TEST Engineering & Management

*July – August 2020*
*ISSN: 0193-4120 Page No. 3119 - 3127*

# A Model for Agricultural Ontology Storage and Access Control Using HBase

Chao Liu[1], QingDong[2],Shaowen Li[1*]
[1]School of Information and Computer Science, Anhui Agricultural University, Hefei, Anhui, China.
[2] Anhui Academy of Agricultural Sciences, Hefei, Anhui, China

***Article Info***
***Volume 83***
***Page Number: 3119 - 3127***
***Publication Issue:***
***July - August 2020***

***Article History***
***Article Received****: 06 June 2020*
***Revised****: 29 June 2020*
***Accepted****: 14 July 2020*
***Publication****: 25 July 2020*

***Abstract***

As ontologies incorporate increasing amounts of knowledge, and knowledge that spans multiple fields, there are increasing concerns relating to data privacy. This paper focusses on the field of agricultural knowledge, introducing a role-based model for access control and proposing a scalable HBase model for the storing agricultural ontologies. Access control is achieved by adding role identifiers to the row key. A series of query algorithms are presented and the model is tested by running a series of experiments using different queries and data sets of different sizes.

***Keywords:****Agricultural Ontology; Ontology Storage; Access Control; HBase; RBAC*

## I. Introduction

With the rapid development of knowledge sharing, information integration, and web services, ontology hasattracted much attention and has been a focus of research since the 1990s. Ontology concerns the problem of knowledge representation, organization, and sharing through the formal description of concepts, terms, and their relationships. Current research in ontology is mainly domain ontology, focusing the representation of knowledge within a particular domain—most ontologies are aimed at a specific domain or at the segmentation of domains. Agricultural knowledge is a highly productive knowledge class that can increase the efficiency of the agricultural labor force and therefore production capacity. For example, ontologies have been constructed for fisheries, foodsafety, nutrition, agriculture[1], prototype biosecurity[2], potatoes[3], tea pests.[4] With the successful application of cloud computing, big data, and the internet of things in agriculture, volumes of agricultural data are growing geometrically. Extreme volumes of such data provide a rich corpus of data for modeling agricultural ontologies but they also cause difficulties. The primary issue is with storing and accessing such large volumes of ontological knowledge. There are two main ways to store ontological data:in filesor in a relational database. When stored in files, ontologies can be directly expressed in an ontology representation language, which allows complete preservation of the semantics. As the ontology service develops, these

July – August 2020
ISSN: 0193-4120 Page No. 3119 - 3127

files need to loaded into memory frequently and this can adversely affect the efficiencyof the service. Storage in a relational database can be achieved using tools such as Jena. Once stored in a relational database, data persistence helps to make ontological operations efficient. However, some semantics are lost when ontologies are imported into relational databases because of flaws in the structure itself.

HBase is a non-relational database, which allows data to be stored in a flexible way that can preserve complicated ontological semantics. HBase is based on the Hadoop Distributed File System (HDFS), which is a distributed MapReduce framework that supports efficient service of large volumes of ontological knowledge. In current systems for ontology storage in HBase, row keys are set to differentcombinationsof triple elements and ontologies are mostly decomposed into RDF triples before storage. Abraham constructed three HBase tables with subject, predicate, and object as the row keys for each table, and other elements were stored as row values.[5] Sun constructed six HBase tables, including three tables that were identical with those of Abraham and three additional tables that had the combinations predicate-subject, subject-object, and predicate-object as row keys.[6] Papailiou also used three tables, storing RDF triples under the row keys subject-predicate, predicate-object and object-subject.[7] Punnoose constructed three tables with RDF triples as row keys, though in a different order in each table.[8] The two major considerations in the design of each of these schemes were storage consumption and query efficiency.HBase provides a solid foundation for efficiently serving massive quantities of ontological knowledge. But storage and efficiency are not the only relevant concerns—as ontologies represent more knowledge between and across more fields, there are increasing concerns about data privacy and security. Level-to-level management is increasingly important, as is fine-grained access control. This work focuses on the large-scale storage of agricultural knowledge using HBase, with an emphasis on both efficient storage and secure role-based access control.

