation of ML algorithms employed in the Anomaly-based Detection. We evaluated our approach with five state-of-the-art ML algorithms i.e., KMeans (KM), XGBoost (XGB), Naïve Bayes (NB), Support Vector Machine (SVM), and Random Forest (RF). We used XLSTAT for various test (e.g., KS test, Wilcoxon, and p-value test).

Table III. Architectural Configurations

S#	Configuration	S#	Configuration	S#	Configuration	S#	Configuration
1	C5	9	C1-C4-C5	17	C1-C2-C3-C5	25	C2-C4-C5-C6
2	C1-C5	10	C1-C5-C6	18	C1-C2-C4-C5	26	C3-C4-C5-C6
3	C2-C5	11	C2-C3-C5	19	C1-C2-C5-C6	27	C1-C2-C3-C4-C5
4	C3-C5	12	C2-C4-C5	20	C1-C3-C5-C6	28	C1-C2-C4-C5-C6
5	C4-C5	13	C2-C5-C6	21	C1-C3-C4-C5	29	C1-C3-C4-C5-C6
6	C5-C6	14	C3-C4-C5	22	C1-C4-C5-C6	30	C1-C2-C3-C5-C6
7	C1-C2-C5	15	C3-C5-C6	23	C2-C3-C4-C5	31	C2-C3-C4-C5-C6
8	C1-C3-C5	16	C4-C5-C6	24	C2-C3-C5-C6	32	C1-C2-C3-C4-C5-C6

Datasets: We evaluate our approach with four security datasets. The datasets include KDD [18], DARPA [19], CIDDS [20], and CICIDS2017 [21]. The details are presented in Table IV.

Table IV. Statistics of the Security Datasets

Dataset	Record Type	Training Data		Testing Data		Attack Types
		No. of Records	% of Records	No. of Records	% of Records	
KDD	Normal	972781	19.81%	60593	19.41%	Denial of Service, Remote to Local, Probing, and User to Root
	Attack	3925650	80.14%	250436	80.50%	
DARPA	Normal	1145946	42.07%	658797	43.30%	Denial of Service, Remote to Local, Probing, and User to Root
	Attack	1577550	57.92%	863513	56.71%	
CIDDS	Normal	4706999	83.54%	2303898	81.78%	Port Scan, Ping Scan, Brute Force, Denial of Service
	Attack	927349	16.46%	513275	18.22%	
CICIDS2017	Normal	1109933	84.61%	555005	84.60%	Botnet, Web Attack, Heart Bleed, Infiltration attack, and Denial of Service
	Attack	201889	15.40%	100905	15.41%	

IV. Results

A. Configuration, Adaptation Scenarios, and Data Drift

For configuring target QoS, we insert best possible values for all metrics (e.g., 100% accuracy and 0% FPR) to Eq. 1, which gives a value of **0** for target QoS. We assigned a weight of **1** to all metrics in Eq. 1 to specify same level of significance. We evaluate our approach in two scenarios. *Baseline:* The BDCA system is processing a data type (such as KDD) with the *best* configuration for the specific data type. *Change in input data:* The data input into the system is changed (e.g., from KDD to DARPA). With such a change, the monitor element of the adaptation controller signals the drift calculator, which computes the drift to determine whether the drift is significant. In case of a significant drift, adaptation is triggered. We used KS test [11] to calculate the data drift. The six data shifts, associated divergence D, and *p-values* are shown in Table V. The divergence D is same for shift in both directions e.g., KDD→DAPRA and DAPRA→KDD. The *p-value* is less than 0.05 for all cases, which specifies a significant change in data. Hence, adaptation is triggered in all considered cases.

Table V. KS Test Statistics for various Data Drifts

Data Shift	Divergence D	p-value
KDD→DARPA / DARPA→KDD	0.71	0.004
KDD→CIDDS / CIDDS→KDD	0.58	0.003
KDD→CICIDS2017 / CICIDS2017→KDD	0.38	0.001
DARPA→CIDDS / CDDS→DARPA	0.21	0.007
DAPRA→CICIDS2017 /CICIDS2017→DARPA	0.29	0.019
CIDDS→CICIDS2017 / CICIDS2017→CIDDS	0.32	0.014

