An Attribute-Based Fine-Grained Access Control Mechanism for HBase

Liangqiang Huang, Yan Zhu(✉), Xin Wang, and Faisal Khurshid

Southwest Jiaotong University, Chengdu 611756, China
hwanglq@163.com, {yzhu,xinwang}@swjtu.edu.cn,
faisalnit@gmail.com

Abstract. In the current age of big data, the access control mechanism of HBase, a kind of NoSQL big data management system, needs to be improved, because there are some limitations of Role-Based Access Control (RBAC) in HBase. The coarse-grained access permissions produce little effect in many cases, and the elements used for authorization are not comprehensive enough. Attribute-Based Access Control (ABAC) is suitable for the authorization of NoSQL data storages due to its flexibility. However, it has not been investigated in HBase deeply. The objective of this paper is to study the data access control in HBase and to develop an ABAC-based mechanism for the security of HBase data. In light of the wide column feature of HBase, an Attribute-Based Fine-Grained Access Control mechanism (AGAC) is proposed, which covers two aspects, users' atomic operations and five granularity levels. When a user needs to access data in HBase storage, the AGAC will give the permission or deny by verifying user's atomic operations and by analyzing user's attributes according to the access control policies related to the data granularity level. This access control mechanism is verified on publically available email dataset and is proven to be effective to improve the access control capability of HBase.

Keywords: Column-oriented big data management system-HBase · Database access control · Attribute-Based · Fine-grained · NoSQL (Not Only SQL)

1 Introduction

Data is the treasure of any organizations and data security is absolutely vital. HBase is one kind of NoSQL (Not Only SQL) databases and a wide column big data storage based on Hadoop. It has outstanding capabilities for managing large data and is adopted by many enterprises and organizations, such as *Twitter*, *Facebook*, *Alibaba* etc. However, data access control in HBase is a big challenge for researchers. The *Kerberos* technology is used to verify the identity of user in HBase [1]. RBAC currently provides five coarse-grained permissions in HBase, i.e.; *Admin*, *Create*, *Write*, *Read*, and *Execute* [1]. For example, if Smith is authenticated via *Kerberos* and he has a right to *Write* data, then he can perform any one of *Write* operations such as *put*, *delete*, *append*, etc., because all sub-operations under *Write* are permitted to use. Data in HBase may be maliciously deleted or modified on an account that may cause some

S. Hartmann et al. (Eds.): DEXA 2019, LNCS 11706, pp. 44–59, 2019.
https://doi.org/10.1007/978-3-030-27615-7_4

serious consequences and will be very dangerous for an organization. In addition, RBAC is not applied to the scenarios where some attributes such as time or places that are also important authorization elements. Therefore, a more effective fine-grained access control mechanism should be developed for HBase in order to overcome the current problem.

In recent years, Attribute-Based Access Control (ABAC) [2] attracts the attention of many researchers due to highly customized ability on data protection [3]. Entities (subjects and objects) can be effectively distinguished by their characteristics, which are attributes of entities. In our study, the subject represents a user interacting with HBase, and an object denotes data at a specified granularity level of HBase. A mechanism based on ABAC is developed, which takes the attributes as the basic authorization elements, including entities, operations, and the environment data relevant to an access. Authorization rules based on these elements are defined in the access control policy. Multiple aspects of any entity are described by attributes, e.g., time, IP address, role, so that access control policies can be specified according to the actual situation, and an access to the object will be permitted or not by evaluating access control policies. Compared with RBAC, this ABAC-based mechanism can flexibly use attributes to verify whether an access operation should be authorized or not and overcomes the main limitations of RBAC. For example, an access control policy can be defined to only permit Smith's role to perform the fine-grained operation *put* over a specific time slot instead of a coarse one *write*.

In brief, this paper proposes an Attribute-Based Fine-Grained Access Control mechanism (AGAC) for HBase. The main contributions are as follows.

1. The proposed access control mechanism covers two fine-grained aspects, which are atomic operations and five granularity levels of HBase data.
2. A set of access control policies is defined and managed in HBase. Policies should be correlated with relevant objects in each permission verification by a binding method.
3. An access controller containing parser and verifier is developed for authorization verification, which is the key module of AGAC.

The structure of this paper is as follows. The related work is briefly introduced in Sect. 2. Section 3 discusses two aspects of access control of HBase. The principles of designing and parsing of access control policy are given in Sect. 4. Section 5 addresses the enforcement of our access control mechanism. The experimental verifications are discussed in Sect. 6. Finally, we conclude the AGAC mechanism and give future research directions in Sect. 7.

