A Role-Attribute Based Access Control Model for Dynamic Access Control in Hadoop Ecosystem

Hafsa Ait idar, Hicham Belhadaoui, Reda Filali and Olaf Malassé

Abstract—**Big Data is a term used to describe large and hard-to-handle amounts of data that overwhelm businesses on a daily basis. Apache Hadoop has emerged as a leading framework for storing and processing Big Data in a timely fashion. Hadoop is used in various sectors that often include private and sensitive data, which must be protected from unauthorized access. This paper proposes a new access control model called the Role-Attribute Based Access Control for Hadoop, referred to as H-RABAC, which inherits full advantages of RBAC and ABAC models. This hybrid model enforces access control policies dynamically based on user roles and data attributes. We further outline the necessary modification to XACML authorization policies to meet the security and privacy needs of Hadoop. In addition, we present a real-world use case and application of the H-RABAC model in the banking domain, followed by experimental results that show that the overhead imposed by our framework is minimal and acceptable for enforcing access control policies and protecting sensitive data in Hadoop.**

Index Terms—**Sensitive data, RBAC, ABAC, Access control, Authorization, XACML, Hadoop, Big Data.**

I. Introduction

WE are inundated with data. At present, a huge amount of data is produced by numerous sources, including social applications, sensors, Internet of Things (IoT) devices, business transactions, and many more [1]. It is estimated that more than 59 zettabytes of data will be created, generated, and consumed around the world this year [2]. Such massive collections of structured, semi-structured, and unstructured data, referred to as Big Data [3].

There is no doubt that Big Data has become an essential part of every organization. Companies collect, prepare and analyze a large volume of data to generate valuable insights, predict customer needs, and increase revenues [1]. Leveraging these data sets is not feasible with standard tools due to their diversity and complexity. This requires resilient clusters with high processing power and huge storage capabilities [4].

Big Data encompasses a wide variety of solutions. Hadoop is considered the most popular one. Hadoop is a distributed framework for storing and processing vast amounts of data based on three main components: HDFS for data storage, MapReduce for data processing, and YARN to manage cluster resources [5][6]. Due to its salient features, specifically scalability, cost-effectiveness, fault tolerance, high availability, etc., Hadoop has become ubiquitous for many fields, including healthcare, government, finance, telecommunications, to mention a few of them.

Organizations are using Hadoop to handle their data assets. Sensitive data that may reveal an individual's identity or a company's trade secrets are present among these vast amounts of data. For example, Personally Identifiable Information (PII) (credit card numbers, biometric data, etc.), financial data, customer and supplier information, and so on. Such data must be protected to avoid potential danger if recognized by undesired persons. However, securing this data within Hadoop seems to be increasingly challenging.

Despite Hadoop's popularity, security and privacy are serious concerns that could impede its adoption. To fulfill the security demands, researchers have investigated these issues and have realized the need to build appropriate security modules for Hadoop. Besides this, some Hadoop distribution vendors enhance the official release of Hadoop to add new functionalities. The major Hadoop distributions are Cloudera CDH-6.3.4 (Cloudera Distribution Hadoop), Hortonworks HDP-3.1.5 (Hortonworks Data Platform), and MapR. These distributions have added functionalities focusing on: support, reliability, governance, and security. Apache Sentry [7] and Apache Ranger [8] are the most used security projects introduced by Cloudera and Hortonworks to offer access controls across various Hadoop elements.

Regardless of different approaches provided to improve security in Hadoop, there is currently a lack of proper security models to protect sensitive data stored in Hadoop. To address this issue, we proposed in our previous works the Dynamic Data Sensitivity Access Control (D2SAC) framework [9]–[11]. The D2SAC is a dynamic and automated framework that secures sensitive data as long as it resides in the HDFS. In this paper, we continue enhancing our work by proposing a hybrid access control model which integrates roles and attributes to provide strong and flexible access control. This new model is called Role-Attribute Based Access Control (H-RABAC) for Hadoop. We further propose an extension of XACML authorization policies to control access first to Hadoop services and then to objects within a Hadoop Service. To the best of our knowledge, our paper is the first work to propose a hybrid access control model with an extension of XACML policies in the context of Hadoop security.

The remainder of this paper is organized as follows: Section 2 contains an overview of security issues in Big Data and access control models in Hadoop. We present the proposed D2SAC framework and its components in Section 3. In Section 4, the proposed Role-Attribute Based Access Control (H-RABAC) model is explained in detail, followed by a definition of XACML authorization policies in Section

Manuscript received December 26, 2021; revised December 28, 2022.

H. Ait idar is a PhD candidate at the National High School for Electricity and Mechanics, University of Hassan II, Casablanca, Morocco. E-mail: (hafsa.aitidar93@gmail.com).

H. Belhadaoui is a professor at the National High School for Electricity and Mechanics, University of Hassan II, Casablanca, Morocco. E-mail: (belhadaoui_hicham@yahoo.fr).

R. Filali is a professor at the National High School for Electricity and Mechanics, University of Hassan II, Casablanca, Morocco. E-mail: (filalihilalireda@gmail.com).

O. Malassé is a professor at ENSAM/ParisTech, METZ, France. E-mail: (olaf.malasse@ensam.eu).

5. Section 6 describes the method for implementing the H-RABAC model, and Section 7 provides an example of applying this model. Section 8 contains the evaluation and results of testing the D2SAC framework. Section 9 gives the conclusion of our work and offers a future research direction.

II. Literature Review

Data security and privacy problems have been a grave concern since the early '70s [12]. Traditionally, securing an organization's data relies on three basic security principles: confidentiality, integrity, and availability, widely known as the CIA triad [12][13]. Confidentiality is a security property focusing on protecting data from unauthorized access, disclosure, and theft. Integrity is a concept dealing with protecting data against unauthorized modifications over its lifecycle. Finally, availability is a principle that ensures that data is available to authorized users whenever they need it. Several methods have been proposed to address privacy and security issues over the last years, including authentication mechanisms, access control models, data encryption, data anonymization, etc.

At this time, businesses are increasingly embracing Big Data to extract valuable knowledge from all the available data and predict new trends using a variety of analytics platforms. Unfortunately, most of the Big Data tools are open source and were not designed with security and privacy in mind. Furthermore, traditional security methods cannot meet the volume, variety, velocity, and veracity of Big Data. Therefore, securing Big Data platforms requires scaling existing techniques and developing more advanced solutions. Considering the current researches [14][15][5][16], Big Data security and privacy issues are divided into five categories: Hadoop security, cloud security, monitoring and auditing, key management, and anonymization [14][15].

Hadoop [17] is an open-source framework that was initially designed to handle huge amounts of data in a trusted environment, and thus, security was not considered here [18][14]. The significant growth of Hadoop in organizations managing highly sensitive data caused this framework to be vulnerable to several attacks [5][19], for instance, impersonation attacks, Denial of Service (DoS), CROSS-SITE scripting (XSS), etc. According to authors in [5], Hadoop vulnerabilities are grouped into software, web interface, and network vulnerabilities. Yet these issues can be avoided by providing efficient security measures to ensure Authentication, Authorization, and Auditing, also known as the three A's of security.

The early distributions of Hadoop use the POSIX style permissions and Access Control Lists (ACLs) to specify access to files and directories stored in HDFS [4][20]. Such basic access control is no longer enough because of many attacks targeting data in HDFS. Thus, various security approaches are discussed in [14][19][15], mainly the Kerberos protocol to achieve powerful authentication using symmetric-key cryptography [5]. The Bull eye algorithm to secure and monitor all sensitive information, this approach scans the data initially before its storage into the data node and allows only permitted persons to read or write sensitive data [21][19]. In addition, the Name Node Security Enhance (NNSE) method is used to handle name node issues in HDFS. This method suggests using two name nodes (one is the master and the other is the slave) to assure data availability in a secure way when the master node becomes unavailable [21][14][19].

Furthermore, various tools and solutions were added to enhance security in Hadoop [5][18]. Apache Ranger and Apache Sentry are two important open source security frameworks and pluggable authorization engines used to provide access control in the Hadoop cluster [4][22]. Apache Ranger offers a centralized platform to authenticated users (via Kerberos) to set up and manage security policies for Hadoop resources (HDFS, HBase, Hive, etc.). It also supports fine-grained authorization and centralizes auditing of user access within all Hadoop components [4][9]. Apache Sentry provides fine-grained and role-based authorization for both data and metadata within Hadoop as well as it supports multi-tenant administration [4][7]. Apache Knox Gateway [23] is further used to provide a single access point to secure Hadoop services from unauthorized users. It integrates with the main authentication systems such as Kerberos, LDAP, Active Directory (AD), etc., to simplify access to Hadoop clusters [4][5].

Researchers are increasingly concerned with Hadoop security issues. There have been numerous papers that discuss the major security challenges in Hadoop. Authorization and access control are considered the most important security aspects to protect data from unauthorized access and determine what users can do with data.

The Role-Based Access Control (RBAC) model was seen as a suitable approach to manage users' access permissions for a while. Thus, using roles reduces the administrative tasks for organizations and allows businesses to determine their constraints with more flexibility [24]. However, due to the rapid growth of data and users, utilizing roles in Big Data applications is no longer sufficient. The Attribute-Based Access Control (ABAC) model has received increased intention because it provides great flexibility in making access decisions [25]. In [26], NIST presented strategies to embed roles with attributes, resulting in three approaches: dynamic roles, attribute-centric, and role-centric [27], which significantly inspired our contribution in this paper.

In [25], authors discuss the privacy issues in Big Data context and consider how ABAC technology can protect sensitive data against unauthorized access. A fine-grained access control framework for Hadoop called Vigiles is proposed in [28] to protect data from unauthorized access. Authors in [22] introduce a formal multi-layer access control model, referred to as HeAC, for Hadoop ecosystem based on native access controls capabilities of Hadoop and authorization models of Apache Ranger and Apache Sentry. They also present an Object-Tagged Role-Based Access Control (OT-RBAC) model, which preserves the advantages of RBAC model and offers support for object attributes. Besides that, an extension of OT-RBAC model is proposed in [4] to address the security requirements for Hadoop. Thus, authors present a fine-grained Attribute-Based Access Control model called HeABAC, including the concept of cross Hadoop services trust. A prototypical implementation for both OT-RBAC and HeABAC models is introduced using Apache Ranger. Reddy in [29] presents an access control framework to secure sensitive data in HDFS, which depends entirely on the data owner's guidelines. A fine-grained access control policy is

proposed in [30] to ensure Big Data security in the cloud. The authorization to access data in this approach depends only on the data owners. In [20], authors propose a Content Sensitivity-Based Access Control (CSBAC) framework to prevent unauthorized access to sensitive information. CSBAC uses the data sensitivity estimator to estimate the sensitivity value of the data item with minimal user intervention, but when the data has not been previously encountered, the domain expert must be involved to train the neural network.

In what follows, we present an overview of the D2SAC framework and its components. We then focus on the H-RABAC model, the access control model of D2SAC.