## II. Storage model

### 1. HBase

HBaseis part of the Hadoop project. It is a column-orientated store for unstructured data. Data is stored in rows composed of row key, time stamp, column family, column name, and column value.[9] Row key is the primary key in HBase. It is sorted lexicographically, and tables are searched according to the order of the row key. Time stamp is the time when the data was written and is used to distinguish different versions of the same column data. A column family is a group of columns with similar features and can have arbitrary size. The column name, prefixed by the column family, gives the column position. Column value is a string, and is stored in a cell whose position is determined by the row key, time stamp, column family, and column name. The logical structure of HBase is shown in Table 1.

July – August 2020
ISSN: 0193-4120 Page No. 3119 - 3127

**Table 1.** HBase logical structure

| Row key | Time stamp | Column family: $f_1$ | | | | Column family: $f_2$ | |
|---|---|---|---|---|---|---|---|
| | | Column name | Column value | Column name | Column value | Column name | Column value |
| $k_1$ | $t_1$ | $f_1:n_{11}$ | $v_{1\text{-}11}$ | | | | |
| | $t_2$ | | | $f_1:n_{22}$ | $v_{1\text{-}22}$ | | |
| $k_2$ | $t_3$ | $f_1:n_{31}$ | $v_{1\text{-}31}$ | | | $f_2:n_{31}$ | $v_{1\text{-}31}$ |
| | $t_4$ | | | $f_1:n_{42}$ | $v_{1\text{-}42}$ | | |
| | $t_5$ | | | $f_1:n_{52}$ | $v_{1\text{-}52}$ | $f_2:n_{31}$ | $v_{1\text{-}51}$ |

## 2. RBAC

Role-based access control (RBAC) is a flexible model for access control that uses roles to match users with appropriate permissions. Each role is defined by a set of requirements and is granted certain access permissions. The user acquires permissions though being assigned roles. This model makes the management of user access permissions more convenient.[10]

Most RBAC systems are currently based on RBAC96, which is built around the concepts of *user*, *role*, *session*, and *access permission*. The relationship between roles and users is many-to-many, as is the relationship between roles and access permissions. The RBAC model is defined as follows[11]:

(1) *U* is a set of *users*, *R* is a set of *roles*, *S* is a set of *sessions*, and P is a set of *access permissions*;

(2) *RH* is the role hierarchy;

(3) *PA* is the access permission assignment;

(4) *UA* is the user assignment;

(5) *users*: $S \rightarrow U$ is the mapping of a *session* to a *user*;

(6) *roles*: $S \rightarrow 2^R$ is the mapping of a *session* to a *role* set;

(7) *permissions*: $R \rightarrow 2^P$ is the mapping of a *role* to an *access permission* set.

## 3. Agricultural ontology storage and access control model

The RDF triple is a basic ontological structure and ontologies represented by OWL (Web Ontology Language) can be converted into the RDP triple representation. This study proposes an HBase-based storage and access control model for agricultural ontologies represented in this way. The RDF triple is defined as <*s*, *p*, *o*>, where *s* is the subject, *p* is the property, and *o* is the object. *s* and *o* are concepts and can be taken as nodes. *p* is a relation between concepts and can be taken as a line connecting two nodes. Two HBase tables SR_P_O and OR_P_S, representing the RDF, were designed according to the characteristics of the agricultural knowledge relevant to this study. They are shown in Tables 3 and 4.

Table 2 shows the structure of the SR_P_O table. The row key is composed of *s* and *r*. *r* represents a role, which is set by the access control and determines whether the data can be accessed by a user possessing a particular role. If the role as a property is placed in the column family, the table has to be gradually scanned to determine whether the knowledge can be accessed by the role. Therefore, placing the role in the row key can increase the efficiency. For example, the row key in the second row shows that $s_2$ can be accessed by $r_1$. Similarly, in the third row, $s_3$ can be accessed by $r_1$ but not $r_3$. If there are no *r*'s in the row key then the subject cannot be accessed. The

July – August 2020
ISSN: 0193-4120 Page No. 3119 - 3127

column family *property* has two columns: name and value. Name represents the property of the RDF triple and value represents the object of RDF triple as the property value. The storage structure in the OR_P_S table is similar to that of SR_P_O table. For the RDF triple <$s_3$, $p_2$, $o_2$>, $s_3$ can be accessed by $r_2$ and $r_3$ (see Table 2) whereas $o_2$ can only can be accessed by $r_2$ (see Table 3). Furthermore, $r_3$ can access $s_3$ but not the corresponding property value.

