Research on Multi-Tenant Distributed Indexing for SaaS Application

Heng Li
College of Computer Science/Chongqing University, Chongqing, China
Email: lihengcq@gmail.com

Dan Yang and Xiaohong Zhang
College of Computer Science/Chongqing University, Chongqing, China
Email: {dyang, xhzhang}@cqu.edu.cn

Abstract—**Multi-tenant is the key feature for SaaS application, however, the traditional indexing mechanism has failed in multi-tenant shared scheme database. This paper proposed a multi-tenant distributed indexing mechanism. We create a global index first and then create the local index by MapReduce framework based on Hadoop. We also proposed the process of index update and index merging. Experimental results show that our multi-tenant distributed indexing mechanism has a good acceleration capability in creating index and high efficiency in retrieval, and provide good isolation for tenants to protect data safe.**

Index Terms—**multi-tenant, indexing, distribute, mapreduce**

I. Introduction

In recent years, multi-tenant database schema which should make a good balance between efficiency and customized[1] has become a hot topic for SaaS[2][3]application. Several works have been presented on design and implement multi-tenant database schema, such as "chunk folding" [4] 、"pivot table" [5] 、"xml table" [6] 、"meta data driven" [7] and so on, each technique has its own characteristics and applicable scenarios [8]. However, the research on index of multi-tenant database is still relatively lacking. The traditional indexing mechanism has failed in these schemas for the three reasons below:

(1)The traditional index [9][10] is created on the ordinary data table column attributes which doesn't involve multi-tenant situation. When different tenants search for result, the date obtained by the traditional index contains too much other information which has nothing to do with the tenants, it is a waste storage space, while making a safety hazard for tenant's own data cannot be isolated.

(2)The multi-tenant database needs to be extended according to the tenants' requirements, each schema has its own method. For example, "chunk folding" store the customized date in each chunk, "pivot tables" distinguish and store the customized date in int and string column, in "meta data driven" schema, different types of data will be stored in a varchar type field, tenants get the definition of metadata first, and then navigate to the extended data. Regardless of which kind of schema, it contains large custom property data which belonging to different logical column in the same column, the traditional indexing mechanism has failed, and cannot support combined index for multiple columns attribute.

(3)The date of each tenant is relative small and very large for the whole multi-tenant database. In shared schema model, it will consume a lot of time and take up a lot of unnecessary disk space if we create a unified data index in the traditional way, and the retrieval efficiency is low for the large index file.

To solve this problem, researchers proposed some solutions. In Force.com [11] Weismann used universal Table [12] to store tenant's business data and stored the logic index data in some pivot tables. It solved the problem of homogeneous index column data object in SaaS environment, but the article didn't diccuss more about data synchronization problems between perspective tables and sparse tables, and the platform didn't open its index maintenance strategy. In paper [13][14]it proposed a cracking mechanism in mangoDB database. It is a query-driven method, it dynamically adjust the order of the tuple in each query in accordance with the the avl tree. To transaction-based multi-tenant application, each update must respond in a timely manner, so this query-driven method has obvious defects. Kong lanju etc[15]proposed a multi-tenant index model in "key-value" schema, this model extend the multi-tenant database and add a indexing mechanism based on metadata. The model is built in a single relational database, it didn't explain the mechanism in large concurrent which will lead overload to single machine. In conclude, Stefan Aulbach proposed in his paper: the ideal database system for SaaS has not yet been developed, there left many problems to be solved [16].

This paper proposed a multi-tenant distributed indexing mechanism for SaaS application in cloud computing environment. Firstly, we create a global index and then create the local index by MapReduce[17] framework. It makes a good isolation for each tenant. Experiment showed that our mechanism has a good acceleration ability in creating index, tenants can search their own data quickly and exactly.

II. System Architecture

Figure 1 illustrates our proposed multi-tenant distribute indexing architecture. There are five main components in our design, including filter, source extractor, indexing module, search module and the distributed cloud storage system HDFS[18].