III. Proposed D2SAC framework

Assuring data protection in the Hadoop environment is a complex problem that often depends on several techniques. To address this issue, we proposed in our previous works the Dynamic Data Sensitivity Access Control (D2SAC) [9]–[11] framework that aims to protect sensitive data from unauthorized access in the HDFS. The D2SAC is an automated framework that uses the data itself to calculate its sensitivity and requires no intervention of data owners to make access control decisions. It also provides end-to-end security by keeping information protected as long as it remains in the HDFS. Two scenarios are embedded into our framework to ensure sensitive data protection, as shown in Figure 1.

Scenario 1: users can load data into HDFS in several different ways. When a user wants to create a data in the HDFS (creation request), the sensitivity calculation process is triggered. The Sensitivity Estimator Module (SEM) is thus invoked to calculate the sensitivity value of each data in an automated way without any effort from the data owner. The SEM uses the mathematical model proposed in [11] based on the AHP method [31] and the empirical average to determine the data sensitivity, as shown in Figure 2. Then, the sensitivity value is saved in the MetaDatabase along with other metadata information managed by the Metadata Generator Module (MGM).

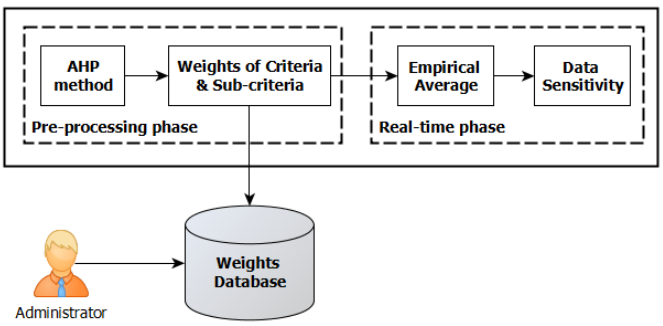

Fig. 2. Sensitivity Estimator Module.

Scenario 2: when a user intends to access a data that is already located in the HDFS (access request). In this case, the Access Enforcement Module (AEM) receives the user's access request and returns the decision about this request (entitled to access or not). This decision is based on the H-RABAC model that will be explained in the next section.

IV. H-RABAC model for Hadoop ecosystem

Access control is one of the essential requirements to limit unauthorized access to data and protect resources in the Hadoop environment. RBAC model is known to grant rights to users based on their business activity, called a role, while ABAC model provides a flexible and fine-grained approach. It defines authorizations based on a set of characteristics of subject, object, and environment. ABAC is helpful in distributed and rapidly growing environments like Hadoop environment and for cases where policy management becomes tedious. For example, instead of a person with the manager role always being able to access information, ABAC may place additional restrictions on his access, such as allowing access only at specific times or from a particular location or IP address. This can enable creating policies with detailed permissions to grant less privilege and thus reduce security issues.

In this section, we present a new Role-Attribute Based Access Control model for Hadoop (H-RABAC) that takes full advantage of RBAC and ABAC models. The main concept of the H-RABAC model is that users are assigned to roles (as in the RBAC model), users and objects are associated with a set of attributes (as in the ABAC model). In addition to fully integrating RBAC and ABAC, the new elements added to our model include roles with attributes and authorization decisions based on extended XACML policies that will combine roles, user attributes, role attributes, and object attributes.

We will now define the conceptual model of H-RABAC, as shown in Figure 3, followed by formal definitions as specified in Table I.

The basic components of H-RABAC model are as follows: Users (U), Groups (G), Subjects (S), Roles (R), Hadoop Services (HS), Data and Objects (OB) belonging to Hadoop Services, and Operations (OP) on objects.

Users, Groups, Subjects and Roles: A user is a person seeking access to services and data within Hadoop. A group is a set of users with the same needs and job requirements. A subject is a process executed on behalf of the user to fulfill operations in the Hadoop cluster. The subject is always executed with full privileges of its creator. A role is a set of permissions and privileges assigned to users and groups in the system.

Hadoop Services: Hadoop ecosystem includes several components like HDFS, HBase, Hive, Yarn, etc. Each Hadoop service supports different types of data and objects (file, table, topics, etc.) that users intend to access. It should be noted that accessing service is primarily required before accessing the supported data.

Objects and Data: An object is a resource managed by a Hadoop service inside the cluster that requires protection from unauthorized access. Examples of such resources are files in HDFS, tables in Hive, etc.

Operations: An operation is an action that authorized users could perform on data and objects. Each Hadoop service supports several operations. For example, HDFS has read or write operations, Hive has create, select, alter operations, and so on.

As shown in Table I, most of the relations in our model are many-to-many. A user can have multiple roles $\{r_1...r_n\}$ and each role can be assigned to many users. The URA relation highlights the benefit of the RBAC model where the management of users' rights becomes easier by simply granting or removing roles. A user can also be a member of

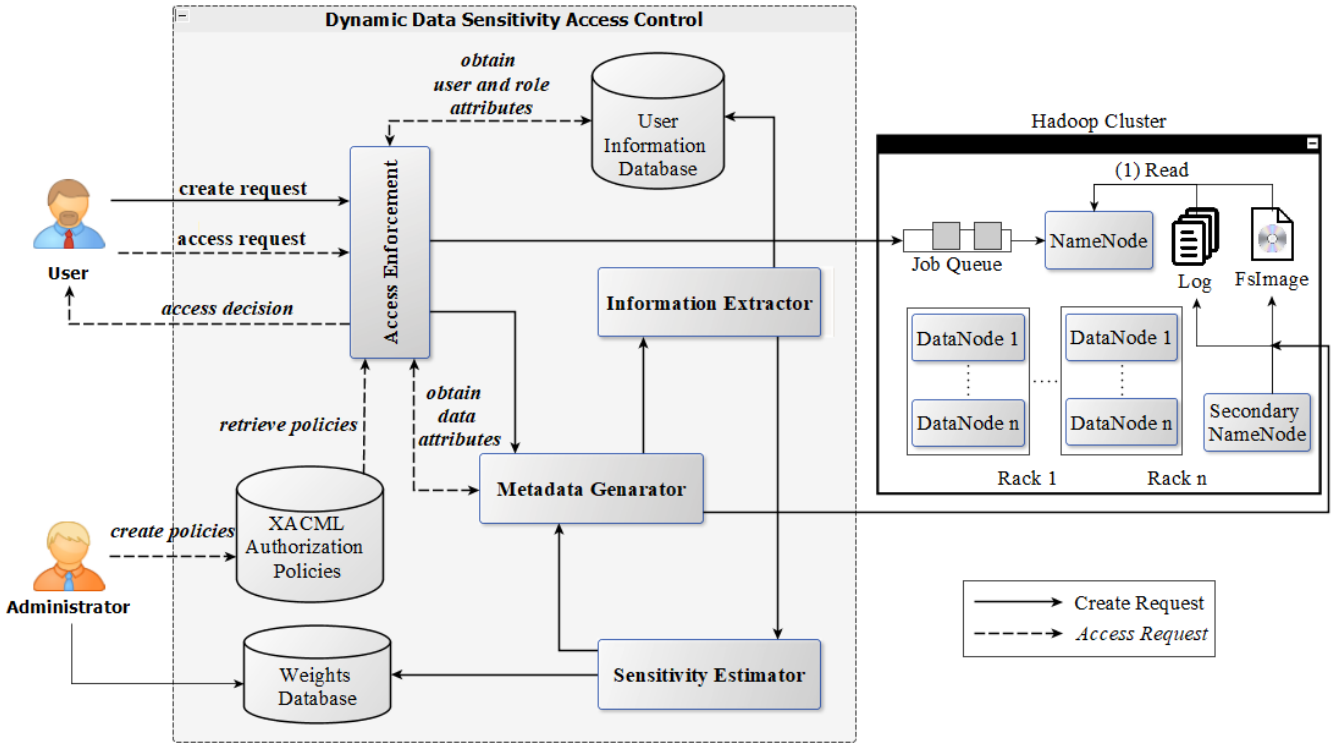

Fig. 1. Proposed D2SAC.

several groups, and a group may contain an outlined number of users. The prime advantage of the UGA relation is the ability to assign and remove various attributes and roles from users conveniently. Similarly, each group may be assigned to multiple roles and each role can be attributed to different groups as reflected by the GRA relation. A Hadoop service may include numerous objects and an object can be used by different Hadoop services as shown by the OHS relation. Finally, several operations can be performed on objects inside a Hadoop service.

Attributes are characteristics that are assigned to different entities in our model. User attributes (UA) is the set of attributes assigned to users, groups and subjects. Object attributes (OBA) is the set of attributes assigned to data and objects. Role attributes (RA) is the set of attributes assigned to roles. Each attribute is a function that takes an entity (U, G, OB, R) and returns one or more values from its range, denoted by $Range(att)$. The range of an attribute consists of a finite set of atomic values. The attribute functions in UA, OBA and RA map U, OB and R respectively to one or a subset of attribute values in the range according to the atomic-valued or set-valued attribute type.

Users are assigned to several groups (UGA relationship) to simplify the administration of roles and attributes. When a user becomes a member of a group, he inherits all the roles and attributes of this group. Therefore, the effective user attributes will then be the union of all attributes assigned directly to users $(ua \in UA)$ and attributes inherited through group membership $(ua(g), \forall g \in UGA(u))$. The effective roles of user will then be the union of all roles assigned directly to users (URA) and roles inherited through group membership $(GRA(g), \forall g \in UGA(u))$, as shown in Table I.

Furthermore, a subject created by a user (denoted by US function) may have a subset or all of the values of the effective roles and attributes of its user creator. It is required that the subject attributes must not exceed the attributes of the user creator $(effectiveA(s) \subseteq effectiveA(US(s)))$ and that the subject roles must not exceed the roles of the user creator $(effectiveR(s) \subseteq effectiveR(US(s)))$.

In addition, XACML authorization policies are defined based on the H-RABAC model elements to control access to Hadoop services and then to Hadoop objects.

V. Authorization Policies for H-RABAC

We specify fine-grained access control rules and policies to control every access to resources in the Hadoop services (HS). The common policy language, XACML (eXtensible Access Control Markup Language) [32], is used to define authorization policies based on a set of characteristics of subject, object, resource, and environment to determine whether an authorization decision is approved or denied. The XACML policy model consists of the following elements, as shown in Figure 4:

- Rule is the fundamental element of a policy. It includes a target, an effect, a condition, and possibly a set of obligations or advices.
- Policy is expressed by a target, a set of rules, a rule-combining algorithm, and may be a set of obligations or advices allowing users to obtain a decision according to a set of parameters.
- Policy Set is an aggregation of several policies and other policy sets. It also contains a policy-combining algorithm to combine the decision of policies and possibly a set of obligations and advices.
- Target identifies a set of requests that a rule, policy, or policy set is supposed to evaluate based on attributes

TABLE I
FORMAL DEFINITIONS OF H-RABAC MODEL

Basic Sets
- U is a finite set of users.
- G is a finite set of groups.
- R is a finite set of roles.
- S is a finite set of subjects.
- HS is a finite set of Hadoop Services.
- OB is a finite set of objects.
- OP is a finite set of operations.