**Table 2.**Storage structure for SR_P_O table

| Row key | Time stamp | Column family: property | |
|---|---|---|---|
| | | Name | Value |
| $s_1$ | $t_1$ | property: $p_1$ | $o_1$ |
| | $t_2$ | property: $p_2$ | $o_2$ |
| $s_2$, $r_1$ | $t_3$ | property: $p_1$ | $o_3$ |
| | $t_4$ | property: $p_3$ | $o_1$ |
| | $t_5$ | property: $p_4$ | $o_4$ |
| $s_3$, $r_2$, $r_3$ | $t_6$ | property: $p_2$ | $o_2$ |
| | $t_7$ | property: $p_3$ | $o_3$ |

**Table 3.** Storage structure for the OR_P_S table

| Row key | Time stamp | Column family: property | |
|---|---|---|---|
| | | Name | Value |
| $o_1$, $r_1$ | $t_1$ | property: $p_1$ | $s_1$ |
| | $t_4$ | property: $p_3$ | $s_2$ |
| $o_2$, $r_2$ | $t_2$ | property: $p_2$ | $s_1$ |
| | $t_6$ | property: $p_2$ | $s_3$ |
| $o_3$, $r_1$, $r_2$, $r_3$ | $t_3$ | property: $p_1$ | $s_2$ |
| | $t_7$ | property: $p_3$ | $s_3$ |
| $o_4$, $r_1$ | $t_5$ | property: $p_4$ | $s_2$ |

The HBase storage model proposed in this work can satisfy seven of the triple pattern query modes shown in Table 4. In each pattern, *s*, *p*, and, *o* represent constants and *?s*, *?p*, and *?o* represent variables. The pattern *(?s, p, ?o)*, in which property is the only condition, is relatively rare. To satisfy this pattern, the storage model would need to have a table with property as the row key and this would increase the space cost of storage. If this pattern is requested, then either of the two tables can queried as a full table.

**Table 4.** Relationships between HBase tables and triple patterns

| Triple pattern | HBase table | |
|---|---|---|
| | SR_P_O | OR_P_S |
| (s, p, o) | √ | √ |
| (s, p, ?o) | √ | |
| (s, ?p, o) | √ | √ |
| (?s, p, o) | | √ |
| (s, ?p, ?o) | √ | |
| (?s, p, ?o) | √ | √ |
| (?s, ?p, o) | | √ |
| (?s, ?p, ?o) | √ | √ |

## III. SPARQLquery algorithms

### 1. Access control and matching triple pattern algorithm

The access control and matching triple pattern algorithm is used to judge whether the input triple pattern matches the HBase triple pattern and access is allowed.

**Algorithm 1** A&M_TP

Input: *t* = (*ts*, *tp*, *to*) is the input triple pattern, *r* is the user's role, *ht*= (*hs*, *hp*, *ho*) is the HBase triple pattern, and *ra* is a role array of *ht*

Output: "Access denied", True or False

```
if (r isn't in ra)
  then return "Access denied"
else
{
  if (t.ts is a variable || t.ts== ht.hs) && (t.tp is a
variable || t.tp == ht.hp) && (t.to is a variable ||
t.to == ht.ho)
    then return True
  else return False
}
```

*July – August 2020*
*ISSN: 0193-4120 Page No. 3119 - 3127*

## 2. Triple pattern query algorithms

The triple pattern query algorithms match the eight triple patterns. Algorithm 2 is the main algorithm for a single triple pattern query and is the entrance query algorithm.