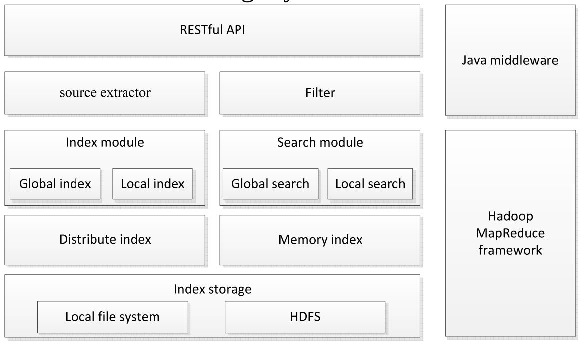
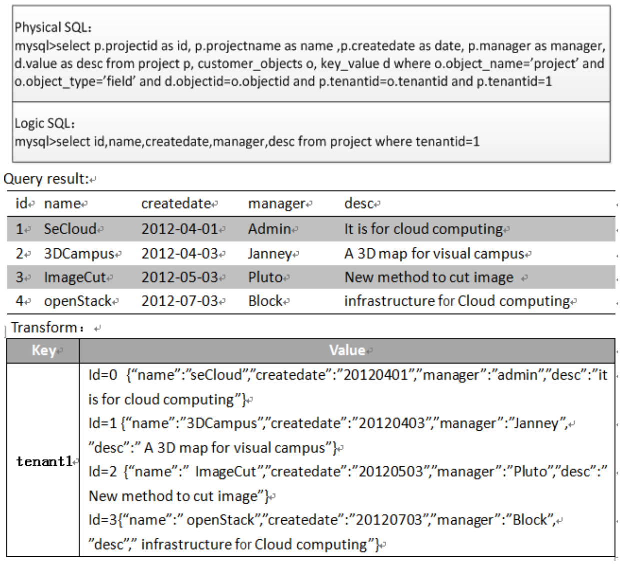


Figure 1. System architecture.

Our architecture has the following features:

(1) multi-tenant awareness and isolation

As Figure 2 shows, in the phase of creating global index, the source extractor get the query result from multi-tenant database by SQL, change the result set into JSON sequence, generate key-value pairs with the tenantid as output format. Then, put the key-value pairs to their own index server cluster by computing the key through distribute hash algorithm. This step implements the initial isolation of tenants' index data.



Physical SQL:

```
mysql>select p.projectid as id, p.projectname as name ,p.createdate as date, p.manager as manager, d.value as desc from project p, customer_objects o, key_value d where o.object_name='project' and o.object_type='field' and d.objectid=o.objectid and p.tenantid=o.tenantid and p.tenantid=1
```

Logic SQL:

```
mysql>select id,name,createdate,manager,desc from project where tenantid=1
```

Query result:

id	name	createdate	manager	desc
1	SeCloud	2012-04-01	Admin	It is for cloud computing
2	3DCampus	2012-04-03	Janney	A 3D map for visual campus
3	ImageCut	2012-05-03	Pluto	New method to cut image
4	openStack	2012-07-03	Block	infrastructure for Cloud computing

Transform:

Key	Value
tenant1	Id=0 {"name":"seCloud","createdate":"20120401","manager":"admin","desc":"it is for cloud computing"} Id=1 {"name":"3DCampus","createdate":"20120403","manager":"Janney", "desc":" A 3D map for visual campus"} Id=2 {"name":" ImageCut","createdate":"20120503","manager":"Pluto","desc":" New method to cut image"} Id=3{"name":" openStack","createdate":"20120703","manager":"Block", "desc"," infrastructure for Cloud computing"}

Figure 2. Source extraction

In the phase of creating local index, we get the tenantid and field out from the JSON document and joined them together as one field, then stored in the inverted index[19] which is illustrated in Figure 3, this is the secondary isolation.