2 Related Work

Many recent studies have been carried out for the fine-grained access control mechanisms in NoSQL databases. In recent survey paper [4, 5], the authors discusses the differences of access control between big data platforms and traditional data management systems, and summarizes the research status of NoSQL databases access control. Fine grained access control within NoSQL databases is still in the very early stage [5].

In [3] Colombo et al. proposed a unified ABAC mechanism for NoSQL databases, but this access control mechanism is oriented to document storage and does not consider features of HBase comprehensively. The key point in [3] is about database query rewriting based on SQL++ technique, but HBase does not support SQL++ [6]. In [7], Kulkarni proposed a fine grained access control model for wide-column NoSQL databases, but it only consider a specified scenario. Longstaff et al. discussed an ABAC-based mechanism [8], which is supposed to be applicable to several NoSQL databases, but it may work only when these databases support a high-level SQL-like language.

In [9], Lai et al. proposed an access control mechanism with fine-grained authorizations to resolve the coarse-grained permissions problem in HBase. Their work improve the access control capabilities to some extent. However, there are still some shortcomings according to their experimental results. For example, if Mr. Smith has *SCAN* permission, he can not only perform *scan* operation but also execute *get* operation in HBase, which may lead to security risk. The studies in [10–12] only focused on document storage of NoSQL, which is different from column-oriented databases, such as HBase.

This paper develops an effective fine-grained mechanism for data access control in HBase, which integrates ABAC techniques with HBase features for improving security of big data management.

3 Aspects of Access Control in HBase

The access control mechanism of HBase must cover two aspects, one is data that will be accessed, and another is the operations which are used to manipulate data resources.

For example, Mr. Smith submits a query *q* to obtain data at column *cdata* of column family *cf* in table *t* under the namespace *default*. An example of HBase data is shown in Fig. 1.

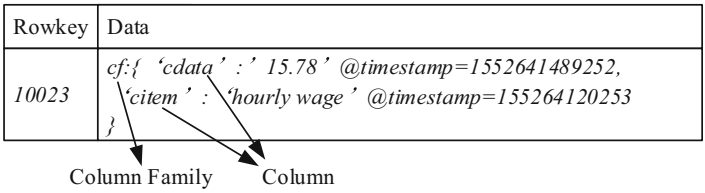


Fig. 1. An example of data in table *t* under the namespace *default.*

Mr. Smith uses *GET* (see Table 1) operation to fetch the data value *15.78* from *default.t.cf.cdata.* When Smith has submitted *q*, HBase should verify whether Smith has right to execute *GET* and whether he is permitted to access the data located by *default.t.cf.cdata*. To this end, the fine grained access control is needed.

3.1 Design of Granularity Levels of Access Control

Relational databases (e.g., Microsoft SQL Server) can authorize users to access data at the levels of database, table, and column, etc. MongoDB, a document storage of NoSQL, grants users to access data at the level of database, collection, document, and field. HBase is a column-oriented database, which is designed to accommodate semi-structured big data that could vary in file size, data types and columns. The data model in HBase consists of several logical components such as namespaces, tables, rows, column families, columns, and cells. Leaned from the access control mechanisms of RDBMS and MongoDB, a fine-grained access control mechanism for HBase should consider five levels of granularity at global database, namespace, table, column family, and column. These five granularity levels form a tree structure to demonstrate the path for addressing a data value, where the global database is the root node, columns are leaf nodes, and each node in the tree can inherit all access control policies from its parent node. For example in Fig. 2, the table *t* can inherit all access control policies from global database and namespace *default*.

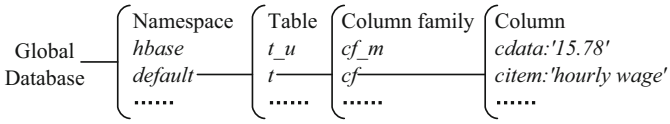


Fig. 2. The policy inheritance relationship between five granularity levels.

3.2 Design of Access Operation Control

RDBMS can authorize users data access operations by SQL, for example, *GRANT SELECT ON TABLE_1 TO SMITH*. HBase does not support SQL, but provides the basic way of Java API to manipulate data. By calling these basic Java APIs, the functionalities can be implemented similarly to those in terms of SQL. The Java APIs are classified in terms of data manipulation functions, which are grouped as 8 types and 46 kinds operations [13]. Some are shown in Table 1. The letter in a parenthesis represents the finest grained access level of operation. G, N, T, CF and C stand for global database, namespace, table, column family, and column respectively.