**Algorithm 2** Q_TP
Input: $t$ = ($ts$, $tp$, $to$) is the input triple pattern, and $r$ is the user's role
Output: *Result* is the set of triple patterns that match $t$
if ($t.ts$ isn't a variable)
then *Result* = QSR_P_O($t$, $r$)
else if ($t.to$ isn't a variable)
then *Result* = QOR_P_S ($t$, $r$)
else
*Result* = TSR_P_O ($t$, $r$) or *Result* = TOR_P_S ($t$, $r$)

In Algorithm 2, if the subject of the triple pattern $t$ isn't a variable, then *result* is obtained by the QSR_P_O algorithm. If the object of the triple pattern $t$ isn't a variable, then *result* is obtained by the QOR_P_S algorithm. If only the property of triple pattern $t$ isn't a variable, then *result* is obtained through traversal of the SR_P_O or OR_P_S table.

**Algorithm 3** QSR_P_O
Input: $t$=($ts$, $tp$, $to$) is the input triple pattern, and $r$ is the user's role
Output: *Result* is the set of triple patterns that match $t$
*Result* = ∅
Find rows in the SR_P_O table whose row key contains $t.ts$, to generate a candidate set $cs$.
foreach (row: $cs$)
{
$ht_i$= ($hs_i$, $hp_i$, $ho_i$)is a triple pattern that row $i$ represents, $ht_i.hs_i$ is the subject of row $i$ in the row key, $ht_i.hp_i$ is the property name of row $i$, and $ht_i.ho_i$ is the property value of row $i$
$ra_i$ is a role array of row$_i$in row key
if (A&M_TP($t$, $r$, $ht_i$, $ra_i$) == True)
then *Result* = *Result* + $t$
}
return*Result*

**Algorithm 4** QOR_P_S
Input: $t$=($ts$, $tp$, $to$) is the input triple pattern, and $r$ is the user's role
Output: *Result* is the set of triple patterns that match $t$
*Result* = ∅
Find rows of the OR_P_S table whose row key contains $t.to$, to generate a candidate set $cs$.
foreach (row: $cs$)
{
$ht_i$= ($hs_i$, $hp_i$, $ho_i$) is a triple pattern that row$i$ represents, $ht_i.hs_i$ is the property value of row $i$, $ht_i.hp_i$ is the property name of row $i$, and $ht_i.ho_i$ is the object of row$i$in row key
$ra_i$ is a role array of row$_i$in row key

if (A&M_TP($t$, $r$, $ht_i$, $ra_i$) == True)
then *Result* = *Result* + $t$
}
return*Result*

**Algorithm 5** TSR_P_O
Input: $t$=($ts$, $tp$, $to$) is the input triple pattern, and $r$ is the user's role
Output: *Result* is the set of triple patterns that match $t$
foreach (row: SR_P_O)
{
$ht_i$= ($hs_i$, $hp_i$, $ho_i$)is a triple pattern that row$i$ represents, $ht_i.hs_i$ is the subject of row $i$in the row key, $ht_i.hp_i$ is the property name of row $i$, and $ht_i.ho_i$ is the property value of row $i$
$ra_i$ is a role array of row$_i$in row key
if ($r$ is in $ra_i$&&$t.tp$ == $ht_i.hp_i$)
then *Result* = *Result* + $t$
}
return *Result*

**Algorithm 6** TOR_P_S
Input: $t$=($ts$, $tp$, $to$) is the input triple pattern, and $r$ is the user's role

July – August 2020
ISSN: 0193-4120 Page No. 3119 - 3127

Output: *Result* is the set of triple patterns that match *t*
foreach (row: OR_P_S)
{
$ht_i$=($hs_i$, $hp_i$, $ho_i$)is a triple pattern that row*i* represents, $ht_i.hs_i$ is the property value of row *i*, $ht_i.hp_i$ is the property name of row *i*, $ht_i$. $ho_i$ is the object of row *i*in the row key
$ra_i$ is a role array of row$_i$in row key
if (*r* is in $ra_i$&&*t.tp* == $ht_i.hp_i$)
then *Resul t*= *Result* + *t*
}
return *Result*

According to the *ts* or *to* of the input triple pattern *t*, algorithms 3 and 4 first obtain rows that are eligible by indexing the row key and then matching with *t*. Algorithms 5 and 6 directly match *t* by traversing either the SR_P_O or OR_P_S table. Although algorithms 5 and 6 are slower than algorithms 3 and 4, they are not commonly needed in agricultural ontology services. Omitting a table whose row key contains property reduces the required storage space by one third.

