Research Article

Multiple Wide Tables with Vertical Scalability in Multitenant Sensor Cloud Systems

Kun Ma and Bo Yang

Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan 250022, China

Correspondence should be addressed to Kun Ma; ise_mak@ujn.edu.cn

Received 14 November 2013; Revised 14 February 2014; Accepted 20 February 2014; Published 2 April 2014

Academic Editor: Gianluigi Ferrari

Copyright © 2014 K. Ma and B. Yang. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Software-as-a-service (SaaS) has emerged as a new computing paradigm to provide reliable software on demand. With such an inspiring motivation, sensor cloud system can benefit from this infrastructure. Generally, sharing database and schema is the most commonly used data storage model in the cloud. However, the data storage of tenants in the cloud is approaching schema null and evolution issues. To address these limitations, this paper proposes multitenant multiple wide tables with vertical scalability by analyzing the features of multitenant data. To solve schema null issue, extended vertical part is used to trim down the amount of schema null values. To reduce probability of schema evolution, wide table is divided into multiple clusters that we called multiple wide tables. This design reaches the balance between tenant customizing and its performance. Besides, the partition and correctness of multiple wide tables with vertical scalability are discussed in detail. The experimental results indicate that the solution of our multiple wide tables with vertical scalability is superior to single wide table, and single wide table with vertical scalability in the aspects of spatial intensity and read performance.

1. Introduction

Software-as-a-service (SaaS), in its broadest sense, refers to an on-demand software, which is delivered as services over the Internet [1]. SaaS has been incorporated into the strategy of all leading enterprise software companies to develop various multitenant applications [2]. One of the biggest selling points for these companies is the potential to reduce IT support costs. According to International Data Corporation’s (IDC) latest market report, SaaS will grow at a 29.2% annual compound rate through 2013–2017 [3]. With such an inspiring motivation, sensor cloud infrastructure is becoming popular because it can provide a flexible and configurable platform for several legacy Web sensor applications [4]. As for emerging sensor cloud system, serviceology for services is a general tendency [4], which explores theoretical structures of service concepts and generates scientific systematization of services and products. Since multitenancy is an essential component for SaaS, data storage model is a prime problem in multitenant sensor cloud applications. Due to increasing demand on sensor data and their support in multitenancy, sensor cloud architecture has been introduced as an integration of cloud computing into wireless sensor networks (WSNs) to innovate a number of other new techniques, such as WSN cloud services, databases, and applications. Some relevant publications conduct their own research on tenant customizing and its performance issues [5, 6]. Currently, there are at least three virtual data storage models: separate database, sharing the database with separate schemas, and sharing the database with sharing schemas [7, 8]. The most popular approach is the third one. Salesforce.com has passed up traditional virtualization software for custom technology that puts up to 10,000 customers on 15 databases and 100 servers using this approach [9].

There are some challenges in sensor cloud scenarios. First, cloud users were still approaching some practical difficulties, in particular when handling a large amount of sensor data [10]. The traditional data storage approach will become
a bottleneck problem when working on large data sets of interactive sensor data. The second challenge is how tenants access their sensor data in the sharing storage model with a logical view.

Wide table [11] was introduced as an authoritative multitenant sparse data storage model. This raw model has some limitations. First, the data in wide table is left-intensive. Since tenants consume columns of wide table from left to right, the right column might be assigned a value of null. This will consume a lot of spaces when working with massive tenants [12]. Another improved version of this wide table is called single wide table with vertical scalability [7]. This model stores personalized data in the extended vertical part. Although this model can reduce the amount of schema null values, it will lead to another schema evolution issue [13]. With the increase of tenants’ requirements, the length of tenants’ schema exceeds the preset width of core horizontal part. In this situation, some data in the vertical part are forced to move into the horizontal part to address schema null issue manually. Further details are presented in Sections 2 and 3.