Table 1. Examples of operation types and operation sets.

Type	Set
DDL	*ALTER(CF), CREATE(N), DESCRIBE(T), LIST(T), DROP(T)*
DML	*APPEND(C), DELETE(C), GET(C), TRUNCATE(T), SCAN(C), PUT(C)*
Snapshot	*SNAPSHOT(T), RESTORE_SNAPSHOT(T), DELETE_SNAPSHOT(G)*

These operations are strong tools to manage and query big data, but may be misused. So there is a security risk posed to HBase. To this end, operations provided by HBase will be studied in our fine-grained access control mechanism in order to develop policies for managing them reasonable.

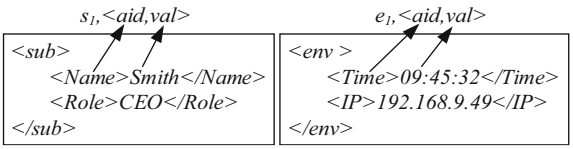
4 Access Control Policy and Its Parser

4.1 Definition of Policy

The key elements of AGAC are attributes of subjects, objects, operations and the environment relevant to a request. These elements constitute an access control policy. An access control policy is a XML file. We define the key concepts of policy as follows and several of them drew inspiration from [3].

Definition 1. (Subject): Subjects define users interacting with HBase. A subject S is a set of attributes which characterizes a user. A subject attribute s_i is a pair <*aid, val*>, where *aid* is the attribute identifier, and *val* represents the current value of s_i.

For example, if Mr. Smith's role is CEO, the subject of Smith is defined in Fig. 3(a).



(a) attributes S of Smith (b) attributes E of environment

Fig. 3. Example of attributes.

Definition 2. (Object): An object O denotes data at a specified granularity level. For example, data at the specified column family *cf*, which path is *default.t.cf*.

Definition 3. (Environment): An environment E is a set of attributes which characterize the access request context, which can be used to influence authorization. An environment attribute e_i is a pair <*aid, val*>, where *aid* is the attribute identifier, and *val* specifies the value of e_i.

For example, an environment E is defined by the time and IP address, then E can be defined in Fig. 3(b).

Definition 4. (Access Control Policy): An access control policy is a triple <*operations, sub, env*>, where *operations* is a set of permitted operations, *sub* specifies attributes of subject that must meet the conditions of the authorization, *env* specifies which attributes of environment must meet the conditions of the authorization

Two policy instances are given in Fig. 4.

Policy Instance 1

```
<policy>
  <operations>CREATE,DROP,SCAN,GET,PUT,
      DELETE</operations>
  <sub><Roles type="list">CEO</Roles></sub>
  <env>
    <IP type="ip" subnetMask="26">192.168.9.1</IP>
    <Time type="time">08:00:00-22:30:00</Time>
  </env>
</policy>
```

Policy Instance 2

```
<policy>
  <operations>CREATE,SCAN,PUT,DELETE</operations>
  <sub><Roles type="list">CEO;Director</Roles></sub>
  <env >
    <IP type="ip" subnetMask="24">192.168.9.1</IP>
    <Time type="time">09:00:00-20:00:00</Time>
  </env>
</policy>
```

Fig. 4. Example of policy instances.

For example, if *Policy Instance 1* is bound with the object *default.t.cf.cdata*, that means all operations on object *default.t.cf.cdata* are controlled by *Policy Instance 1*.

If Mr. Smith's role is CEO, he uses *GET* to fetch a data from *default.t.cf.cdata* at time *14:20:23*, and the IP address of Smith's device is *192.168.9.49*, his data access will be permitted. This is because CEO is in the authorized list of *Policy Instance 1*, IP address belongs to the authorized IP address segment (*192.168.9.1-192.168.9.62*) which is calculated by subnet mask and IP start address, and the access time is in the authorized time slot. Therefore, Smith is permitted to access data located by *default.t.cf.cdata*.

On the contrary, if Smith's role is CTO and/or the access time is *07:58:00*, then Smith is rejected to access data located by *default.t.cf.cdata*.

4.2 Binding Policies with Objects

A policy is bound with an object to determine to which object the policy applies. A policy can be bound with multiple objects and multiple policies can be bound with one object. An object is identified by the row key, and a policy instance is specified by the policy serial number. Connecting the row key and the policy serial number will bind them together. Such a binding information will be managed in a HBase table, AGACL (Attribute-Based Fine-Grained Access Control List).