Basic Relations and Functions
- $URA \subseteq U \times R$, a many-to-many mapping of user-to-role assignments, equivalently $U \rightarrow 2^R$
- $GRA \subseteq G \times R$, a many-to-many mapping of group-to-role assignment, equivalently $G \rightarrow 2^R$
- $UGA \subseteq U \times G$, a many-to-many mapping of user-to-group assignments, equivalently $U \rightarrow 2^G$
- $OHS \subseteq OB \times HS$, a many-to-many mapping of objects to Hadoop services, equivalently $OB \rightarrow 2^{HS}$
- $OBP \subseteq OB \times OP$, a many-to-many mapping of each object to a set of operations, equivalently $OB \rightarrow 2^{OP}$
- UA is a finite set of user attribute function.
- OBA is a finite set of object attribute function.
- RA is a finite set of role attribute function.
- For each attribute att in $UA \cup RA \cup OBA$, Range(att) represents a finite set of atomic values.
- attType: $UA \cup RA \cup OBA$ = {set, atomic}, determines attributes as set or atomic valued.
- Each attribute function ua in UA maps a user or group (U or G) to one or a set of attribute values in the Range(ua) based on atomic or set-valued attribute type. Formally,

$$\forall ua \in UA, ua : U \cup G \rightarrow \begin{cases} Range(ua) & \text{if attType(ua) = atomic} \\ 2^{Range(ua)} & \text{if attType(ua) = set} \end{cases}$$

- Each attribute function oba in OBA maps an object (ob in OB) to one or a set of attribute values in the Range(oba) based on atomic or set-valued attribute type. Formally,

$$\forall oba \in OBA, oba : OB \rightarrow \begin{cases} Range(oba) & \text{if attType(oba) = atomic} \\ 2^{Range(oba)} & \text{if attType(oba) = set} \end{cases}$$

- Each attribute function ra in RA maps a role (r in R) to one or a set of attribute values in the Range(ra) based on atomic or set valued attribute type. Formally,

$$\forall ra \in RA, ra : R \rightarrow \begin{cases} Range(ra) & \text{if attType(ra) = atomic} \\ 2^{Range(ra)} & \text{if attType(ra) = set} \end{cases}$$

Effective Roles and Attributes of User
- $effectiveR : URA \subseteq U \times R$, defined as $effectiveR(u) = URA(u) \cup (_{\forall g \in UGA(u)} \cup GRA(g))$
- For each attribute ua in UA:

$$effectiveA(u) = ua(u) \cup (_{\forall g \in UGA(u)} \cup ua(g))$$

Effective Roles and Attributes of subject
- US: $S \rightarrow U$, a function mapping each subject $s \in S$ to its user creator $u \in U$.
- For each attribute ua in UA and s in S:

$$effectiveA(s) : S \rightarrow \begin{cases} Range(ua) & \text{if attType(ua) = atomic} \\ 2^{Range(ua)} & \text{if attType(ua) = set} \end{cases}$$

and $effectiveA(s) \subseteq effectiveA(US(s))$
- $effectiveR : S \rightarrow 2^R$, mapping each subject $\forall s \in S$ to a set of roles and $effectiveR(s) \subseteq effectiveR(US(s))$.

Authorization Policies
- ***Global Policy:*** As its name suggests, the global XACML policy renders the authorization decision based on the result returned by Policy 1 and Policy 2.
- ***Policy 1:*** Authorization policy to control access to Hadoop services.
- ***Policy 2:*** Authorization policy to control access to data and objects inside a Hadoop service.

Access Decision
- A user $u \in U$ (or its subject $s \in S$) with the role $r \in R$ is authorized to access a Hadoop Service $hs \in HS$ if attributes of user u and attributes of its role $r \in R$ satisfy the defined Policy 1(P1), Formally, $P1_{access}(u : U, r : R, hs : HS) = Permit$.
- A user $u \in U$(or its subject $s \in S$) with the role $r \in R$ is permitted to perform an action on an object $ob \in OB$ in a Hadoop service $hs \in Hs$ if attributes of user u, attributes of its role r and attributes of object ob satisfy Policy 2(P2). Formally, $P2_{operation}(u : U, r : R, ob : OB, hs : HS) = Permit$.
- A user $u \in U$ (or its subject $s \in S$) with the role $r \in R$ is authorized to execute an operation on an object $ob \in OB$ in a Hadoop Service $hs \in HS$ if and only if both decisions of P1 and P2 are evaluated to permit. Formally, $P1_{access}(u : U, r : R, hs : HS) = Permit$ AND $P2_{operation}(u : U, r : R, ob : OB, hs : HS) = Permit$.

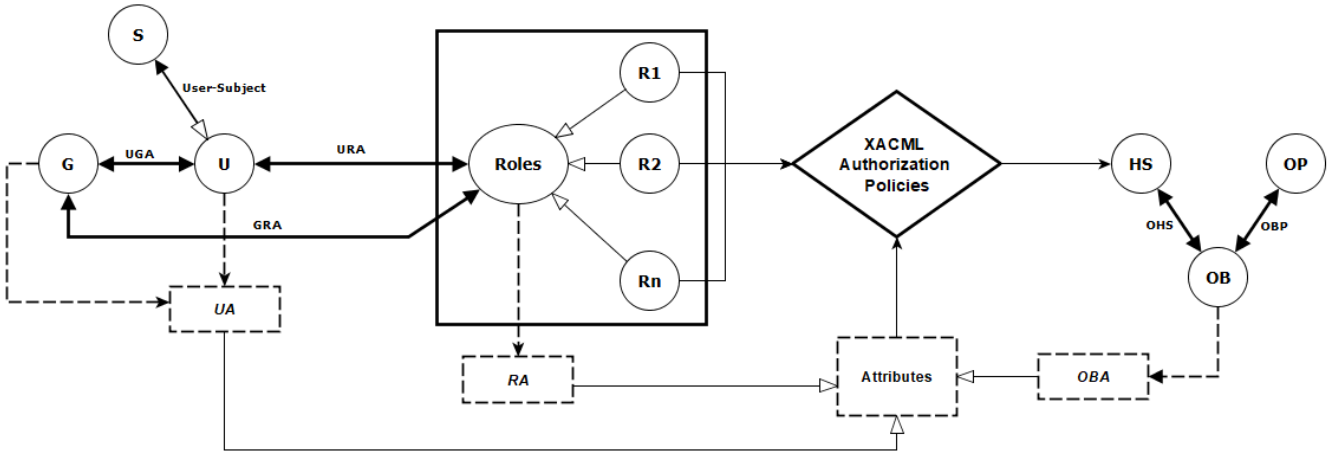

Fig. 3. Conceptual model of H-RABAC.

describing the subjects, resources, actions, and environment of the request.
- Condition is a boolean function that can be added to a rule to further define the applicability of this rule.
- Effect determines the result of evaluating the request. It is either "permit" or "deny".
- Obligation is an optional element that is provided along with an authorization decision to enhance that decision (either permit or deny).
- Advice is an additional piece of information in a rule, policy, or policy set that is returned with an authorization decision (permit or deny) and can be ignored.
- Policy-combining algorithm is the procedure of combining authorization decisions from several policies.
- Rule-combining algorithm is the procedure of combining authorization decisions from different rules. Some examples of these algorithms are Deny-overrides, Permit-overrides, First-applicable, etc.

To express permissions within the H-RABAC model, a modification of XACML is required. XACML defines policies based on attributes, but it does not directly support RBAC. Because of this, we extend the original XACML to add new elements and modify some existing elements.

In the H-RABAC model, any user or application should be first authorized to access the Hadoop service before performing operations on its object. To meet this security requirement, we define a global XACML policy that consists of two individual authorization policies, one to control access to Hadoop services and the other to manage access to resources and objects within Hadoop services.

A. Global XACML policy

Access control policies in the Hadoop environment need to be flexible and dynamic. For this purpose, we propose a global policy that consists of a policy set element with an empty target to match any user request and two main policies: Policy 1 to control access to Hadoop services and Policy 2 to control access to objects inside a Hadoop service, as shown in Figure 5. This global policy can combine various sub-policies with different rules and manage conflicts between these policies. Conflicts can be between different policies or rules used in a policy.

A user $u \in U$ (or its subject $s \in S$) with the role $r \in R$ is authorized to execute an operation on an object $ob \in OB$ in a Hadoop service $hs \in HS$ if and only if both decisions of P1 and P2 are evaluated to permit. Formally,

$$P1_{access}(u : U, r : R, hs : HS) = Permit \wedge$$
$$P2_{operation}(u : U, r : R, ob : OB, hs : HS) = Permit.$$

B. Access Control Policy for Hadoop Services

D2SAC is a multi-layer authorization framework that controls access at the Hadoop service level and then at the data and object level. Thus, the first authorization layer checks whether a user is allowed to access a particular Hadoop service. For this reason, we define an authorization policy for each Hadoop service $hs \in HS$, as shown in Figure 6. This policy outlines under which conditions a user $u \in U$ with a role $r \in R$ can access a Hadoop service HS in the Hadoop cluster. Thus, significant changes are performed, notably in the policy's target and rule's target elements, respectively.

Policy 1 contains a target element that specifies that this policy only applies to requests asking for access to a specific Hadoop service. Once the proper policy has been verified, its rules are evaluated. To support the RBAC model directly in XACML, we add the role element in the rule's target and remove the resource element because this policy concerns only access to Hadoop services. Both user attributes and role attributes are used to make the access control decision. Based on these attributes, if the rule's condition is true, then the rule's effect returns a permit value giving access to that Hadoop service. Otherwise, if the condition is false, then the rule's effect returns a deny value meaning that access to that Hadoop service is not allowed.

A user $u \in U$ (or its subject $s \in S$) with the role $r \in R$ is authorized to access a Hadoop service $hs \in HS$ if attributes of user u and attributes of its role $r \in R$ satisfy the defined Policy 1(P1), Formally, $P1_{access}(u : U, r : R, hs : HS) = Permit$.