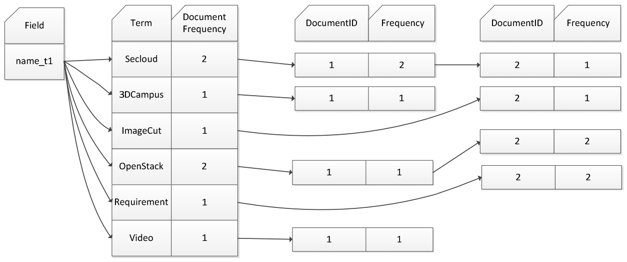


Figure 3. Inverted index

When tenant request for search, the filter obtained the tenantid from context, locate to the index server cluster by tenantid. In the phase of local search, $T=\{t1,t2,t3\ldots..t|t|\}$ is the feature dictionary of system, $U=\{u1,u2,u3\ldots u|u|\}$ is the tenants' collection, $F=\{f1,f2,f3\ldots..f|f|\}$ is the fields' collection, $D=(d1,d2,d3\ldots.d|d|)$ is the documents' collection, the index can be represented as a collection of $I=\{<f,t,d>|r(f,t,d,u)>0\}$, $r(f,t,d,u)$ is discriminant function, it returns a positive value when t is the feature item of d which joined f and u as a field. Suppose local search as $S=\{f,t,d,u\}$,

$$D(t)=\Phi(t,S)=\{d|(t,d)\in I\} \qquad (1)$$

D(t) is the document collection of retrieve vocabulary, Φ is retrieval functions of local search. We use r as discriminant function, and isolate the tenants' search quest by (f,t,d,u).

(2) Index cluster grouping

Different tenants have different index cluster groups. In each group, there is one server used as namenode, and the left severs are datanodes. When index data increases, we can increase the compute nodes of the cluster group to improve performance. Meanwhile, the Master/Slave model in each cluster group ensure the availability of Indexing Service in the case of internal nodes crash.

(3) Restful Api style

REST is proposed in a doctoral thesis in 2000, it is suitable to use in the complex network environment. Our proposed architecture used Restful interface to implement data access and exchange both in the phase of creating index and returning the search result, and it has the following characteristics: Since the stateless of communication itself, it allows different servers to handle different requests in a series of requests to improve server's scalability. It can simplify the software requirements by using browser as a client. REST dependence is smaller than other mechanisms superimposed on top of the HTTP protocol, and it don't need additional resource discovery mechanism. It has a better compatibility in the software technology evolution period.

(4) MapReduce Framework

We create local index by MapReduce framework. MapReduce is a computing framework which process and generate large data sets in map function, the programmers design the processing of every data block, and in reduce function, and there will be a reduction of the intermediate results. Users only need to specify the map and reduce functions to write distributed parallel programs. When running on the MapReduce program on cluster, the programmer need not worry about how to input data block, allocation and scheduling, at the same time the system will also handle the failure problem of some cluster nodes and the communication of inter-node. MapReduce applications get a list of key-value pairs as an input. The Map method processes each key-value pair in the input list separately, and outputs one or more key-value pairs as a result. map(key, value)-> [(key, value)]. The Reduce method aggregates the output of the Map method. It gets a key and a list of all values assigned to this key as an input, performs user defined aggregation on it and outputs one or more key-value pairs.reduce(key, [value]) ->[(key, value)]. Parallelization in the MapReduce framework is achieved by executing multiple Map and Reduce tasks concurrently on different machines in the cluster.

III. System Architecture

A. Create Index

As figure.4 (a) shows, we use consistency hash function on the machine node and the index data to implement unified computing, mapping them in a circle address space (0~232-1). After source extractor output the key-value pairs:<tenantid,queryResult>, we calculate the location of each pair by hash function on tenantid, find out which master node it belongs to, read each line of the value and generate the collection of original documents, then build a global B+ tree index(Figure 5 shows), at last, we upload the documents to the master node, save the global index file and original documents as key-value pairs to HDFS which the tenant corresponding.