AGACL is created under the namespace *hbase* for managing policies and policies binding information. The detailed binding method is described as follows.

Row Key of AGACL. The row key of AGACL is designed by using the feature of row keys in HBase, which means the HBase data is stored uniquely in a table according to the sort order of dictionary of row keys. They can be designed as:

$$<\textit{namespace name}\,[:\textit{table name}\,[:\textit{column family name}\,[:\textit{column name}]]]>$$

The value of row key denotes the path of the objects to which the policies are applied. For example, the value of row key *"default:t:cf:cdata"* means the policies stored in this row of AGACL table, which will be used to control the access of the data located at the cell addressed with *default.t.cf.cdata*. The special value *'hbase:global'* specifies the policies are applied on the global database.

Column Family of AGACL. As the column can only be attached to the column family, the column family *'abcf'* (Attribute-Based Column Family) is created in AGACL.

Column of AGACL. Multiple policies can be applied to an object, so that a serial number is designed to identify each policy. The column can be designed as:

$$<\textit{policy}:\ \textit{serial number}>$$

Suppose we want to bind *Policy Instance 1* and *Policy Instance 2* with column *cdata* in column family *cf* in table *t* under the namespace *default*, the format of policy in AGACL is shown in Fig. 5. The parts marked with black box should be replaced with actual content of the XML files (see Fig. 4), which is simplified here to reduce the space.

Rowkey	Data
default:t:cf:cdata	*abcf:{ 'policy:1' : 'Policy Instance 1' @timestamp=1552641489256, 'policy:2' : 'Policy Instance 2' @timestamp=1552641901941 }*

Fig. 5. Format of policy bound in AGACL.

4.3 Policy Processing

From the operational point of view, binding or unbinding means to write a policy into or delete a policy from AGACL according to the path of object. In our access control mechanism, we use PUT operation to write a policy into AGACL, and DELETE to take the policy away from AGACL. The *Superuser* in HBase has all permissions and can initialize the access control mechanism of HBase.

We developed a parser using *dom4j* [14] (a flexible XML framework for Java). The parser is used to analyze a policy in XML to produce an authorization condition set. The parsing procedure is as follows.

Step 1: Load the XML file to the parser.
Step 2: Get the list of operations according to the label *operations*.
Step 3: Get subject attributes, including the list of roles.
Step 4: Get environment attributes, including the IP address segment calculation by subnet mask and IP start address, and the time slot.

5 Enforcement of AGAC

5.1 The Framework of AGAC

HBase provides a framework-*Coprocessor* [1] that allows developers to write their own program in the server to implement fine-grained access control. In this study, we develop an Attribute-Based Fine-Grained Access Controller (AGACer) based on *Coprocessor*, which is integrated into HBase. AGACer can perform policy parsing and access authorization verification based on users' attributes and policies, when users submit a data access request. The framework of our AGAC mechanism is shown in Fig. 6.

The workflow of our framework consists of seven main steps. ① A user submit a data access request. ② Send the request and the attributes to verifier. ③ Search and find the policies based on the path of the required object. ④ Send the relevant policies to the parser. ⑤ Send the key attributes of the parsed policies to verifier. ⑥ If the verification is successful, the request is permitted to execute, else the permission is denied, exception is returned. ⑦ Return the result of the request under the successful permission.

AGACer contains parser and verifier. The parser is used to analyze policies in XML to produce the authorization conditions (ref. Sect. 4.3). The verifier evaluates whether the operation in the request and user's attributes meet the authorization conditions.

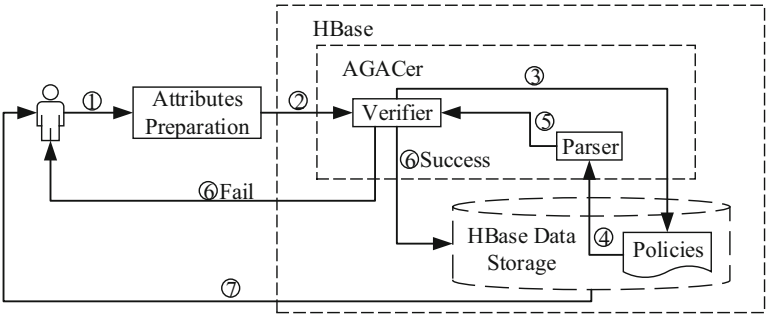


Fig. 6. The framework of AGAC.

5.2 Preparation of Users' Attribute Set

The subject is users' static attributes, such as CEO, Project Leader, and Department Manager. In our work, subject is prepared and managed through information transformation from HR department or from the e-forms filled by users when they register in HBase system.