C. Access control policy for Hadoop Objects

Authorizing access to data and objects (resources) within a Hadoop service requires special handling since not all resources are treated equally. A second authorization layer

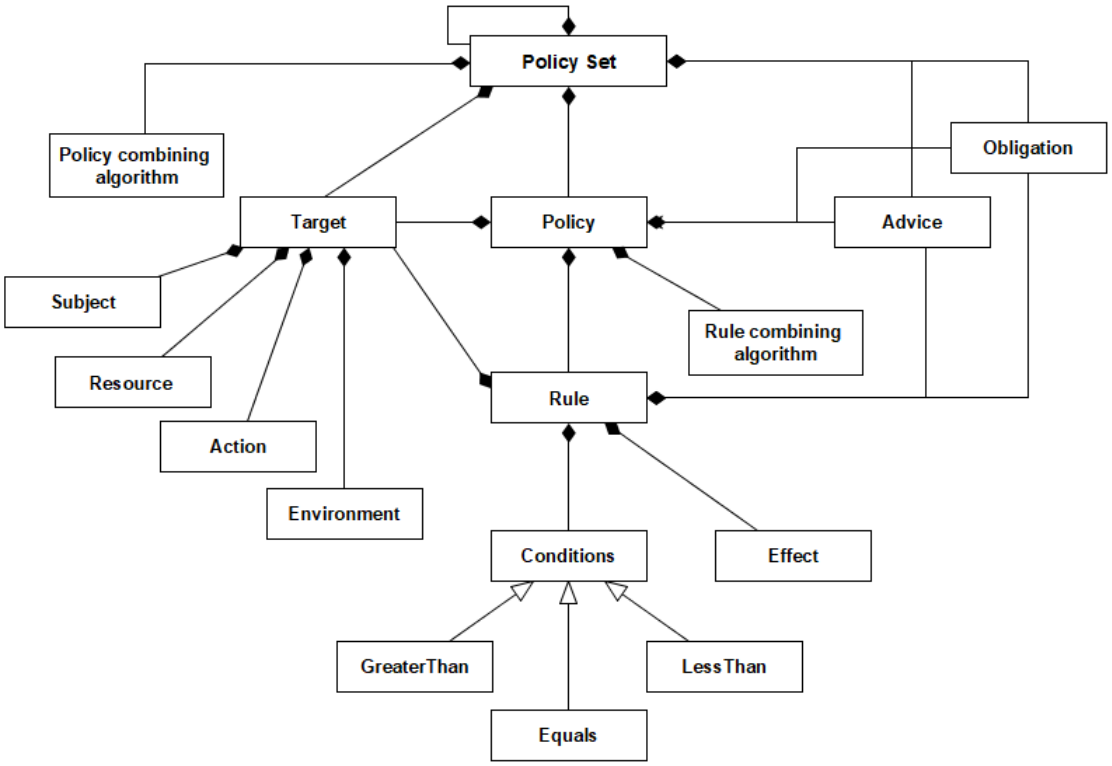

Fig. 4. Original XACML.

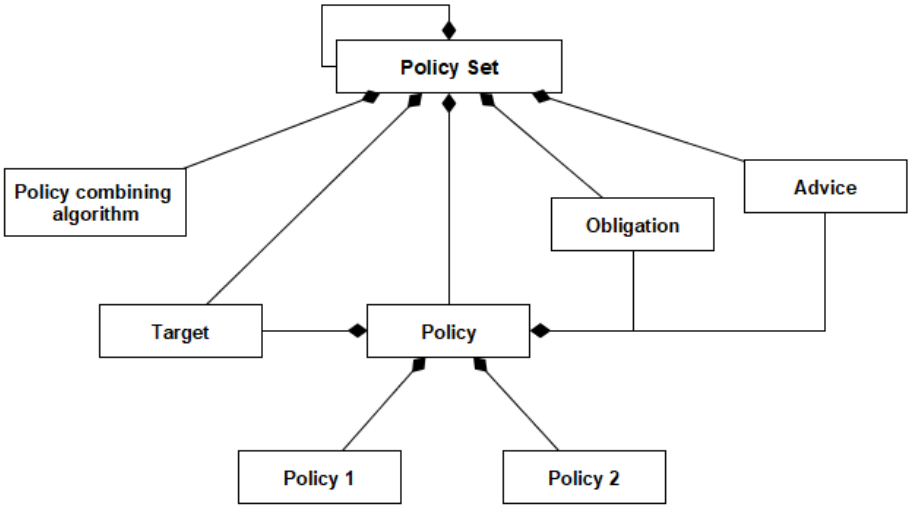

Fig. 5. Global XACML policy.

is proposed in our framework to check if a user has the necessary permissions to access particular data in a Hadoop service. Access control policies (Policy 2) are therefore created to determine under which conditions a subject $u \in U$ with a role $r \in R$ is authorized to perform an operation $op \in OP$ on an object $ob \in OB$ in a Hadoop service $hs \in HS$. Hence, significant changes are performed concerning the original XACML.

Because some data are more sensitive than others, data sensitivity and classification must be considered in this authorization layer to ensure data protection. Data classification is the process of categorizing data according to its degree of sensitivity. For this reason, a sensitivity value is calculated for each data using the proposed mathematical model in our previous paper [11]. Based on this value, a classification label is assigned to each data.

The policy's target element specifies that this authorization policy is defined for each classification level, as shown in Figure 7. In addition, in the rule's target element, we add the subject's role to reflect the benefits of the RBAC model, and each role is associated with a set of attributes.

To render the access control decision, the user attributes, the role of the user, its attributes (mainly the weight), and the object attributes (especially the sensitivity) are used. Based on these attributes, if the rule's condition evaluates to true, then the rule's effect returns a permit value giving the authorization to perform the demanded operation on that object. Otherwise, if the condition evaluates to false, then the

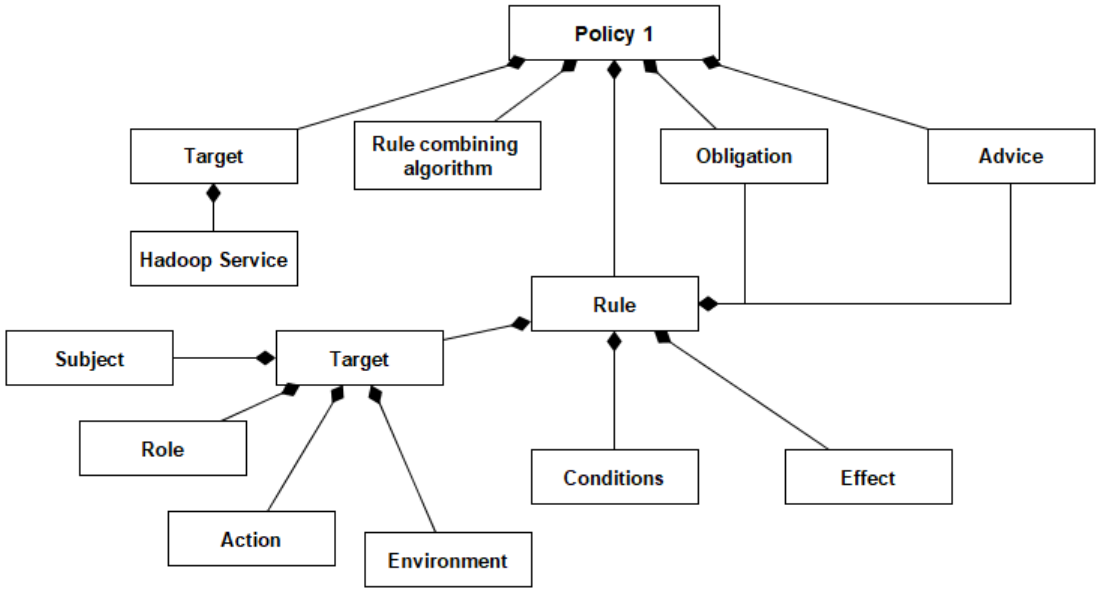

Fig. 6. Access control policy for Hadoop services.

rule's effect returns a deny value meaning that the requested operation on that object is not allowed.

A user $u \in U$(or its subject $s \in S$) with the role $r \in R$ is permitted to perform an action on an object $ob \in OB$ in a Hadoop service $hs \in Hs$ if attributes of user u, attributes of its role r and attributes of object ob satisfy Policy 2(P2). Formally, $P2_{operation}(u : U, r : R, ob : OB, hs : HS) = Permit$.

VI. H-RABAC IMPLEMENTATION METHOD

The H-RABAC model uses dynamic XACML policies to control access to each object in a Hadoop service. These policies are dynamic because the answer to the same question changes based on some dynamic element of the access context, such as time of the day, IP address, or location from which the user is trying to access an object.

The H-RABAC model incorporates user attributes, role attributes, and object attributes into the access request using a context enricher to provide fine-grained extensions. As its name suggests, the context enricher enhances the user's initial request with additional attributes and information used by the authorization policy conditions to approve or deny the access request. Therefore, before the policy engine evaluates the user's access request, the context enrichers are invoked to update the request's context with necessary extra information.

The Access Enforcement Module (AEM) is responsible for managing access in our framework. The AEM intercepts each user access request, then checks the stored authorization policies already defined by the administrator in order to return an access control decision. Thus, the procedure for making an access decision in our model consists of 2 phases, as shown in Figure 8.

1. Policy Enforcement Phase

Each user sends an access request that we call the Initial Access Request (IAR), which contains four authorization requirements, which are: the requester's identifier, the requested object, the action that the user wants to perform on that object, and the service he wants to access, as shown in (1).

$$IAR \Rightarrow \{username, object, action, Hadoopservice\} \quad (1)$$

After receiving this initial request, the context enricher enhances the IAR by adding additional user, role, and object attributes depending on the active policy conditions, as shown in (2). After that, the Enriched Access Request (EAR) is sent for evaluation. For example, suppose a user, Bob wants to access a report1 object to make changes, but the authorization policy specifies that no user can modify this object after 11:00 p.m. In that case, the context enricher will add the current time into Bob's access request. This enriched request will then be evaluated to return an access decision, granting access to Bob if the request complies with the time condition specified in the policy or denying access otherwise.

$$EAR \Rightarrow \{username, object, action, UA, RA, ObA, contextattribute...\} \quad (2)$$

2.Policy Evaluation phase

In this phase, the Enriched Authorization Request (EAR) is evaluated by the AEM, which uses a policy engine that compares the information in the EAR with the active security policies and then makes an authorization result, as shown in Figure 8. We explain each step (1)–(7) as follows:

1) During the pre-processing phase, the security administrators create XACML authorization policies that will be used for access control and store them in a database.
2) Each user sends an access request called the Initial Access Request (IAR), demanding access to a specific resource.
3) After receiving the IAR request, the AEM invokes the context enricher class to add attributes and needed information.
4) - 4') In this step, to build the EAR, user and role attributes are retrieved from the User Information Database, object attributes are retrieved from the Metadata Generator Module, and other environmental attributes are also added if necessary.