Tenant customizing and its performance are two interdependent elements which depend on each other and limit each other in the context of multitenant data storage of sensor data. Our motivation of this paper is to achieve this balance between flexibility and complexity. In our paper, we propose multiple wide tables with vertical scalability in sensor cloud system, for the purpose of raising data intensity of data spaces, as well as spatial-temporal performance. Multitenant multiple wide tables are composed of core horizontal metadata and extended vertical metadata, in order to limit schema null issue. In addition, dividing a single wide table into various multiple wide tables with vertical scalability in sensor cloud system reduces the probability of schema evolution.

The contributions of this study are divided into several aspects. We attempt to address schema null and evolution issues using this new model. First, our data model provides a logical relational view with a minimum of resources. Second, our data model improves read and write performance by adding indexes to frequently accessed columns. Third, this model reduces the waste of null values in relational database with vertical scalability. Finally, dividing a wide table into multiple wide tables addresses schema evolution issue effectively. In conclusion, our data model achieves the balance between tenant customizing and its performance.

The remainder of the paper is organized as follows. Section 2 discusses the related work, and Section 3 presents multitenant multiple wide tables with vertical scalability of the sensor cloud system. First, we add some reserved columns dividing a wide table into different multiple wide tables to address the schema evolution issue. And then, the data model of multiple wide tables with vertical scalability is discussed in detail. Next, we analyze the correctness and implementations of this model. Section 4 gives the experimental evaluation of our multiple wide tables with vertical scalability. In Section 5, we discuss the superiority of our proposed multitenant multiple wide tables with vertical scalability in sensor cloud system with qualitative assessments. Brief conclusions and future work are outlined in the last section.

2. Related Work

In recent years, WSNs have become an established technology for a large number of applications, ranging from monitoring to event detection, as well as target tracking [14]. On one hand, cloud WSN applications with sharing storage usually lead to massive data. On the other hand, the cloud tenants have the requirements on personalization. That is to say that different tenants have different personalized columns. Accordingly, the storage of multitenant sharing data is more challenging.

This section outlines some classical sharing multitenant data storage models and features in sensor cloud system and introduces private table, extension table, document store, and wide table. Figure 1 illustrates the structure and metadata of different data models. Columns A and B are the customizing data, and columns C, D, and E are the personalized data.

2.1. Private Table. The most basic way to support extensibility is to give each tenant their own private table. In this simple approach, what the query-transformation layer needs to do is renaming tables. Thus, this approach has stronger pertinence and better expansibility on the customization and isolation. However, only moderate consolidation is provided, since
many tables are required. Aulbach et al. stated that relational structures might be made more competitive in the case of over 50,000 tables [15].

2.2. Extension Table. This approach is combined with splitting of the extension into a separate table. The sharing data are stored in the public storage (basic tables), while the separate data are stored in the extension table [16]. Because multiple tenants may use the same extension, extension tables as well as basic tables should be given an identity column of a tenant (tenantID). However, this approach has not yet resolved the expansion problem. It still needs a mechanism to map the basic and extended tables into one logical table.

2.3. Document Store. NoSQL databases are often highly optimized using key/value stores that are intended for simple retrieval and appending operations, with the goal being significant performance benefits in terms of latency and throughput [17]. Take document store for example, it provides JSON-style documents with dynamic schemas to store data. Although this dynamic structure minimizes null values, relational query is destroyed by this structure. The query of this model will produce an amount of connective operations, which make it difficult to refactor a relational tuple with n columns. We will compare this solution with our solution in Section 4.

2.4. Wide Table

2.4.1. Single Wide Table. Single wide table is usually highly sparsely populated so that most data can be fit into a line or a record [18]. Using this solution, queries are composed on only a small subset of the attributes. However, this model will produce the schema null problem, since it has too many null values. Moreover, it cannot provide dynamic customization capabilities since the reserved column is still limited in numbers. Indexing on a table generally brings up extra high storage and update costs. When the data set is sparse, the extra cost can be overcome by using a sparse index. A sparse index is one special kind of partial index, which maps only the non-NULL values to object identifiers [11].