The environment parameters are users' dynamic attributes, such as IP address, request time, and place. Environment attributes are prepared when a user submits an access request to HBase data, which is accomplished automatically by our mechanism.

5.3 Process of Access Control

We use an example to show the procedure of access control. Let's assume that *Policy Instance 1* (Fig. 4) is already bound with the Namespace *default* by *Superuser*. Mr. Smith submitted a database request Q which uses *CREATE* operation to create a table t at namespace *default*, the process of authorization verification is addressed in Fig. 7.

Attributes Preparation. An example of the attributes of Mr. Smith and the environment attributes relevant to Q are shown in Fig. 8.

Policy Acquiring. The access object of Q is the namespace *default*. Because of the policy inheritance, all policies from its parents, global database, should be firstly applied to control the object access. However, there are no policies on the global database, therefore we can use all policies from the namespace *default*, where the data object locates.

Policy Evaluation. Based on the policies acquired at the previous step, the attribute values of User Smith and *CREATE* are evaluated as whether they meet the authorization conditions. For example, the IP address of Smith should belong to the authorized IP address segment. If the attributes and operation together conform to one

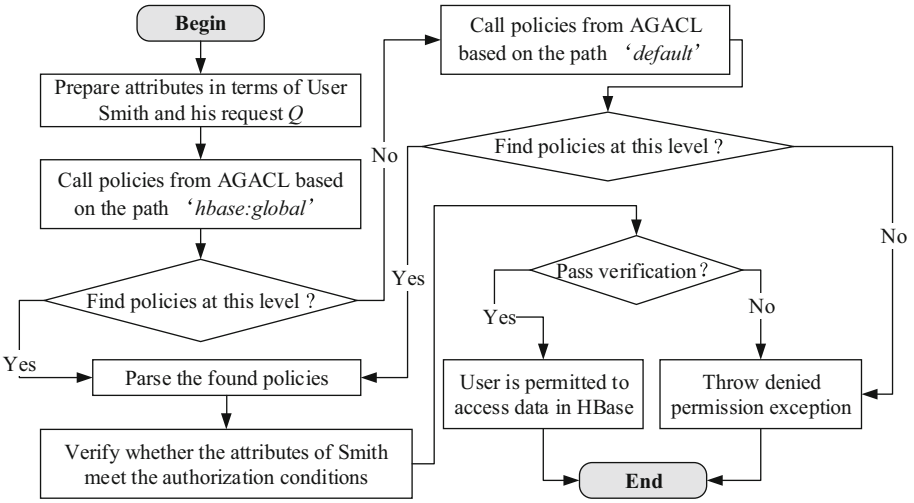


Fig. 7. The verification process of authorization.

Attributes of Smith:

```
<attributes>
   <sub><Role>CEO</Role></sub>
   <env><IP >192.168.9.49</IP><Time>10:30:21</Time></env>
</attributes>
```

Fig. 8. Attributes of Mr. Smith

Table 2. Policy evaluation algorithm.

Input: *policies* obtained from previous step; an operation *op*; *S* denotes user attributes; *E* denotes environment attributes. **Output:** *true* denotes the access is successfully permitted, *false* denotes the fail case.

Begin
For each *policy* in *policies* do
If *policy* does not contain *op*, then go to next 'For' loop.
If *S* does not meet the authorization requirements of *policy*, then go to next 'For' loop.
Else if *E* meet the authorization conditions of *policy*, then return *true*.
End For
Return *false*;
End

of the policies, Mr. Smith is permitted to manipulate access object using *CREATE*. The policy evaluation algorithm is described in Table 2.

In the example, Mr. Smith passed the access authorization verification and can execute Q, because both attributes and access operation conform to the authorization conditions of *Policy Instance 1*.

6 Experimental Verification and Analysis

We have designed two experiments to verify the effectiveness and policy inheritance. The access control mechanism is effective, if AGAC can control the access operations over HBase data correctly. In policy inheritance evaluation, parents' policies should be inherited by children for managing data access. However, policies cannot be inherited between peers or from children to parents.

To the best of our knowledge, the ABAC-based mechanism in HBase has not been applied. It is the first study to suggest this solution, which is verified by applying different test cases to confirm effectiveness of our proposed AGAC mechanism.

The experiments are carried out on a cluster built by three PCs. The cluster is based on the master-slave architecture with one master node and two slave nodes. The environment configuration is as follows, *Ubuntu 16.04 LTS, HBase 1.2.6, Hadoop 2.7.6, Zookeeper 3.10.4, JDK 1.8*.