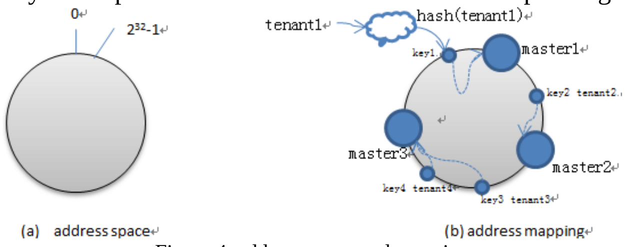


Figure 4. address space and mapping

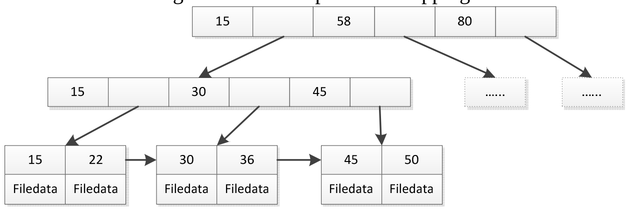


Figure 5.Global B+ tree

```
Algorithm 1. CreatGlobalIndex
Input:<tenantid,queryResult>
Output:documents collection
Begin
1.  Get collection of Key as S
2.  for key in S  // Iteration
      get the distribution area R by hash function on key
      get the MasterNodeLocation by Inverse operation on R
3.  get the value by key
4.  for value in V //second iteration,split the query results
      docs=readline(value)
      build B+ tree globalindex with docs
      save<globalindex,docs> to HDFS
5.    for doc in docs // third iteration,split docs to singal doc
        upload doc to MasterNodeLocation
      end for
    end for
  end for
END
```

B. Create Local Index

The master node in the cluster contains namenode and is responsible for distributing task called JobTracker. The slave node contains datanode and is responsible for job runs called TaskTracker. There are four steps below:

Step1. Master node divided the input files into N blocks, and send them to N slaves, meanwhile, the JobTracker send the CreatLocalInvertedIndexMap 、CreatLocalInvertedIndexReduce function to each slaves.

Step2.Each slave machine run CreatLocalInvertedIndexMap, resolve the input document, output <tenantid+field,term>as key-value pairs.

Step3.After sort the pairs in step2, each reducer runs the CreatLocalInvertedIndexReduce function, resolve the input pairs, joins tenantid and filed as luence field, use the terms as luence index value, then put the luence field and luence value in a luence document model. At last, output<tenantid, LuceneDocumentWrapper > as key-value pairs.

Step4.In hadoop's jobconf, we set the customized class LuceneOutputFormat as the output format, call luence index module to generate inverted index, save the temporary index to the local path of reduce node, after all reduce node finish, merge each index file for a complete index file and copy it to HDFS.

```
Algorithm 2 MRCreateLocalInvertedIndex
Input:documents collection
Output:index file
BEGIN
1.Init MapReduce,set and distribute mapreduce task
2.map(LongWritable key, Text text,  OutputCollector<Text, Text> output, Reporter reporter) //CreatLocalInvertedIndexMap task
      get Fields by resolve input document
      get Terms by resolve input document
      for term in Terms // Iteration
        set key1=tenantid join field
        set value1=term
        output(key1,value1)
      end for
3.reduce(Text key, Iterator<Text> value, OutputCollector<Text, LuceneDocumentWrapper> output, Reporter reporter) //CreatLocalInvertedIndexReduce
      create luencedoc as luence document model
      while iter.hasNext() // Iteration
      get term from value
      luencedoc.add(key,term) //put key and term into
                luence document model
        set key3= tenantid which obtained from key
        set value3=format(luencedoc)
        output(key3,value3) //reduce output
4.get value in step3 and call luence index module to generate inverted index
    save the temporary index to the local path of reduce node
    If all reduce nodes finish
      Optimization index and close
    merge each index file for a complete index file
    copy the index file to HDFS
    END
```

C. Search Process

Step1.When filter obtained the search request, insert ed it into the user's queue, then get the tenantid as key fro m the context, set the search request sentence as value, pu t them into hash table: hashtable<Key:tenantid,Value:List <SearchQuery>>. By this means, different tenants' search request is assigned to the corresponding master node.

Step2.When master node get the search request, it identify fields and keywords by lexical analysis and create a syntax tree by Syntax analysis, in figure 6.