B. Evaluation Results

1) Adaptation Accuracy: Adaptation accuracy measures the ability of our approach to select components that result in a configuration with a high QoS. We measure adaptation accuracy using top-5 accuracy measure. In our case, the top-5 accuracy means whether the configuration selected by *QuickAdapt* from the 32 configurations (Table III) is included in the configurations with the top 5 achieved QoS. Table VI presents the configurations selected for the four datasets and the achieved QoS for the 20 experimental cases (i.e., 5 ML algorithms × 4 datasets). According to the top-5 measure, the **adaptation accuracy of our approach is 70%**. This means that for 14/20 cases; the selected components and the resulting configuration are included in the top-5 configurations. The top-5 accuracy is 60% (3/5) for KDD, 80% (4/5) for DAPRA, 80% (4/5) for CIDDS, and 60% (3/5) for CICIDS2017. The components in each of the selected configurations are in accordance with the rules presented in Table II. For instance, the higher percentage of duplicate records in KDD and CIDDS (i.e., 78% and 40.5%) results in the selection of Removal of Duplicates component for KDD and CIDDS datasets.

Table VI. Achieved QoS for various datasets and ML algorithms

Dataset	Selected Configuration	KM	XGB	NB	SVM	RF
KDD	C1-C3-C4-C5	1.31	0.86	0.61	0.25	0.71
DARPA	C4-C5	0.99	0.59	0.72	0.78	2.01
CIDDS	C1-C5-C6	0.61	0.55	0.66	0.48	1.1
CICIDS2017	C2-C3-C5-C6	0.36	1.05	0.21	1.33	0.18

2) Adaptation Time: Adaptation time measures the speed with which *QuickAdapt* adapt a system to the change in input data. Adaptation time is counted from the moment adaptation is triggered to the moment the system gains the desired state [13]. The adaptation time of *QuickAdapt* for the four datasets is shown in Fig. 4. On average, ***QuickAdapt* takes around 7.1 sec to adapt the system**. The time is primarily elapsed in the descriptive analysis of a dataset and re-composition of the system i.e., adding and removing components. On average, 55.36% adaptation time in elapsed in descriptive analysis while 44.63% is elapsed in re-composition. The descriptive analysis time is longest (i.e., 9.75s) for CIDIDS2017 followed by CIDDS (3.75S), DARPA (1.5s), and KDD (0.75s). This trend shows that the descriptive analysis time relates to the number of features and number of instances in the dataset. CICIDS2017 having 81 features and 655,910 instances took the longest time while KDD having 42 features and a smaller number of instances (i.e., 311,029) took the shortest time. The time elapsed in re-composition is 4.5sec for CICIDS2017, 4.2sec for KDD, 2.3sec for CIDDS, and 1.7sec for DARPA. This trend is in line with the number of components invoked for each dataset as presented in Table VI. The invocation of four components for CICIDS2017 and KDD took longer as compared to three and two components invocation for CIDDS and DARPA respectively.

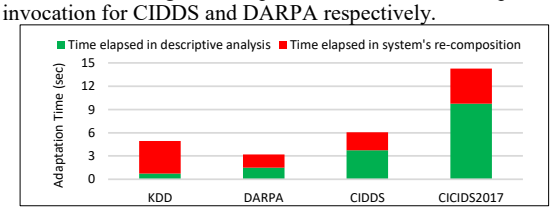

Fig. 4. Adaptation Time of QuickAdapt

3) Optimization: We assess optimization by comparing the accuracy and prediction time exactly exactly before and after adaptation. Table VII reports the accuracy and prediction time for the four datasets (averaged out over the five ML algorithms) before and after adaptation. On average, ***QuickAdapt* improves accuracy and prediction time by 4.52% and 17.6% respectively**. The accuracy improves by the largest margin for KDD and CICIDS2017 (5.1%) followed by DARPA (4.2%), and CIDDS (3.7%). The same trend is observed for prediction time with KDD benefiting the most (i.e., 21.3%) followed by CICIDS2017 (18.5%), DARPA (15.4%), and CIDDS (15.2%). A reason for the significantly larger impact for KDD and CICIDS2017 is the inclusion of the feature selection component. Given the comparatively large number of features in both datasets (i.e., 41 features in KDD and 82 in CICIDS2017), the feature selection component selects the most relevant features, which helps in the improvement of accuracy and prediction time. In order to assess whether the achieved improvement is statistically significant, the values before and after adaptation are subjected to Wilcoxon's signed ranked test at a significance level $\alpha = 0.05$. The *p-values* for accuracy and prediction time are 0.004 and 0.0001 respectively. Since both *p-values* are lower than 0.05, we can conclude that *QuickAdapt* significantly improves both accuracy and prediction time of a BDCA system.