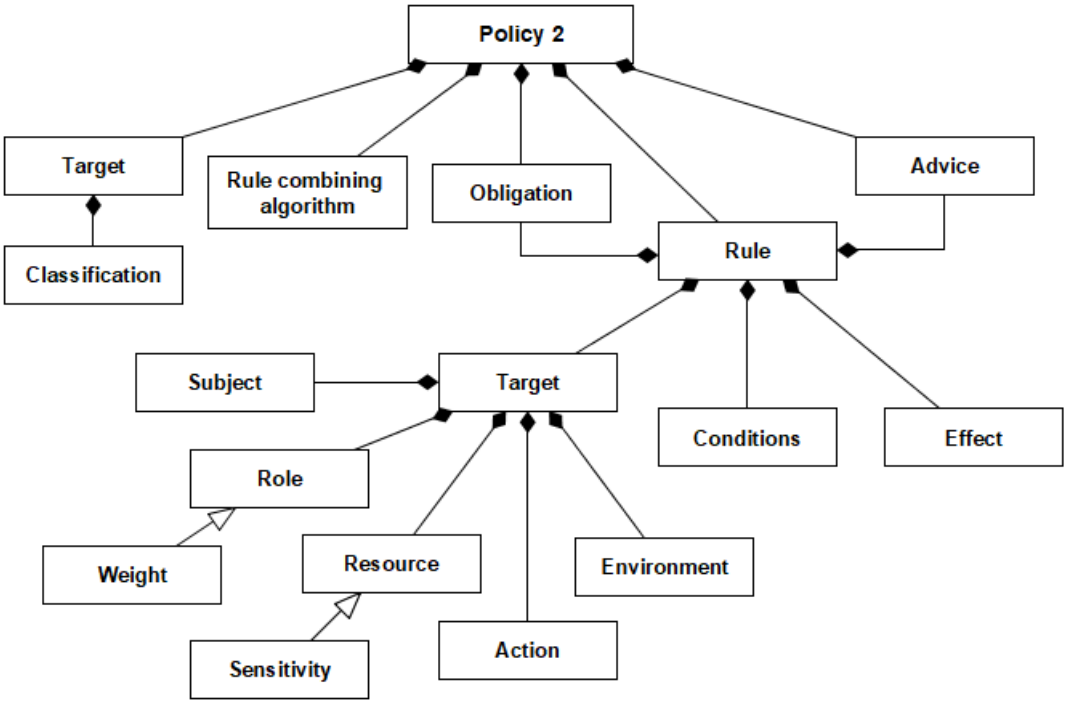

Fig. 7. Access control policy for Hadoop objects.

5) The EAR is sent to the policy engine for evaluation based on all these attributes.
6) The policy engine is the heart of the AEM. To evaluate the EAR requests, it retrieves the appropriate policies based on the policy's target element and compares the policy information against the EAR information to make an access decision.
7) Finally, based on this comparison, the access decision is sent to the requesting user, either permit or deny.

Our proposed implementation of the H-RABAC model includes defining the authorization rules and all XACML security policies using the Abbreviated Authorization Language (ALFA) [33]. This language is typically designed to provide a high-level description of XACML policies, facilitate writing and reading for XACML policy writers, and allow simple and quick verification of XACML policies by the company's stakeholders.

Therefore, we chose ALFA Plugin for Eclipse as a means of editing XACML policies. Thus, all policies presented in the next section are expressed using this plugin. We also used the open-source decision engine WSO2 Balana [34] to evaluate these policies. In addition, all incoming requests and context enrichers are developed using Java Language.

Figure 9 shows the part of the code to create the Enriched Access Request (EAR) based on the Initial Access Request (IAR). The received IAR ("Bob23", "/sensitiveData/annualCreditCardReport.csv", "read", "HDFS") clearly shows that the user Bob23 wants to read the "/sensitiveData/annualCreditCardReport.csv" file in HDFS. We first check if the demanded file is already stored in the HDFS. If so, then the necessary additional attributes of the user (role, department, experience...), role (weight, security level), object (sensitivity, classification...), and environment (location and time) are added to the IAR to build the EAR. After that, the Balana policy engine evaluates all incoming requests. After receiving the EAR request, we initialized the Balana engine that checks the stored security policies to find the appropriate policy whose target element matches the incoming request. Then, based on the comparison result, if the decision is equivalent to permit, the user can run its Spark job that displays the demanded data . Otherwise, a notification is sent to that user, asking him to contact the administrator to find the reason for this rejection, as shown in Figure 10 .

VII. H-RABAC MODEL APPLICATION

In this section, we will illustrate an application of the H-RABAC model in the real world. For that, we consider that data is already stored in the HDFS, and the sensitivity value is calculated for each data before enforcing the access control.

As demands for data protection in a moving environment escalate, several businesses are involved, especially financial institutions and banks that handle a wide variety of sensitive data.

Let us suppose a banking organization, which we call Organization A, uses Hadoop to store its continuously generated data. In this use case, we are interested in data issued from the finance department of Organization A as it is considered one of the critical departments dealing with highly sensitive data due to its primary functions including examining financial statements and reporting, accounting, forecasting budgets, managing internal and external accounts, etc. Thus, we specify access control rules and policies to control every resource access in Organization A. We define XACML policies to control access to both Hadoop service and objects inside a Hadoop service.

A. Global XACML policy

A global access policy is defined as shown in Figure 11 . This policy applies to any requests (hence the lack

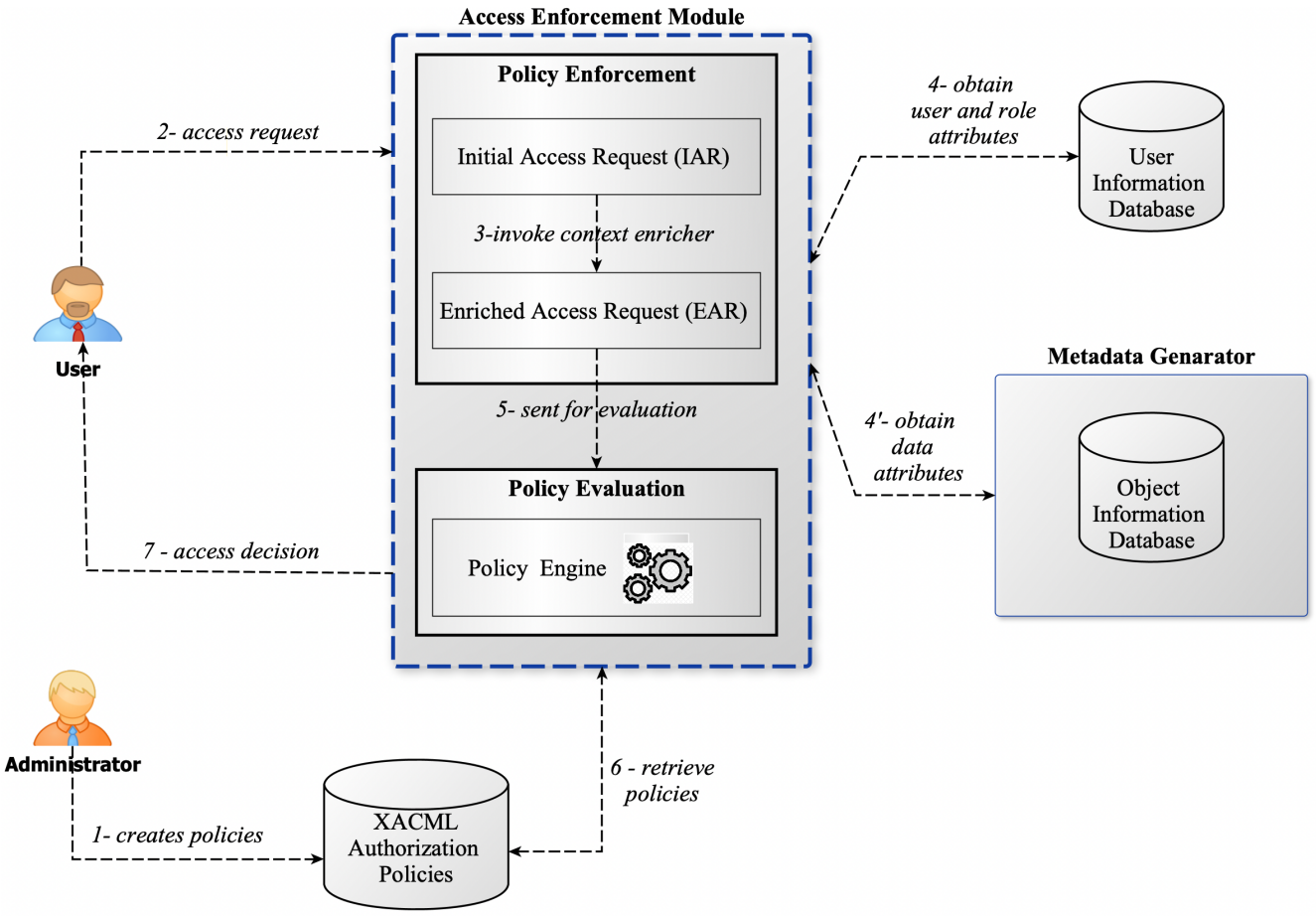

Fig. 8. Procedure for making an access control decision.

of the target element), it contains two referenced policies: *hadoopServicesAccess (P1)* and *hadoopObjectsAccess (P2)*, and it uses the *denyOverrides* policy-combining algorithm to combine the results of these policies. In this combining algorithm, if a decision returns deny, the final result is deny. The denyOverrides algorithm is therefore used in cases where a deny decision should have priority over a permit decision.

In addition, we define obligation and advice expressions in this policy. If the access had been authorized (on permit decision), a notification should be sent to the administrator, including a message and the name of the user who accessed the data. Otherwise, a notification must be sent to the requested user if access has been denied (on deny decision).

B. Access Control Policy for Hadoop Services

The first authorization layer within our framework controls access at the Hadoop service level. The hadoopServicesAccess policy (P1) only applies to requests asking for access to a specific Hadoop service. In this example, the target clause concerns the Hadoop service of type HDFS. As shown in Figure 12, the policy contains three rules and uses the *firstApplicable* rules-combining algorithm. In this algorithm, the order in which the rules are defined is significant because each rule will be evaluated according to its order of appearance in the policy. Thus, for a rule, if the target matches and the condition is true, the policy evaluation must stop, and the corresponding effect of the rule will be the result of the policy evaluation (permit or deny). Otherwise, if the target is false or the condition is false, the next rule in the order will be evaluated, and so on.

The hadoopServicesAccess policy allows access to the HDFS service for all users during working hours and blocks access for junior users during off-hours.

- The first rule allows access to HDFS between 7:00 pm and 7:00 am only for senior users. The access is denied for users with other roles, and a message is sent to the demanded users.
- The second rule allows access to HDFS service for users with any roles (junior or senior) as long as the user belongs to the finance department and is located in Paris.
- Finally, if none of the targets of the previous rules match, this last rule is applied to return a deny decision.

C. Access Control Policy for Hadoop Objects

The hadoopObjectsAccess policy (denoted as P2) concerns access to objects inside a Hadoop service. As shown in Figure 13, the policy is applicable to requests with secret resource classification. This policy contains two rules that define the conditions under which a user can read or edit secret data and uses the denyOverriders rules-combining algorithm to make a decision.