6.1 HBase Data Storage Construction and Experiment Preparation

The dataset used in this study is *Enron Corpus* [15] publically available, which contains emails and related information from a company employees. The dataset contains the subject, recipients, body and sending date of email, etc. The dataset is preprocessed and imported into HBase. Accessing data will be managed with 5 granularity levels from global database, namespace, table, and column family to column. The schema is shown in Table 3.

Table 3. Schema of enron dataset in HBase.

Table	Rowkey	Column family
enronEmail	*sender:messageId*	*message :{body, subject, recipientList, sendDate ...}*
enronUser	*emailId*	*info :{firstName, lastName, status ...}*

It is supposed that two users (U_1 and U_2) want to manipulate the Enron email data in HBase using a set of operations. Those operations with high usage frequency are selected in the assessment, which are listed in Table 4.

Table 4. Requests for testing.

Request	Content
Req1	Create a specified table *testTable1* under the namespace *default* using *CREATE* operation
Req2	Use *DROP* to drop the table *testTable2* under the namespace *default*
Req3	Use *GET* to query a specified email record sent by *lisa.gang@enron.com*
Req4	Use *SCAN* to query from email address *marie.heard@enron.com* the historic email records
Req5	Add the body of an email record using *PUT*
Req6	Delete the body of an email record using *DELETE*

The attributes of U_1 and U_2 are prepared (shown in Fig. 9). The test time is from 1 p.m. to 3 p.m (obtained from server in real time). Two access control policy instances from Sect. 4.1 are used in the experiments.

Attributes of U1

```
<attributes>
   <sub><Role>CEO</Role></sub>
   <env><IP>192.168.9.23</IP></env>
</attributes>
```

Attributes of U2

```
<attributes>
   <sub><Role>Director</Role></sub>
   <env><IP>192.168.9.81</IP></env>
</attributes>
```

Fig. 9. The attributes of U_1 and U_2.

6.2 Experiments and Analysis

The access control of five granularity levels will be evaluated on the hierarchy from top downwards.

Case 1. Binding *Policy Instance 1* with the global database. U_1 and U_2 execute requests in Table 4, respectively. If users' requests are permitted to execute, HBase return the execute results. For example, Fig. 10 shows the result of *Req3* executed by U_1.

```
body: According to the scheduling sheet, deal#380414 is 400mw total.  This is a 50 mw deal.  25mw was booked
recipientList: TO:jennifer.thome@enron.com
reference: From:        Thome, Jennifer  Sent:  Monday, June 24, 2002 2:05 PMTo:        Gang, LisaSubject:
sendDate: 2002/6/24 15:10:37
subject: RE: BPA - Dec. 2001
```

Fig. 10. The result of *Req3* executed by U_1.

If users' requests are rejected, HBase throw the permission denied exceptions. Figure 11 shows the rejection information of *Req1* executed by U_2.

```
org.apache.hadoop.hbase.security.AccessDeniedException: Insufficient permissions:[operate=CREATE;namespace:default]
        at org.apache.hbase.security.AGAC.AGACer.preCreateTable(AGACer.java:143)
        at org.apache.hadoop.hbase.master.MasterCoprocessorHost$11.call(MasterCoprocessorHost.java:216)
        at org.apache.hadoop.hbase.master.MasterCoprocessorHost.execOperation(MasterCoprocessorHost.java:1146)
        at org.apache.hadoop.hbase.master.MasterCoprocessorHost.preCreateTable(MasterCoprocessorHost.java:212)
```

Fig. 11. Rejection information of *Req1* executed by U_2.

All access control results of Case 1 are shown in Table 5, where *Pass* means access is permitted, and *Reject* means access is not allowed.

Table 5. Results of Case 1.

	Req1	*Req2*	*Req3*	*Req4*	*Req5*	*Req6*
Results of U_1	*Pass*	*Pass*	*Pass*	*Pass*	*Pass*	*Pass*
Results of U_2	*Reject*	*Reject*	*Reject*	*Reject*	*Reject*	*Reject*

The results show that the proposed access control mechanism is effective and can make correct authorization verification. Since *Director* is not in the authorized list of *Policy Instance 1*, the IP address of U_2 does not belong to the authorized IP address segment, all requests of U_2 are rejected.

The result of *Req1* executed by U_1 shows that the policy inheritance is successful, because *Policy Instance 1* is bound with the global database, the access permission of namespace *default* is inherited from that of root level (global database), although *Req1* is to access data in the namespace *default*.