Table VII. Optimization in Accuracy and Prediction Time

Dataset	Accuracy (%)		Optimization (%)	Prediction Time (sec)		Optimization (%)
	Before Adaptation	After Adaptation		Before Adaptation	After Adaptation	
KDD	86.7	91.7	5.1%	18.3	14.5	21.3%
DARPA	80.5	84.8	4.2%	32.1	27.1	15.4%
CIDDS	77.3	81.1	3.7%	23.2	19.8	15.2%
CICIDS2017	87.3	92.4	5.1%	20.1	16.3	18.5%

4) Comparison with ADABTics: Fig. 5 compares the adaptation time of the two approaches for various datasets. On average**, adaptation time for *QuickAdapt* is 105 times lower than *ADABTics***. This is because *ADABTics* executes all configurations before selecting the best configuration while *QuickAdapt* does not rely on executing configurations rather selects a configuration based on fuzzy rules and descriptive analysis of the data. Since *ADABTics* calculates and subsequently compares the QoS delivered by each configuration, it always selects the configuration with the best

QoS. Hence, adaptation accuracy for *ADABTics* is 100% as compared to the 70% adaptation accuracy for *QuickAdapt*. Fig. 6 compares the optimization achieved in accuracy and prediction time by both approaches. On average, *ADABTics* optimizes accuracy by 6.2% as compared to 4.5% optimization for *QuickAdapt*. The average optimization in prediction time is 26.1% for *ADABTics* and 17.6% for *QuickAdapt*. The optimization comparison is aligned with the adaptation accuracies for two approaches. *ADABTics* always selects a configuration with the best QoS, which is calculated based on accuracy and prediction time. Therefore, optimization with *ADABTics* is higher than *QuickAdapt*. Whilst the adaptation accuracy for *QuickAdapt* is lower than *ADABTics*, the gain in adaptation time (i.e., 105 times) for *QuickAdapt* is far significant than the marginal lose (i.e., 30%) in adaptation accuracy.

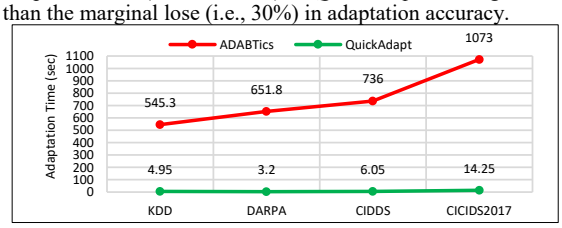

Fig 5. Adaptation Time for QuickAdapt Vs ADABTics [8]

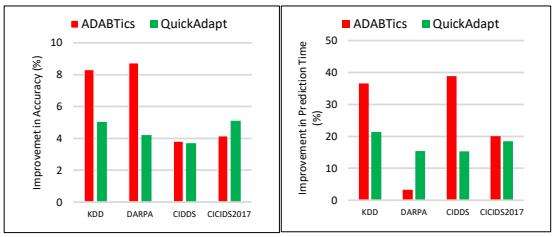

Fig. 6. Optimization Comparison for QuickAdapt Vs ADABTics

V. Related Work

Big Data Cyber Security Analytics (BDCA): Several Hadoop-based BDCA systems (e.g., [6, 7, 22, 23]) have been presented. Among these, [23] is evaluated with only one ML algorithm while the rest with two ML algorithms. The evaluation of these systems is limited from the perspective of evaluation dataset too. Except [6], the rest of the studies used only one dataset. [7] and [23] have been evaluated on a single machine. Only [22] and [23] focus on software architecture. *Adaptation for QoS Optimization:* Several studies (e.g., [8-10, 24]) have explored the adaptation for QoS optimization. Calinescu et al., [9] use KAMI model based on Bayesian estimator to estimate model parameters for adapting a service-based system. The approach proposed in [10] uses a hierarchical model for choosing network QoS-parameters to adapt mobile computing system. Edwards et al., [24] propose monitoring each component of a robotic system and triggering adaptation when a component's performance degrades. *Optimizing Adaptation Time:* The adaptation approach proposed in [8] takes around 751.5 sec on average to find the optimal configuration from the same search space of 32 configurations also considered in our study. Several recent studies [25-27] also faced the same issue of large adaptation time and proposed solutions accordingly. Alipourfard et al., [25] employed Bayesian optimization to reduce the adaptation time by up to 75%. Hill et al., [26] used hill climbing to select an optimal configuration, which reduces the adaptation time by around 50%. Jiang et al., [27] exploit spatial and temporal correlations among various configurations to quickly select the optimal configuration for video analytics, hence, reducing adaptation time by 2-3 times. Unlike previous works, our work is in a different domain, employs a different adaptation approach (i.e., using fuzzy logic and descriptive statistics), and is evaluated with variety of ML algorithms and security datasets. Since, [8] is the only work in the domain of BDCA, we have directly compared our adaptation approach with [8] in §IV.B.