After controlling access to Hadoop services, the second layer provides access to Hadoop objects. Thus, the hadoopObjectsAccess policy (denoted as P2) concerns access to objects inside a Hadoop service. As shown in Figure 13,

```
String userName = "Bob23";
String operation = "read";
String resource = "/sensitiveData/annualCreditCardReport.csv";
String hadoopService = "HDFS";

try {
  Configuration conf = new Configuration();
  FileSystem fileSystem = FileSystem.get(new
      URI("hdfs://ec2-nameNode.eu-west-3.compute.amazonaws.com:9000"), conf);


  // Check if the file already exists
  Path path = new Path(resource);
  if (!fileSystem.exists(path)) {
    System.out.println("This file does not exist in HDFS.");
    return;
  }

  else {
    String classification= objectAttributes.getClassification(resource);
    double sensitivity= objectAttributes.getSensitivityOfData(resource);
    long resourceID = objectAttributes.getIdOfData(resource);
    String resourceProject = objectAttributes.getProject(resource);

    String department = userAttributes.getDepartmentOfUser(userName);
    String address = userAttributes.getAddressOfUser(userName);
    int experience = userAttributes.getExperienceOfUser(userName);
    String status = userAttributes.getStatusOfUser(userName);
    String userProject = userAttributes.getProjectOfUser(userName);

    String role= userAttributes.getRoleOfUser(userName);
    double weight = roleAttributes.getWeightOfRole(role);
    String securityLevel = roleAttributes.getSecurityLevelOfRole(role);

    java.time.LocalTime time = java.time.LocalTime.now();
    String location = ProjectConstant.subnet;

    String EAR = policy.createEnrichedAccessRequest(hadoopService, classification, role,
        operation, department, sensitivity, weight, time, location, address, experience,
        status, userName, resourceID, securityLevel, userProject, resourceProject);

    System.out.println("======= XACML Enriched Request =========");
    System.out.println(EAR);
    System.out.println("=======================================");
```

Fig. 9. Part of code to create the EAR.

the policy applies to requests asking for access to resources (objects) with a secret classification. This policy contains two rules, which define the conditions under which a user can read or edit secret data and uses the *denyUnlessPermit* rules-combining algorithm to make a decision. In this combining algorithm, if any decision is permit, the result will be permit. Otherwise, the result will be deny.

- The first rule allows users with junior or senior roles to read secret data during working hours (from 7:00 am to 7:00 pm) if and only if the user's role weight is equal to or greater than the sensitivity of the secret data and the user is working on the project the data is part of.
- The second rule permits senior users with a high security level and more than seven years of experience to edit data classified as secret or below only from the intranet (192.168.2.0), and if their role weight is equal to or greater than the sensitivity of the requested data.

Figure 14 illustrates an example of applying these defined policies. In this example, the user Bob with the direct attributes (*ID: 1001, Diploma: Financial Engineer, Department: Finance, Experience: 10 years, Project: Contribution*) is assigned to the Interne Group, thereby inheriting the attributes of this group (*CompanyName: OrganizationA, Address: Paris, Status : Interne*) which are added to his direct attributes. Further, each role in our model has a set of attributes. Bob is associated with the senior manager role having the following attributes (*ID: 34, Security-Level: high, Weight: 0.57*). In addition, when Bob creates a subject, it inherits a subset of Bob's attributes (*Role: Senior Manager, Department: Finance, Status: interne, Experience: 10 years*) as needed. The CreditCardReport object is also associated with a set of attributes (*Sensitivity: 0.36, Classification: Secret, Project: Contribution*).

Based on Bob's attributes, the subject created by Bob, the attribute of the senior manager role, and the attributes of the CreditCardReport object, it is clear that rules defined in

```
    policy.initBalana();
    PDP pdp = policy.getPDPNewInstance();
    String response = pdp.evaluate(EAR);

    System.out.println("============= XACML Response =============");
    System.out.println(response);
    System.out.println("==========================================");

    try {
      ResponseCtx responseCtx = ResponseCtx.getInstance(policy.getXacmlResponse(response));
      AbstractResult result = responseCtx.getResults().iterator().next();

      if(AbstractResult.DECISION_PERMIT == result.getDecision()){
        System.out.println("\n" + userName + " is authorized to " + operation + " the "
            +resource+ " \n\n");

        List<ObligationResult> obligations = result.getObligations();
        for(ObligationResult obligation : obligations){
          List<AttributeAssignment> assignments = ((Obligation)
              obligation).getAssignments();
          for(AttributeAssignment assignment : assignments){
            System.out.println("Obligation : " + assignment.getContent() +"\n\n");
          }
        }
        //Initialize SparkSession
        SparkSession spark = SparkSession.builder()
                                         .appName("scala read from hdfs")
                                         .config("spark.master", "local")
                                         .getOrCreate();
        //Read request
        Dataset<Row> df = spark.read()
                               .option("header", "true")
                               .option("sep", ";")
                               .csv("hdfs://ec2-nameNode.eu-west-3.compute.
                               amazonaws.com:9000/sensitiveData/
                               annualCreditCardReport.csv");
        //Display Data
        df.filter("status = 'debit'")
          .groupBy("gender","age")
          .agg(count("*").as("numberOfDebtors"))
          .orderBy(desc("numberOfDebtors"))
          .show((int) df.count(), false);

      } else {
        System.out.println("\n" + userName + " is NOT authorized to " + operation + " the "
            + resource + " \n");
        List<Advice> advices = result.getAdvices();
        for(Advice advice : advices){
          List<AttributeAssignment> assignments = advice.getAssignments();
          for(AttributeAssignment assignment : assignments){
            System.out.println("Advice : " + assignment.getContent() +"\n\n");
          }
        }
      }
    } catch (ParsingException e) {
      e.printStackTrace();
    }
  }
} catch (Exception e) {
e.printStackTrace();
}
```

Fig. 10. Part of code to evaluate requests.

```
namespace OrganizationA{

    obligation notifyAdministrator = "https://organizationA.com/obligation/
                                      notifyAdministrator"
    advice notifyUser = "https://organizationA.com/advice/notifyUser"

    policyset globalAccessPolicy {

        apply denyOverrides

        hadoopServicesAccess
        hadoopObjectsAccess

        on permit {
            obligation notifyAdministrator{
                    Attributes.message = stringConcatenate("This user ",
                                          stringOneAndOnly(user.name),
                                        " has obtained access to this ",
                                          stringOneAndOnly(resource.identifier),
                                          " resource.")
            }
        }

        on deny {
            advice notifyUser{
                   Attributes.message = "You cannot access this resource.
                                         Contact your administrator for details."
            }
        }
    }
}
```

Fig. 11. Global XACML policy.

authorization Policies P1 and P2 are satisfied by this subject . Thus, Bob is allowed to access the Hadoop HDFS service and the read operation on the CreditCardReport object is also authorized to Bob.

Let's assume that another user, Anne, from the same department and project but with a different role (let us say *junior role with a weight of 0.3*) tries to perform a read action on the same object, CreditCardReport, during working hours. In this case, the rules defined in P1 are satisfied, which means that Anne will be allowed to access the Hadoop HDFS service, but the first rule defined in P2 will not be satisfied since the weight of the junior role is less than the sensitivity of the CreditCardReport object. Thus, according to the global authorization policy described above, Anne will not be authorized to perform the read operation on the report file.

VIII. Experiments and Results

To validate the proposed D2SAC framework, several experiments have been performed to evaluate the performance of this framework on the one hand and the overhead imposed by D2SAC on the other. Thus, in this section, we provide a detailed description of the Hadoop cluster configuration, the datasets used, and the analyses of the experiments.

A. Hadoop cluster setup

To evaluate the proposed D2SAC framework, we configured and deployed a Hadoop 3.3.3 cluster using the Amazon Web Services (AWS) Cloud platform. Our cluster consisted of five Elastic Cloud Computing (EC2) instances. We chose a high-performance xlarge general-purpose (t2.xlarge) instance with 4 CPUs and 16GB of memory for the NameNode, and for the DataNodes, we chose the large general-purpose (t2.large) instance type with 2 CPUs and 8GB of memory [35]. For all machines in our cluster, Linux (Ubuntu Server 22.04 LTS) was chosen as the Amazon Machine Image (AMI).

We also allocated an elastic public IP address for each instance and set up password-less SSH on instances to connect easily to these instances. Thus, to access Hadoop instances, we need to SSH with the public IP address and the created key pair (the .pem file). In this Hadoop cluster, we set the replication factor to 3 and the data block size to 128MB. Furthermore, to compare the overhead imposed by the D2SAC framework, we deployed another Hadoop cluster with the same hardware configuration (instances, AMI, replication factor, block size…) but without using the proposed framework.

B. Data used

To evaluate the correctness and overhead of the proposed framework, we used datasets generated from Kaggle [36]. Kaggle is an online platform for data scientists that allow users to find, collaborate and publish datasets. This platform aims to help professionals achieve their goals by solving data science challenges through powerful tools and resources. Kaggle offers the possibility to explore, analyze, and download quality data from several domains (education, retail, healthcare…) and supports various dataset formats (csv, json, zip, etc.). In this work, we used banking datasets generated from Kaggle, which contained sensitive information about customers of OrganizationA, such as credit card number, credit card type, IBAN, balance, etc. All of these datasets were divided into datasets of size 1GB to 10GB into

```
namespace OrganizationA{

    import Attributes.*
    import user.*
    import role.*

    advice notifyUserHS = "https://organizationA.com/advice/notifyUser"

    policy hadoopServicesAccess{

        target clause hadoopService =="HDFS"

        apply firstApplicable

        rule permitAccessOnlyToSenior{
            target clause currentTime<"07:00:00":time or currentTime>"19:00:00":time
            condition not(stringOneAndOnly(role.name) =="Senior")
            deny
            on deny {
               advice notifyUserHS{
                        Attributes.message = "You cannot access this service outside
                                              office hours.Contact your administrator
                                              for details."
               }
            }
        }

        rule permitAccessToAll{
            target clause role.name =="Senior" or role.name =="Junior"
            condition stringOneAndOnly(user.department) == "Finance" &&
                      stringOneAndOnly(user.address) == "Paris"
            permit

        }

         rule denyAccessOtherwise{
            deny
       }

    }
}
```

Fig. 12. Hadoop services access policy.

steps of 1GB.

To access these datasets and retrieve the requested information, we used the Spark 3.1.3 engine that was installed on the EC2 instances and developed three Spark jobs to run on these datasets. These Spark jobs were implemented in Java and executed in Hadoop using the Java API. The first Spark job (S1) filtered all the data corresponding to a customer with a debit credit card. It displayed their information (customerNumber, firstName, lastName, iban, balance) to know the debtor customers. The second Spark job (S2) counted the number of customers for each country and each credit card type and ranked them in descending order. The third Spark job (S3) filtred customers with debit cards and then grouped them by gender and age to count the total number of debtor customers.

C. Results

In this section, we analyze the overhead required by the D2SAC framework to access different datasets via Spark jobs. The overhead depicted here is the time our framework takes to display data on a standard output device.

Figure 15 compares the running time for Spark jobs on different datasets ranging from 1GB to 10GB. The keys S1, S2 and S3 stand for the three Spark jobs presented in the previous section. We can see that the processing time of Spark job (S2) is higher, closely followed by Spark job (S1) and finally Spark job (S3), which takes less time to run. This is because each Spark job contains different transformations and actions; some operations are more complex than others.

The Spark job S2 has to perform the groupBy() transformation, which is considered a wide transformation requiring shuffling and merging data to obtain the final result. Then, the S2 has to run the count() aggregate method on the grouped data to count the number of customers and rank them. Thus, the processing time of S2 is significantly affected due to shuffling data across different nodes resulting in taking more time to run.