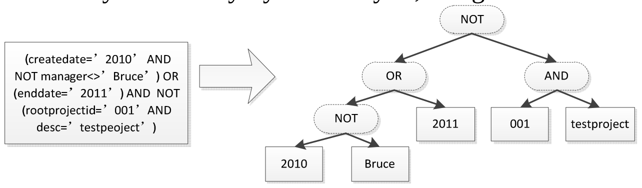

(createdate='2010' AND NOT manager<>'Bruce') OR (enddate='2011') AND NOT (rootprojectid='001' AND desc='testpeoject')
NOT
OR
AND
NOT
2011
001
testproject
2010
Bruce

Figure 6. Request sentence and syntax tree

Step3.Master node get the local index file from HDFS, obtain the document linked list by search in the local index, then computing relevency between query sentence q and document d through formula(2) below, return the linked list collection by the order of relevency.

$$score(q,d)=\frac{V_q \cdot V_d}{|V_q||V_d|}=\frac{\sum_{i=1}^{n} w_{i,q} w_{i,d}}{\sqrt{\sum_{i=1}^{n} w_{i,q}^2}\sqrt{\sum_{i=1}^{n} w_{i,d}^2}} \tag{2}$$

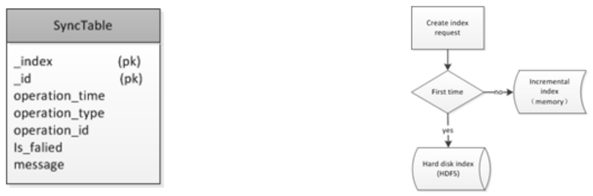
Wi,q is the weight of the word i in document d

Step4.Master node get the global index and original documents, search the linked list collection in the global index, return the corresponding original documents.

```
Algorithm 3 GetSerachResult
Input:search request sentence
Output:result documents
BEGIN
1.get the tenantid from context
set hashtable=<Key:tenantid,Value:List<SearchQuery>>
Distribute search request to master node
2.init D
3.do lexical analysis and syntax analysis with SearchQuery
4.join tenantid and field, create search object
5.get inverted index from hdfs to memory
6.for t in SearchQuery // Iteration
  get docs from inverted index with t
  and docs to D
7.do boolean operations on D,merge the result
8.computing the relevency by formula(2),return the documentid
9.get the original documents through search doucumentid in global index
10.return result documents
END
```

D. Index Update

Usually, indexing is a relatively time consuming process, espasially when data set is very large but there is less new record, it is too costly to rebuild the entire index. In this paper, we choose main index plus incremental index way to achieve index update. Firstly, we create a synchronization table in the multi-tenant database, Figure 7(a) shows, then we set a refresh interval time, when source extractor get the new data, it also update the operation_id in the synchronization table. At last, source extractor send the new data to master node and inform that it is an incremental update, master node create an incremental update index in the memory as Figure 7(b) shows.



(a) synchronization table (b) index update process

Figure 7. Index update

E. Index Merge

When the memory is not enough for growing index, we need to merge the index in memory and hard disk. Figure 8 shows the mode of index merge.

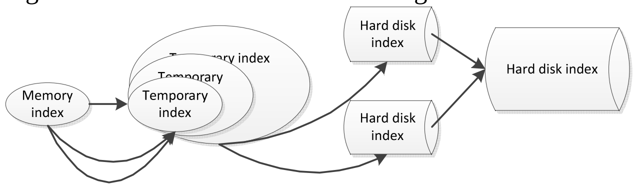


Figure 8. Index merge

```
Algorithm 4 MergeIndex
BEGIN
1.  set a Threshold
    if memory> Threshold
    add memory index to merge queue,create temporary index
2.  set interval time t1,t2
    if t>t1  save the temporary index to hard disk and clean the temporary index
    if t>t2  merge index in hard disk
END
```

IV. Experiment

The environment we conducted experiments on contained 3 clusters, each cluster composed of 4 dual processors nodes , each node has 1 GB of main memory. The index data of tenants 1-10 are distributed in three groups in the cluster. In each cluster, there was one namenode and the left were datanodes. The operating system was Linux Ubuntu 12. We write java client on multiple hosts simultaneously to simulate multi-tenant session, each client using multiple threads to achieve concurrent users, each session runned in its own thread and got the target database by connection pool. Since there is no standard data set for this task, we construct a base schema of a particular business domain application from the data schema in TPC-W database[20].We append a tenantid column so that it can be shared by multiple tenants as “Common tables”, we choose the “meta data driven” schema to store the customized data. The purpose of our experiment is testing the index creation and search capabilities in our multi-tenant distributed indexing mechanism.