Case 2. Unbinding *Policy Instance 1* from global database firstly, then binding *Policy Instance 2* with the namespace *default*. U_1 and U_2 execute requests in Table 4, respectively. The access control results are shown in Table 6. Since neither *DROP* nor *GET* is an operation permitted by *Policy Instance 2*, *Req2* and *Req3* issued by both users are rejected. It shows that the proposed access control mechanism can control fine-grained operations.

Table 6. Results of Case 2.

	Req1	*Req2*	*Req3*	*Req4*	*Req5*	*Req6*
Results of U_1	*Pass*	*Reject*	*Reject*	*Pass*	*Pass*	*Pass*
Results of U_2	*Pass*	*Reject*	*Reject*	*Pass*	*Pass*	*Pass*

Case 3. Unbinding *Policy Instance 2* from namespace *default*, then binding *Policy Instance 1* and *Policy Instance 2* with table *enronEmail*. U_1 and U_2 execute each request in Table 4, respectively.

The access control results are shown in Table 7. Because the execution level of *CREATE* is the namespace level, *Req1* is rejected, although *CREATE* is permitted on the table level. This proves that high levels cannot inherit low-level policies. The

execution level of *DROP* is at the table level, but there are no policies bound with *testTable2* and policies cannot be inherited between peers, so that *Req2* is rejected, because *Req2* should be permitted or not to execute could not be judged.

Table 7. Results of Case 3.

	Req1	*Req2*	*Req3*	*Req4*	*Req5*	*Req6*
Results of U_1	*Reject*	*Reject*	*Pass*	*Pass*	*Pass*	*Pass*
Results of U_2	*Reject*	*Reject*	*Reject*	*Pass*	*Pass*	*Pass*

Case 4. Unbinding *Policy Instance 1* and *Policy Instance 2* from table *enronEmail*. Binding *Policy Instance 1* with column family *message*. U_1 and U_2 execute each request of Table 4, respectively. The authorization verification is as the same as the cases above. The access control results are shown in Table 8.

Table 8. Results of Case 4.

	Req1	*Req2*	*Req3*	*Req4*	*Req5*	*Req6*
Results of U_1	*Reject*	*Reject*	*Pass*	*Pass*	*Pass*	*Pass*
Results of U_2	*Reject*	*Reject*	*Reject*	*Reject*	*Reject*	*Reject*

Case 5. Unbinding *Policy Instance 1* from column family *message*. Binding *Policy Instance 1* with column *body*. U_1 and U_2 execute the requests given in Table 4, respectively. The authorization verification is as the same as the cases above. The access control results are shown in Table 9.

Table 9. Results of Case 5.

	Req1	*Req2*	*Req3*	*Req4*	*Req5*	*Req6*
Results of U_1	*Reject*	*Reject*	*Reject*	*Reject*	*Pass*	*Pass*
Results of U_2	*Reject*	*Reject*	*Reject*	*Reject*	*Reject*	*Reject*

AGAC mechanism proposed in this paper is proven to be effective by the experiments. It can safeguard HBase data and system to a very large extent by verifying who can use which operations to manage what kind of data.

6.3 Time Overhead Analysis

We use *executeTime* in milliseconds to record the process time of AGACer.

$$execute\,Time = endTime - startTime \tag{1}$$

Time overhead for verifying authorization is affected by many factors; such as network bandwidth of the cluster and machine performance. The time cost of our access control mechanism has two parts. One is the time for accessing AGACL to acquire the policy set for authorization. Another is the time for verifying authorization.

For example, the time overhead of Case 1 is shown in Fig. 12. The time overhead for U_1's successful authorization verification is significantly less than the time overhead for rejecting U_2. This is because U_1 passes authorization verification at the global level, which is the coarsest grained level and did not need much time to obtain the policies, while verifying authorization for U_2 needs more time to search and obtain the polices along the fine-grained path.

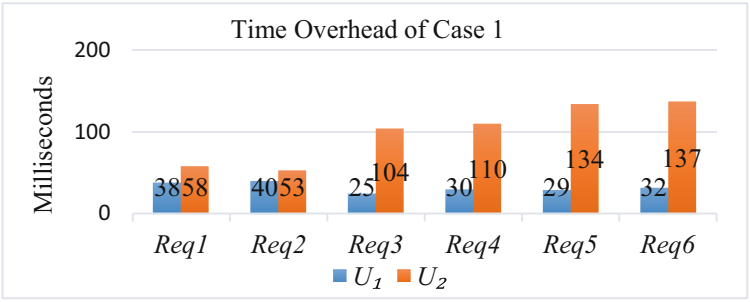


Fig. 12. The time overhead of Case 1.