VI. Conclusion

We presented *QuickAdapt*, an adaptation approach that uses descriptive statistics of security events data and fuzzy rules to enable a BDCA system to quickly adapt to the changes in security events data. *QuickAdapt* reduces the adaptation time by a factor of 105 (i.e., 105×) with minimal impact on the adaptation accuracy. *QuickAdapt* relies on manually defined fuzzy rules. In future, we plan to employ machine learning approaches for automatically generating fuzzy rules. Although *QuickAdapt* is designed to enable fast and accurate security analytics, the techniques (e.g., fuzzy rules and control actions) presented in this paper can be customized and applied to other domains such as traffic optimization and Internet-of-Things.

References

1. Cárdenas, A.A., P.K. Manadhata, and S. Rajan, *Big data analytics for security intelligence. Available at https://goo.gl/wxKqDV.* 2013.
2. Zuech, R., T.M. Khoshgoftaar, and R. Wald, *Intrusion detection and big heterogeneous data: a survey.* Journal of Big Data, 2015.
3. Ullah, F. and M.A. Babar, *Architectural Tactics for Big Data Cybersecurity Analytics Systems: A Review.* Journal of Systems and Software, 2019.
4. Kim, G., S. Lee, and S. Kim, *A novel hybrid intrusion detection method integrating anomaly detection with misuse detection.* Expert Systems with Applications, 2014: p. 1690-1700.
5. Mazel, J., et al., *Hunting attacks in the dark: clustering and correlation analysis for unsupervised anomaly detection.* Journal of Network Management, 2015.
6. Chen, T., et al., *Efficient classification using parallel and scalable compressed model and its application on intrusion detection.* Expert Systems with Applications, 2014.
7. Rathore, M.M., A. Ahmad, and A. Paul, *Real time intrusion detection system for ultra-high-speed big data environments.* The Journal of Supercomputing, 2016(9).
8. Ullah, F. and M.A. Babar, *An Architecture-driven Adaptation Approach for Big Data Cyber Security Analytics.* International Conference on Software Architecture, 2019: p. 41-50.
9. Calinescu, R., et al., *Dynamic QoS management and optimization in service-based systems.* Transactions on Software Engineering, 2010.
10. Chuang, S.-N. and A.T. Chan, *Dynamic QoS adaptation for mobile middleware.* Transactions on Software Engineering, 2008.
11. Lopes, R.H., I. Reid, and P.R. Hobson, *The two-dimensional Kolmogorov-Smirnov test.* International Workshop on Advanced Computing and Analysis Techniques in Physics Research, 2007.
12. Wang, L.-X. and J.M. Mendel, *Generating fuzzy rules by learning from examples.* Transactions on systems, man, and cybernetics, 1992. **22**.
13. Villegas, N.M., et al. *A framework for evaluating quality-driven self-adaptive software systems*. in *Symposium on Software engineering for adaptive and self-managing systems*. 2011.
14. Holcomb, Z.C., *Fundamentals of descriptive statistics*. 2016.
15. Ullah, F. and M.A. Babar, *Quantifying the Impact of Design Strategies for Big Data Cyber Security Analytics Systems: An Empirical Investigation. Available at https://bit.ly/2V6lShQ.* 2019.
16. Hadoop, A., *Apache Hadoop. https://goo.gl/GLWG9Q*. 2009.
17. Mahout, A., *Apache Mahout. https://goo.gl/uUVJER*. 2017.
18. KDD, *KDDcup99 Knowledge discovery in databases. https://goo.gl/Jz2Un6*. 1999.
19. MIT, *DARPA intrusion detection evaluation data set. Available at https://goo.gl/jYBYNe*. 1998.
20. Ring, M., et al., *Flow-based benchmark data sets for intrusion detection.* ECCWS, 2017.
21. Sharafaldin, I., A.H. Lashkari, and A.A. Ghorbani, *Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization.* ICISSP, 2018.
22. Marchal, S., X. Jiang, and T. Engel, *A big data architecture for large scale security monitoring.* Congress on Big Data, 2014.
23. Las-Casas, P.H., et al., *A Big Data architecture for security data and its application to phishing characterization.* 2016.
24. Edwards, G., et al. *Architecture-driven self-adaptation and self-management in robotics systems*. in *SEAMS*. 2009. IEEE.
25. Alipourfard, O., et al. *Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics*. in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 2017.
26. Hill, D.N., et al. *An efficient bandit algorithm for realtime multivariate optimization*. in *SIGKDD*. 2017. ACM.
27. Jiang, J., et al. *Chameleon: scalable adaptation of video analytics*. in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 2018. ACM.