## IV. Experiment and discussion

### 1. Experimental data

Experimental data concerned knowledge of tea pests. Five data sets were generated using a process similar to LUBM. The data profile is as follows:

In each ontology

- every atom is assigned by 1-4 roles, which are $r_1$, $r_2$, $r_3$ and $r_4$

In each order

- 3-9 families are subclasses of the order

In each family

- 2-7 pests are subclasses of the family
- every pest causes damage in 1-4 ways, which are plant-sucking, root-eating, leaf-eating, and tree-drilling
- every pest harms 1-4 of the tea parts, which are leaf, flower, bud, and root
- every pest is eaten by 1-5 predators
- every pest lives in 1-4 tea areas
- every pest is prevented using 1-4 methods, which are agricultural prevention, chemical prevention, physical prevention and biological prevention

The experiment used three data sets that were generated according to this profile. The sizes of the data sets are shown in Table 5.

**Table 5.** Data sets

| Data set | Orders | RDF triples | Size |
|---|---|---|---|
| D1 | 100 | 115362 | 7.1M |
| D2 | 7500 | 8409835 | 467.9M |
| D3 | 20000 | 22936721 | 1.26G |
| D4 | 50000 | 55321649 | 3.02G |
| D5 | 75000 | 86326693 | 4.86G |

### 2. Experimental environment

The experiment was carried out using virtualization software (VMware vSphere Hypervisor (ESXi) 6.0.0). The technical specification of the server was as follows: Lenovo T100 with a 3.10 GHz Intel(R) Xeon(R) E31225 quad-core processor, 20 G memory, and 1T hard disc. Four virtual servers were deployed on this machine, each with a vCPU, 4G memory, and 100G hard disc. All servers ran Ubuntu 14.04 LTS, Hadoop 1.1.2, HBase 0.94.27, and Oracle JDK 1.7. One of the four virtual servers was configured as a master server, on which NameNode and JobTracker were installed. The others were slave servers running DataNode and TastTracker.

### 3. Results and Discussion

The experiment consisted of the following five queries directed to HBase:

**Q1**

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

*July – August 2020*
*ISSN: 0193-4120 Page No. 3119 - 3127*

```
PREFIX te: <http://www.owl-
ontologies.com/teapest.owl#>
SELECT *
WHERE
{?X?Y?Z}
```

**Q2**

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX te: <http://www.owl-
ontologies.com/teapest.owl#>
SELECT ?X
WHERE
{?X rdf:typete:role}
```

**Q3**

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX te: <http://www.owl-
ontologies.com/teapest.owl#>
SELECT ?X
WHERE
{?X rdf:typete:role.
 ?X te:assigned te:pest1}
```

**Q4**

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX te: <http://www.owl-
ontologies.com/teapest.owl#>
SELECT ?X
WHERE
{?X rdf:typete:pest.
 ?X te:harmte:flower}
```

**Q5**

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX te: <http://www.owl-
ontologies.com/teapest.owl#>
SELECT ?X, ?Y, ?Z
WHERE
{?X rdf:typete:role.
 ?Y rdf:typete:pest.
 ?Z rdf:typete:area.
  ?X te:assigned ?Y.
  ?Y te:live ?Z.
  ?X te:assigned ?X}
```