In addition, the number of policies used for each request is also an important factor affecting time overhead, which cannot be ignored.

7 Conclusions and Future Work

Based on the in-depth study of HBase, we proposed a novel access control mechanism AGAC for improving access control capability of HBase.

AGAC mechanism consists of three key parts, i.e., defining access control policies and its parser, storing and managing the policies, and developing an access controller. AGAC covers two fine-grained aspects, which are atomic operations and five data granularity levels, therefore HBase data access is controlled carefully and fine-grained. Our experimental results prove that AGAC improves the access control capability and supports flexibility authorization of HBase. Besides, the proposed AGAC mechanism can be implemented quickly in HBase.

In the future, we plan to improve and extend AGAC mechanism based on following aspects.

1. Policy conflict checking and a default policy will be investigated.
2. The technique for extracting the complete entity attributes and the approach for automatically correlating attributes and permissions should be studied in depth.
3. A better model than XML should be investigated for policy description and quickly parsing, in order to improve the performance.
4. NoSQL databases are in varied forms [16], such as key-value storage, graph databases. To improve the access control ability of these NoSQL databases is still a big challenge. Modifying and integrating AGAC mechanism in graph storages of NoSQL will be our next focus.

Acknowledgement. This work is supported by the Sichuan Science and Technology Program (No 2019YFSY0032).

References

1. Apache HBase™ Reference Guide. http://hbase.apache.org/book.html. Accessed 20 Feb 2019
2. Hu, V.C., Kuhn, D.R., Ferraiolo, D.F.: Attribute-based access control. Computer **48**(2), 85–88 (2015)
3. Colombo, P., Ferrari, E.: Towards a unifying attribute based access control approach for NoSQL datastores. In: Proceedings of the IEEE 33rd International Conference on Data Engineering, pp. 709–720. IEEE Computer Society, San Diego (2017)
4. Colombo, P., Ferrari, E.: Access control technologies for Big Data management systems: literature review and future trends. Cybersecurity **2**(1), 3 (2019)
5. Colombo, P., Ferrari, E.: Access control in the era of big data: state of the art and research directions. In: Proceedings of the 23rd ACM Symposium on Access Control Models and Technologies, pp. 185–192. ACM, Indianapolis (2018)
6. Ong, K.W., Papakonstantinou, Y., Vernoux, R.: The SQL++ unifying semi-structured query language, and an expressiveness benchmark of SQL-on-Hadoop, NoSQL and NewSQL databases. Comput. Sci. CoRR, abs/1405.3631 (2014)
7. Kulkarni, D.: A fine-grained access control model for key-value systems. In: Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy, pp. 161–163. ACM, San Antonio (2013)
8. Longstaff, J., Noble, J.: Attribute based access control for big data applications by query modification. In: Proceedings of the IEEE Second International Conference on Big Data Computing Service and Applications, pp. 58–65, IEEE, Oxford (2016)
9. Lai, Y.Y., Qian, Q.: HBase fine grained access control with extended permissions and inheritable roles. In: Proceedings of the 2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pp. 181–185, IEEE, Takamatsu (2015)
10. Colombo, P., Ferrari, E.: Towards virtual private NoSQL datastores. In: Proceedings of the 32nd IEEE International Conference on Data Engineering, pp. 193–204, IEEE, Helsinki (2016)

11. Colombo, P., Ferrari, E.: Enhancing MongoDB with purpose based access control. IEEE Trans. Dependable Secure Comput. **14**(6), 591–604 (2017)
12. Colombo, P., Ferrari, E.: Fine-grained access control within NoSQL document-oriented datastores. Data Sci. Eng. **1**(3), 127–138 (2016)
13. Huang, L.Q., Zhu, Y., Tao, X.: Research on fine-grained access control method based on HBase. Appl. Res. Comput. (2019). https://doi.org/10.19734/j.issn.1001-3695.2018.08.0648. (In Chinese)
14. Flexible XML framework for Java. https://dom4j.github.io/. Accessed 20 Feb 2019
15. Klimt, B., Yang, Y.: The enron corpus: a new dataset for email classification research. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) ECML 2004. LNCS (LNAI), vol. 3201, pp. 217–226. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30115-8_22
16. DB-Engines Ranking. https://db-engines.com/en/ranking. Accessed 20 Feb 2019