A. *Acceleration Capability Test*

We simulate 3 tenants obtained the documents collection from database , each document is a json sequence which contained 22 key-value pairs. Table 1 shows the documents collection.

TABLE1.
DOCUMENT SIZE

	samples	features
Tenant1	30000	22
Tenant2	120000	22
Tenant3	240000	22

TABLE 2.
RUN TIMES FOR CREATING INDEX

	Tenant1	Tenant2	Tenant3
1node	71s	252s	481s
2nodes	60s	195s	310s
3nodes	35s	122s	212s

From the experiment results (Tables 2 and Figure 9(a)) it is possible to see that the time of each job decrease with the node increase, It means that we can significantly improve the indexing processing capabilities on the same scale data by increase the node. It should also be noted that the real time is less than the record time. This is because the background tasks of the mapreduce framework are relatively slow to start, so each separate mapreduce job that is started slows down the process. In conclude, to different tenants, when the index data is small, the acceleration capability can be ignored, so we can reduce the computing node to save cost, when the index data is large, we can improve the efficiency of indexing by increasing compute nodes

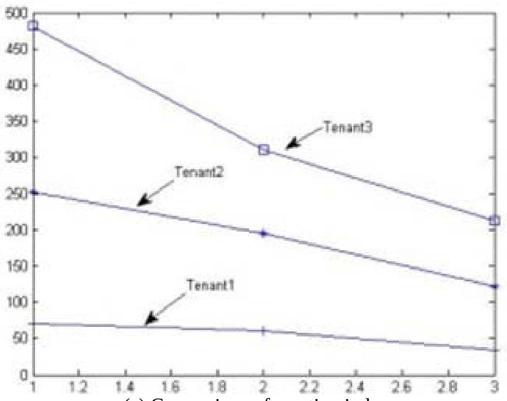
B. *Search Capabilities Test*

In the first experiment, we create three index in different size:73.7M,287.1M,538.8M. Now we simulate 3 tenants request for search, each tenant contained 100 users who send concurrent request. We test 7 types of requests :S1-S7.Table 3 record the average time of per user.

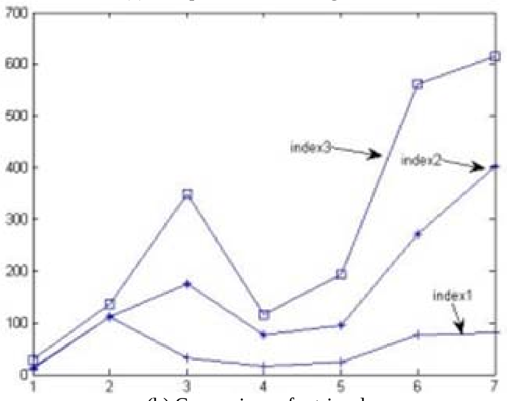
S1:get all document collections
S2:search value from all documents
S3:search in a range : 1<c_id<10000
S4:search by field value:Field=c_phone
S5:combination query1:c_id=111 AND c_phone= 7448718095072587
S6: combination query2:c_zip=prifix(9364) AND NOT c_id<10000 AND c_discount<0.4
S7:combination query 3:(1000<c_id<10000 AND NOT c_zip=prifix(9018)) OR (10000<c_id<20000 AND c_phone=prefix(7448))

TABLE 3.
RUN TIMES FOR RETRIEVAL

	S1	S2	S3	S4	S5	S6	S7
tenant1	10ms	111ms	31ms	15ms	23ms	77ms	81ms
tenant2	12ms	112ms	175ms	77ms	96ms	272ms	403ms
tenant3	28ms	137ms	350ms	115ms	194ms	562ms	615ms



(a) Comparison of creating index



(b) Comparison of retrieval

Figure 9.Experiment result

From Table.3 and Figure 9(b) we can see that it responsed fast in the case of concurrent retrieval, the result returned in milliseconds. Though the retrieval time is different in different request type, the impact on the end user is very weak. To improve retrieval efficiency, it is better to be avoided retrieve in the global scope, and should add the field as filter. We can also concluded that the index file size is an important factor affecting the efficiency of retrieval. If some individual tenants have a huge index data, we can add caching mechanism in the retrieval phase, or split the index file and add distributed retrieval algorithm to improve efficiency.