**Q6**

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX te: <http://www.owl-
ontologies.com/teapest.owl#>
SELECT ?X, ?Y
WHERE
{?X rdf:typete:role.
 ?Y rdf:typete:pest.
 ?X te:assigned ?Y.
 ?Y te:damagete:leaf_eating}
```

Each query was ran five times on each of the databases, and the average response time was calculated. The average response times in MapReduce mode was compared with those obtained in stand-alone mode, as shown in Fig 1.

July – August 2020
ISSN: 0193-4120 Page No. 3119 - 3127

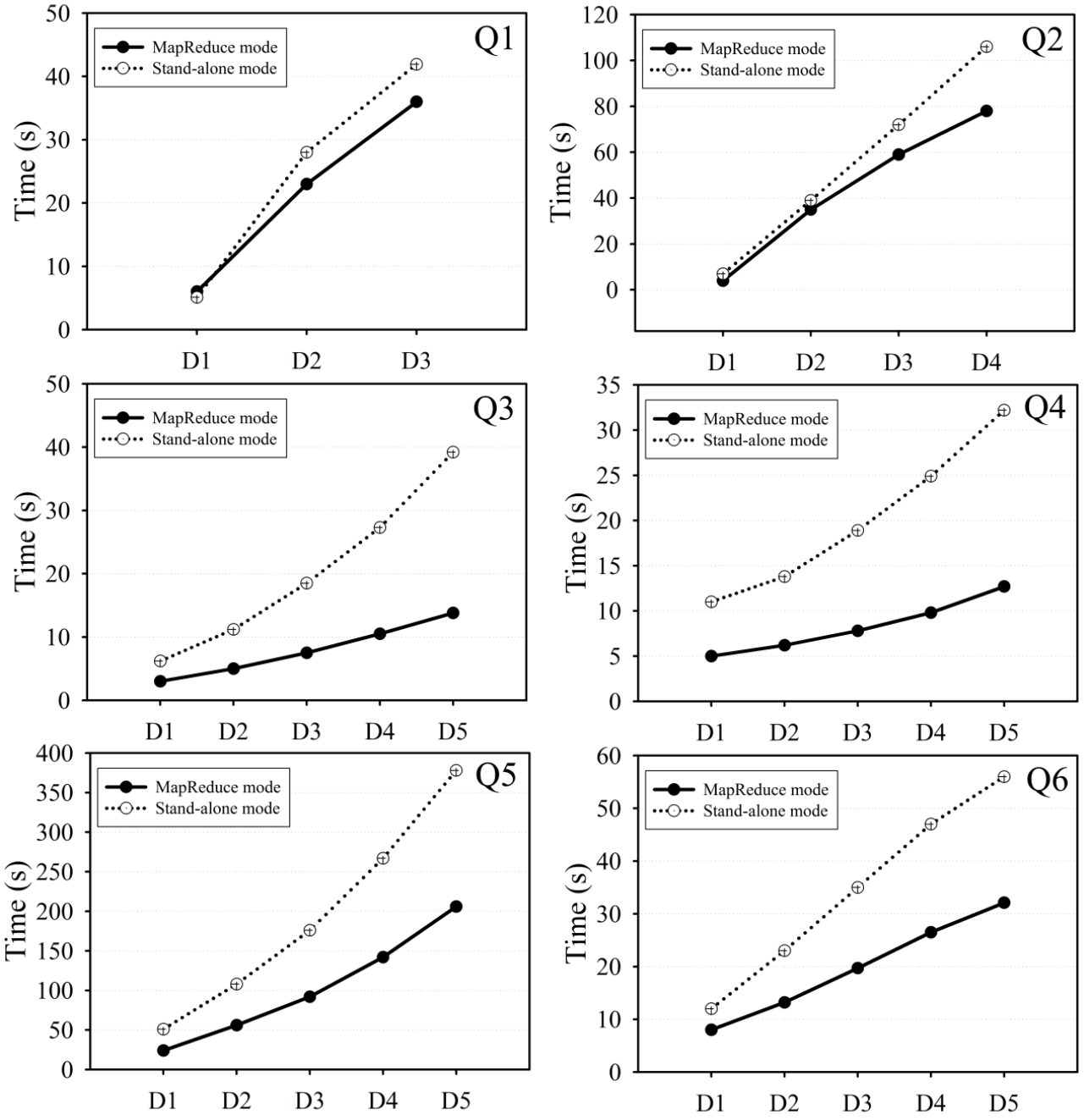


**Fig 1.** Comparison of query times between stand-alone and MapReduce modes. D1-D5 are data sets of increasing size.

Q1 and Q2 are queries based on a triple and are satisfied by a very large number of results. Q1 failed on datasets D1 and D2, and Q2 failed on dataset D2 because the number of results was too large. The MapReduce query times for Q1 and Q2 were shorter than in stand-alone but not very different.

The first clause in Q3 and Q4 defines the type of the variable. The second clause links to the first clause through the subject and its object is the main condition of the query. The query time in stand-alone mode increases more quickly than in MapReduce mode, indicating the scalability of MapReduce mode for Q3 and Q4.

Q5 is a query with multiple variables and triples. It is a complex query because of the complex connection between the clauses, the multiple kinds of logical relationships and the many alternative times of shared variables. This is the most complex query and the time in stand-alone mode grows fastest.

Q6 has two variables and four clauses. The last clause is the main condition for the query and so the alternative times of the shared variables are less than Q5. This improves efficiency. The query time in MapReduce mode grows more slowly than in stand-alone mode.

In summary, the model proposed in this paper performs better that the common stand-alone method and is suitable for handling large-scale ontologies. Provided there are sufficient virtual servers, the model is highly scalable.

## V. Conclusions

This paper proposes a model for agricultural ontology storage and access control based on Hbase, which not only successfully implements access control by setting a role property, but is also suitable for large-scale storage and query. Experimental ontologies were generated according to a set profile that is similar to LUBM. Six kinds of queries were designed and run in both MapReduce and stand-alone modes. The results of these experiments demonstrated the scalability of the proposed model.