The Spark job S1 has to display all records corresponding to customers with a debit credit card. In this job, the transformation functions select and filter are used to select the required columns from the rows satisfying the condition given in the filter function. In the Spark job S3, the filter function is firstly used to keep only debtor customers, then groupBy() wide transformation and the count() aggregate function are applied only to the filtered data, meaning that the amount of data has been reduced before performing the large transformation. Thus, the processing time of S3 is lower when compared to S1 and S2, as shown in Figure 15. The more expensive the requested operations in a Spark job are, the higher the running time to return the desired

```
namespace OrganizationA{

    import Attributes.*
    import user.*
    import role.*
    import resource

    policy hadoopObjectsAccess{

        target clause resource.classification =="Secret"

        apply denyUnlessPermit

        rule permitReadToAnyRole{
             target clause role.name =="Senior" or role.name =="Junior"
                           and action =="read"
             condition doubleOneAndOnly(role.weight) >= doubleOneAndOnly(resource.sensitivity) &&
                       stringOneAndOnly(user.project) == stringOneAndOnly(resource.project)    &&
                       timeInRange(timeOneAndOnly(currentTime), timeOneAndOnly
                        (timeBag("07:00:00":time)),timeOneAndOnly(timeBag("19:00:00":time)))
             permit
        }

        rule permitWriteOnlyToSenior{
             target clause role.name =="Senior" and action =="write"
             condition doubleOneAndOnly(role.weight) >= doubleOneAndOnly(resource.sensitivity) &&
                       stringOneAndOnly(role.securityLevel) =="high"                           &&
                       integerOneAndOnly(user.experience) >= 7                                 &&
                       ipAddressRegexpMatch(stringOneAndOnly(currentLocation),
                       ipAddressOneAndOnly(ipAddressBag("192.168.2.0":ipAddress)))
                permit
        }
    }
}
```

Fig. 13. Hadoop objects access policy.

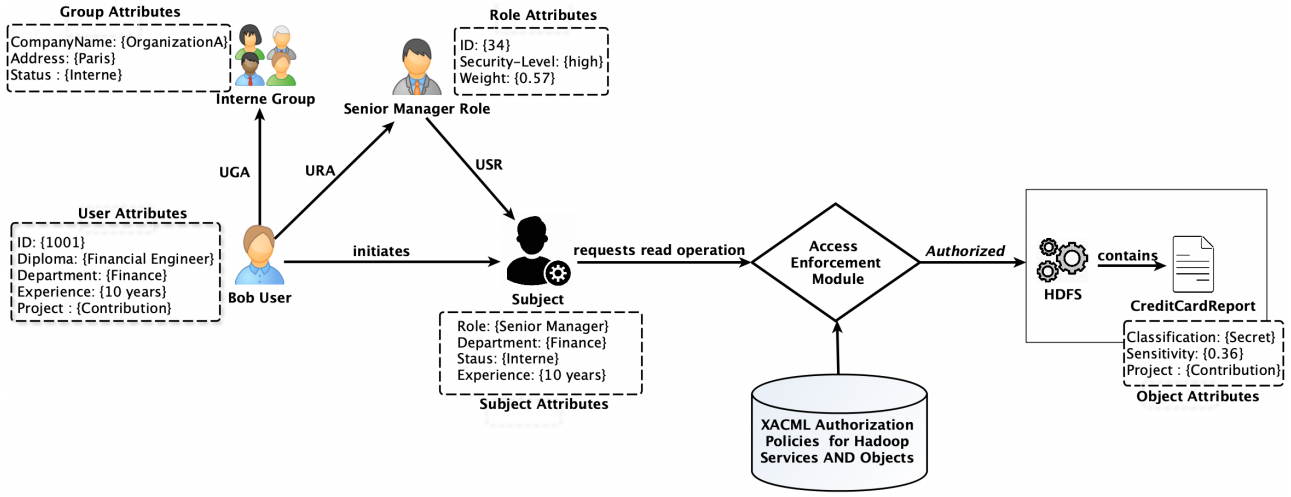

Fig. 14. H-RABAC model use of case.

result. Therefore, optimizing spark jobs through appropriate caching, optimal data format, and job monitoring to get the best performance and speed up the jobs is highly recommended.

Figures 16, 17 and 18 compare the running times of Spark jobs S1, S2, and S3 between the proposed D2SAC framework and a Hadoop implementation (naïve implementation) using the same datasets. The processing time of Spark jobs increases rapidly as the size of datasets increases. This is explained by the fact that all records in a dataset need to be processed to get results since they are structured data. Thus, the jobs running on a large dataset take longer to terminate either in our framework or in Hadoop.

On average, about 4.4% overhead is imposed by the D2SAC framework to access different datasets via these Spark jobs. This is a necessary trade-off to protect sensitive data from unauthorized access. We can justify this overhead by the time required for the D2SAC framework to control access to Hadoop services and objects. To this end, the D2SAC must retrieve the appropriate access control policies, evaluate the user requests and render an access decision. The demanded data is then displayed to the user if and only if this user is authorized to run its Spark job. Therefore, the overhead required by the D2SAC framework is not an

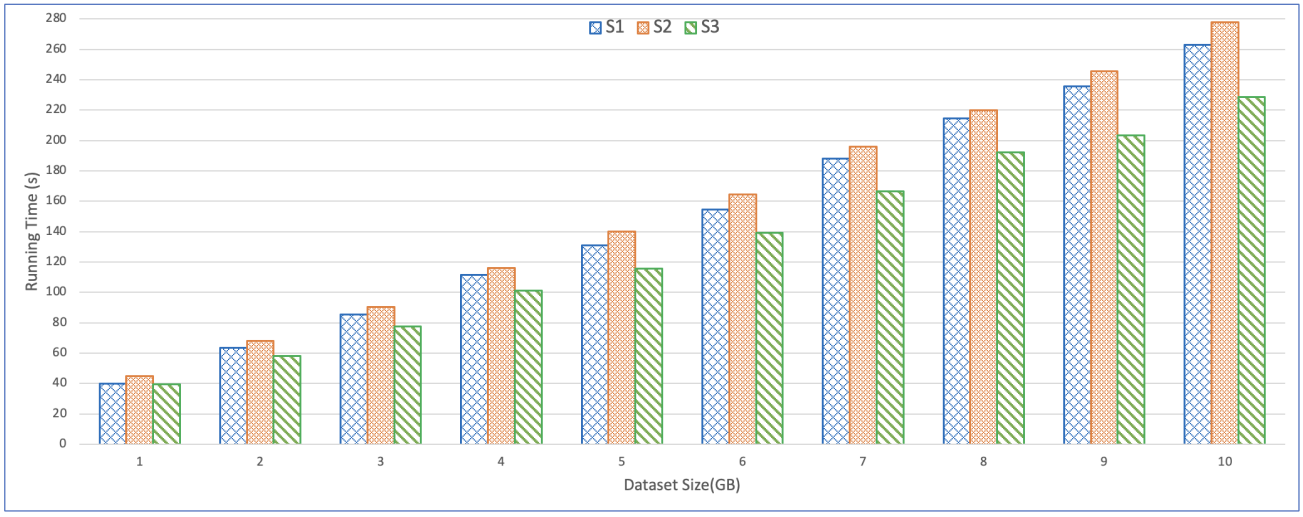

Fig. 15. Comparison of running time of Spark jobs on different datasets.

obstacle to its usability but a necessity to control access and protect sensitive data in Hadoop.

IX. Conclusion

There is no doubt that Big Data has brought significant benefits to enterprises, but the use of Big Data comes with commensurate risks. The need to protect data and secure sensitive information often arises. Access control is one of the essential requirements to limit unauthorized access in the Hadoop ecosystem. Therefore, in our previous papers, we proposed the D2SAC framework that provides end-to-end lifecycle protection by keeping sensitive data protected as long as it remains in the HDFS.

This paper proposes extensions to the D2SAC framework. It concentrates on the Access Enforcement Module responsible for providing dynamic access control. Thus, we propose a hybrid access control model called the H-RABAC model that inherits the benefits of both RBAC and ABAC models. The conceptual model of H-RABAC is presented, followed by formal definitions of each component. We additionally highlight extensions applied to XACML authorization policies by adding new elements to meet the Hadoop security requirements. By doing so, we first control access to Hadoop services and then to resources within Hadoop services.

For future work, we aim to improve the mathematical model used within the Sensitivity Estimator Module to recalculate the data sensitivity as the sensitivity value evolves, leading to changes in access control decisions. We also intend to use Cloud Computing to implement our framework to validate the efficiency and flexibility of D2SAC.

References

[1] V. Pendyala, "The Big Data Phenomenon," in Veracity of Big Data, Berkeley, CA: Apress, 2018, pp. 1–15.
[2] IDC, The premier global market intelligence company. https://www.idc.com/
[3] A. Gandomi and M. Haider, "Beyond the hype: Big Data concepts, methods, and analytics," Int. J. Inf. Manag., vol. 35, no. 2, pp. 137–144, Apr. 2015.
[4] M. Gupta, F. Patwa, and R. Sandhu, "An Attribute-Based Access Control Model for Secure Big Data Processing in Hadoop Ecosystem," in Proceedings of the Third ACM Workshop on Attribute-Based Access Control, Tempe AZ USA, Mar. 2018, pp. 13–24.
[5] G. S. Bhathal and A. Singh, "Big Data: Hadoop framework vulnerabilities, security issues and attacks," Array, vol. 1–2, p. 100002, Jan. 2019.
[6] N. Zanoon, A. Al-Haj, and S. M. Khwaldeh, "Cloud Computing and Big Data is there a relation between the two: a study," Int. J. Appl. Eng. Res., vol. 12, no. 17, pp. 6970–6982, 2017.
[7] Apache Sentry. https://sentry.apache.org/
[8] Apache Ranger. https://ranger.apache.org/
[9] Ait idar Hafsa, K. Aissaoui, H. Belhadaoui, and R. F. Hilali, "Dynamic Data Sensitivity Access Control in Hadoop Platform," in 2018 IEEE 5th International Congress on Information Science and Technology (CiSt), 2018, pp. 105–109.
[10] Ait idar Hafsa, H. Belhadaoui, and R. Filali, "A Conceptual Model for Dynamic Access Control in Hadoop Ecosystem," in Advances on Smart and Soft Computing, vol. 1188, F. Saeed, T. Al-Hadhrami, F. Mohammed, and E. Mohammed, Eds. Singapore: Springer Singapore, 2021, pp. 421–430.
[11] Ait idar Hafsa, H. Belhadaoui, and R. Filali, "A Mathematical Model to Calculate Data Sensitivity in Hadoop Platform Using the Analytic Hierarchy Process Method," IAENG Int. J. Comput. Sci., vol. 47, no. 4, pp. 765–774, 2020.
[12] E. Bertino and E. Ferrari, "Big Data Security and Privacy," in A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years, vol. 31, S. Flesca, S. Greco, E. Masciari, and D. Saccà, Eds. Cham: Springer International Publishing, 2018, pp. 425–439.
[13] NIST Big Data Public Working Group, "NIST Big Data Interoperability Framework: volume 4, security and privacy, version 2," National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 1500-4r1, Jun. 2018.
[14] D. S. Terzi, R. Terzi, and S. Sagiroglu, "A survey on security and privacy issues in Big Data," in 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST), 2015, pp. 202–207.
[15] B. Bashari Rad, N. Akbarzadeh, P. Ataei, and Y. Khakbiz, "Security and Privacy Challenges in Big Data Era," Int. J. Control Theory Appl., vol. 9, no. 43, pp. 437–448, 2016.
[16] H. Ye, X. Cheng, M. Yuan, L. Xu, J. Gao, and C. Cheng, "A survey of security and privacy in Big Data," in 2016 16th International Symposium on Communications and Information Technologies (ISCIT), 2016, pp. 268–272.
[17] Apache Hadoop. https://hadoop.apache.org/
[18] P. P. Sharma and C. P. Navdeti, "Securing Big Data Hadoop: a review of security issues, threats and solution," Int J Comput Sci Inf Technol, vol. 5, no. 2, pp. 2126–2131, 2014.
[19] B. Saraladevi, N. Pazhaniraja, P. V. Paul, M. S. S. Basha, and P. Dhavachelvan, "Big Data and Hadoop-a Study in Security Perspective," Procedia Comput. Sci., vol. 50, pp. 596–601, 2015.
[20] T. K. Ashwin Kumar, H. Liu, J. P. Thomas, and X. Hou, "Content sensitivity based access control framework for Hadoop," Digit. Commun. Netw., vol. 3, no. 4, pp. 213–225, Nov. 2017.
[21] M. Behera and A. Rasool, "Big Data Security Threats and Prevention Measures in Cloud and Hadoop," in Data Management, Analytics and Innovation, vol. 808, V. E. Balas, N. Sharma, and A. Chakrabarti, Eds. Singapore: Springer Singapore, 2019, pp. 143–156.