V. CONCLUSIONS

This paper proposed a Multi-Tenant distributed indexing mechanism for SaaS application, which has a good acceleration capability in creating index and high efficiency in retrieval, and provide good isolation for tenants to protect data safe. In future work, we will go on researching on caching mechanism and distributed retrieval algorithm, for further optimize performance and reduce costs.

ACKNOWLEDGMENT

This work is supported by State Natural Sciences Foundation Projects of China under Grant (60975015), Chongqing science research project funding under Grant (CSTC2009AC2057).

REFERENCES

[1] F.Burno."Exeuting an IP Protection Strategy in a SaaS Environment", http://www.slideshare.net/Rinky25/saas-environment, Jul. 22, 2011.

[2] M. Dokas, R. J.Wallace, R. Marinescu, S. Imran, and F. S.Foping. Towards a Novel Early Warning Service for State Agencies: A Feasibility Study. In Information Technologies in Environmental Engineering, pages 162–175. Springer Berlin Heidelberg, 2009.

[3] F. S. Foping, I. M. Dokas, J. Feehan, and S. Imran. On Using Software as a Service to Deploy an Early Warning Service. In International Conference on Enterprise Information Systems and Web Technologies, pages 161–168, Orlando,Florida, USA, 2009. ISRST.

[4] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1195-1206, New York, NY, USA, 2008. ACM.

[5] R. Agrawal, A. Somani, and Y. Xu. Storage and querying of e-commerce data. In VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases, pages 149{158. Morgan Kaufmann Publishers Inc., 2001.

[6] D. Florescu and D. Kossmann. Storing and querying XML data using an RDMBS. IEEE Data Eng. Bull, 22(3):27-34,1999.

[7] Li heng, Yang dan, Zhang xiaohong, A new meta-data driven data-sharing storage model for SaaS, International Journal of Computer Science Issues, Volume 9, Issue 6,2012

[8] S.Aulbach, T.Grust, D.Jaeobs, A.KemPer and M.Selbold, A Comparison of Flexible Schemas for Software as a Service, SIGMOD, 2009.

[9] Fuqing Zhao, An Improved PSO Algorithm with Decline Disturbance Index, Journal of Computers, 2011, 691-697

[10] Aiguo Li, RSR-tree: A Dynamic Multi-dimensional Index Structure, Journal of Computers,2011,2552-2558

[11] Salesforce AppExchange. http://www.salesforce.com

[12] The Foree.com Multitenant Architecture, Understanding the Design of Salesforee.com’s Internet Application Development Platform.

[13] Kersten M, Manegold S. Cracking the database sore Proceedings of the CIDR.Asilomar,CA,USA,2005:213-224

[14] Idreos S,KerstenM,Manegold S.Databasecracking Proceedings of the CIDR.Asilomar,CA,USA,2007:68-78

[15] Kong LanJu, Li QingZhong,Research on Index of Multi-Tenant Based on Key-Values for SaaS Application. chinese journal of computers. 2010 vol32 No.12

[16] S. Aulbach, D. Jacobs, A. Kemper, and M. Seibold, “A comparison of flexible schemas for software as a service,” in Proceedings of the 35th SIGMOD international conference on Management of data, ser. SIGMOD ’09. New York, NY, USA: ACM, 2009, pp. 881–888.

[17] Dean J, Ghemawat S. Map/Reduce: Simplied Data Processing on Large Clusters[C].In:OSDI 2004, San Francisco,2004,137-150

[18] S. Ghemawat, H. Gobioff, S.-T. Leung, The google file system, SIGOPS Operating Systems Review 37 (2003) 29–43.

[19] Shaojun Zhong, A Design of the Inverted Index Based on Web Document Comprehending, Journal of Computers,2011,664-670.

[20] http://www.tpc.org/tpcw/

Heng Li is a lecture in Chongqing University. Currently, he is a PhD student in College of Computer Science of Chongqing University. His interests are in cloud computing, data mining & machine learning.

Dan Yang is a professor of Chongqing University. Current research interests: data mining, computer vision, machine learning, enterprise informatization.

Xiaohong Zhang is a professor of Chongqing University.