**Acknowledgments.**ShaowenLi is thecorresponding author. This research was supported by Philosophy and Social Sciences Planning Projectof Anhui, China (GRANT_NUMBER: AHSKF2019D037).

## REFERENCES

[1] Kawtrakul A (2012) Ontology engineering and knowledge services for agriculture domain. Journal of Integrative Agriculture11(5):741-751.

[2] Lauser B, Keizer J (2002) A comprehensive framework for building multilingual domain ontologies: creating a prototype biosecurity ontology. In Proceedings of the International Conference on Dublin core and Metadata Applications, Germany, Dublin

[3] Haverkort A J, Top J L (2011) The potato ontology: delimitation of the domain, modeling concepts, and prospects of performance. Potato Research 54(2):119-136.

[4] Sun J, Li S W, Zhang L, Liu C, Zhao H Y, Yang JG. (2013) Ontology constructionin tea pest domain. In Proceedings of the 7th Chinese Semantic Web Symposium and 2nd Chinese Web Science Conference (CSWS 2013), China, Shanghai

[5] Abraham J, Brazier P, Chebotko A, et al (2010) Distributed Storage and Querying Techniques for a Semantic Web of Scientific Workflow Provenance. In Proceedings of the IEEE International Conference on Services Computing(SCC 2010), USA, Miami

[6] Sun J, Jin Q (2010) Scalable RDF store based on HBase and MapReduce. In Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE), China, Chengdu

[7] Papailiou N, Konstantinou I, Tsoumakos D, et al. (2012) H2RDF: adaptive query processing on RDF data in the cloud. In Proceedings of the 21st International Conference on World Wide Web, France, Lyon

[8] Punnoose R, Crainiceanu A, Rapp D (2012)Rya: a scalable RDF triple store for the clouds. In Proceedings of the 1st International Workshop on Cloud Intelligence, Turkey, Istanbul

[9] Lee H, Shao B, Kang U (2015) Fast graph mining with HBase. Information Sciences 315(C):56-66.

[10] Cruz J P, Kaji Y, Yanai N (2016) RBAC-SC: Role-based Access Control using Smart Contract. IEEE Access4:1-12.

[11] N. Damasceno C D, C. Masiero P, Simao A (2018) Similarity testing for role-based access control systems. Journal of Software Engineering Research and Development 6:1-37.