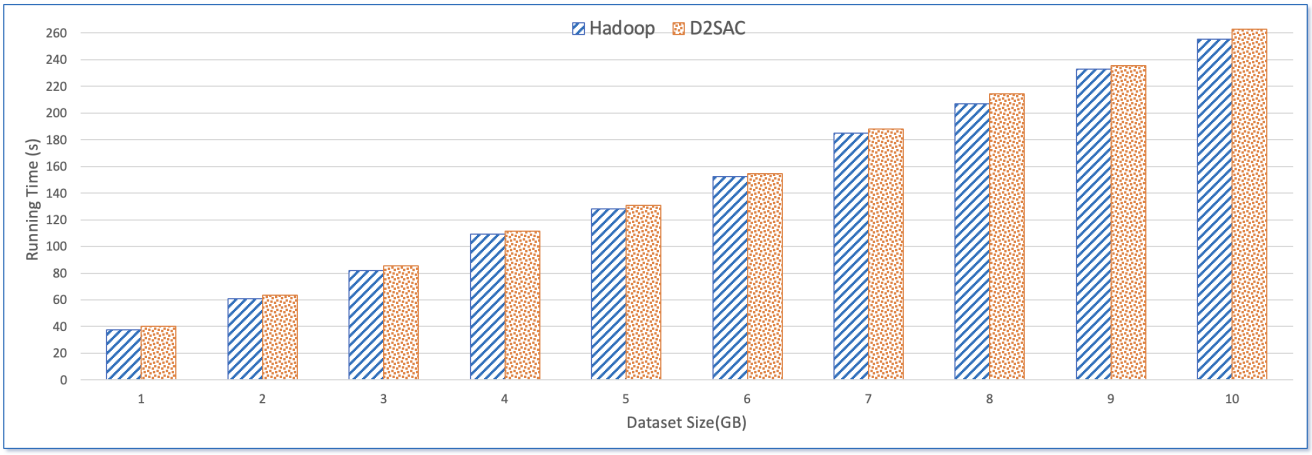

Fig. 16. Analysis of the overhead imposed by the D2SAC framework to access datasets via Spark job 1.

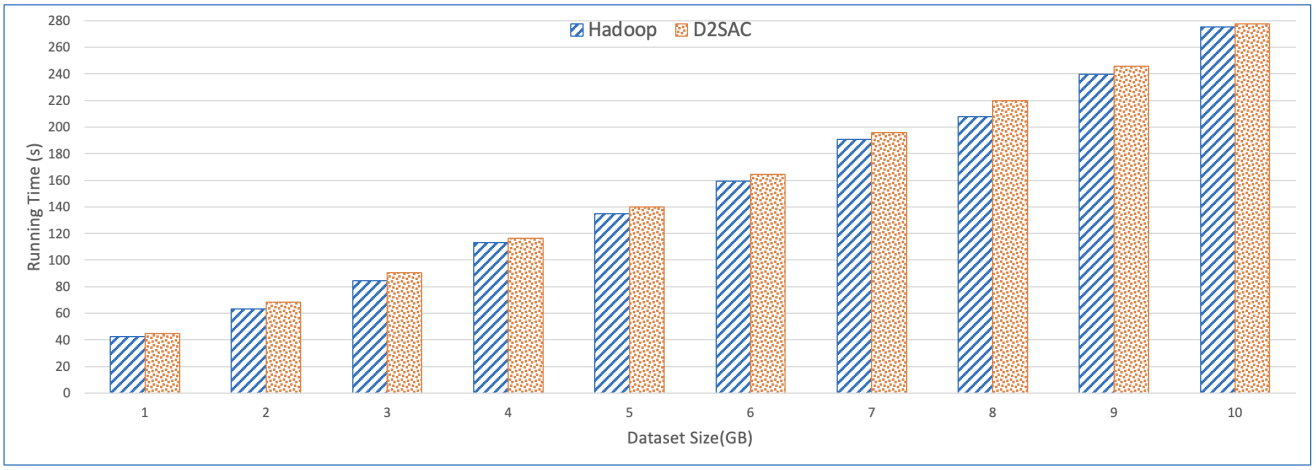

Fig. 17. Analysis of the overhead imposed by the D2SAC framework to access datasets via Spark job 2.

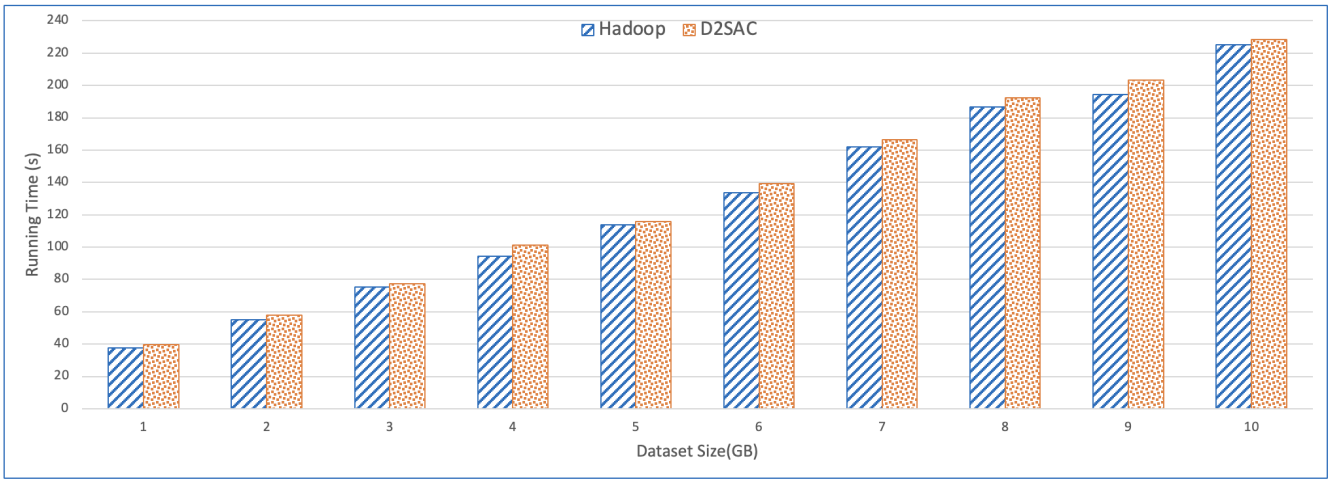

Fig. 18. Analysis of the overhead imposed by the D2SAC framework to access datasets via Spark job 3.

[22] M. Gupta, F. Patwa, and R. Sandhu, "Object-Tagged RBAC Model for the Hadoop Ecosystem," in Data and Applications Security and Privacy XXXI, vol. 10359, G. Livraga and S. Zhu, Eds. Cham: Springer International Publishing, 2017, pp. 63–81.
[23] G. Kapil, A. Agrawal, A. Attaallah, A. Algarni, R. Kumar, and R. A. Khan, "Attribute based honey encryption algorithm for securing Big Data: Hadoop distributed file system perspective," PeerJ Comput. Sci., vol. 6, p. e259, Feb. 2020.
[24] Y. A. Younis, K. Kifayat, and M. Merabti, "An access control model for cloud computing," J. Inf. Secur. Appl., vol. 19, no. 1, pp. 45–60, Feb. 2014.
[25] A. Cavoukian, M. Chibba, G. Williamson, and A. Ferguson, "The importance of ABAC: attribute-based access control to Big Data: privacy and context," Priv. Big Data Inst. Ryerson Univ. Tor. Can., 2015.
[26] D. R. Kuhn, E. J. Coyne, and T. R. Weil, "Adding attributes to role-based access control," Computer, vol. 43, no. 6, pp. 79–81, 2010.
[27] X. Jin, R. Sandhu, and R. Krishnan, "RABAC: role-centric attribute-based access control," in International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security, 2012, pp. 84–96.
[28] H. Ulusoy, M. Kantarcioglu, E. Pattuk, and K. Hamlen, "Vigiles: Fine-grained access control for mapreduce systems," in 2014 IEEE International Congress on Big Data, 2014, pp. 40–47.
[29] Y. B. Reddy, "Access control for sensitive data in hadoop distributed file systems," in Third International Conference on Advanced Communications and Computation, INFOCOMP, 2013, pp. 17–22.
[30] Q. Yuan, C. Ma, and J. Lin, "Fine-grained access control for Big Data based on CP-ABE in Cloud Computing," in International Conference of Young Computer Scientists, Engineers and Educators, 2015, pp. 344–352.
[31] T. L. Saaty, "Decision making with the analytic hierarchy process," Int. J. Serv. Sci., vol. 1, no. 1, pp. 83–98, 2008.
[32] eXtensible Access Control Markup Language (XACML) Version 3.0. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html
[33] Abbreviated Language for Authorization Version 1.0. 2015.
[34] Wso2 Balana, https://github.com/wso2/balana.
[35] AWS Instance Specifications, https://aws.amazon.com/fr/ec2/instance-types.
[36] Kaggle, https://www.kaggle.com.