2.4.2. Single Wide Table with Vertical Scalability. Another improved version of the single wide table is a single wide table with vertical scalability [12]. This model extracts the personalized data from the relational wide table and then describes it using extended vertical metadata. Each row in an instance of extended vertical metadata is a key/value pair, which is used to store the personalization of tenants to fulfill the requirements. In the event that the personalization of tenants is identical, extended vertical metadata can be omitted. The advantage of this approach is that it can reduce the waste of data resources efficiently. However, this model will produce schema evolution issue with the increase of customizing columns. In this case, the length of tenants’ schema exceeds the preset width of wide table.

2.5. Challenges of Current Wide Table. In the wide table solution, all the tenants’ sensor data are stored in a sharing database. The features of multitenant sensor data have brought with them our own two new challenges.

First, it leads to the schema null issue when tenants do not customize some columns. If one tenant has customized this column, a value might be assigned to be useful at run time. If one tenant has not customized this column, the value of this column must be null. Multitenant sensor data that already looks like is left-intensive, since tenants consume the data from left to right. This is a key issue of data storage of multitenants to be solved.

In the relational wide table, we suppose \( C_i \) as the customizing column count of the data, \( N_i \) is the personalized (non-customized) column count of the data, and \( i \) is the index number of the tenants. We denote the length of a wide table as \( \text{LENGTH} = C_{i} + N_{i} \). We suppose \( R_{i} \) as the row number of tenant \( i \), and \( n \) as the tenant number. The data intensity of relational wide table is

\[
\rho = \frac{\sum_{i=1}^{n} (R_{i} \cdot C_{i})}{\sum_{i=1}^{n} (\text{LENGTH} \cdot R_{i})} = \left( 1 + \frac{\sum_{i=1}^{n} (R_{i} \cdot N_{i})}{\sum_{i=1}^{n} (R_{i} \cdot C_{i})} \right)^{-1}.
\]

It is observed that \( 0 \leq \rho \leq 1 \). When \( N_{i} = 0, \rho \) is 1. When \( C_{i} \) is far smaller than \( N_{i}, \rho \) is the largest, which causes schema null issue.

Second, it will lead to the schema evolution issue in the case of single wide table solution with vertical scalability. When the length of tenants’ schema exceeds the preset width of a single wide table, the extended vertical part of a single wide table should be transferred to the core horizontal part. We call this transfer schema evolution. Therefore, fixed length of customizing columns can introduce a flaw that affects the performance of the cloud database. In addition, \( C_{i}, N_{i}, \) and \( R_{i} \) in the expression of data intensity are the forecast on the tenants’ customizing. Due to inaccurate forecast on the tenants’ requirements, single wide table with vertical scalability is approaching schema evolution issue.

3. Multitenant Multiple Wide Tables with Vertical Scalability

3.1. Schema Evolution Issue. In order to reduce the probability of schema evolution, we attempt to add some additional reserved columns to improve single wide table with vertical scalability. Moreover, the wide table is partitioned to formulate multiple wide tables by the sum of the amount of customizing and reserved columns. The structure of multiple wide tables with vertical scalability is shown in Figure 2. One of multiple wide tables is divided into customizing and reserved columns, which is called a cluster. The personalized data are stored in the extended vertical part, and the customizing data are stored in the core horizontal part. On one hand, vertical scalability can reduce the access granularity of tuples. On the other hand, the reserved columns in the core horizontal part can reduce the probability of schema evolution.
3.2. Data Model. In our paper, multitenant multiple wide tables with vertical scalability are composed of core horizontal metadata and extended vertical metadata. We use $ore = (tenantID, rowKey, columns, reservedColumns)$ as the core horizontal metadata, where $tenantID$ is the unique identification of the tenant, $rowKey$ is the primary key of the wide table, and $columns$ and $reservedColumns$ are a set of customizing and reserved columns, respectively. We use $extended = (tenantID, rowKey, columnKey, columnValue)$ as the extended vertical metadata, where $tenantID$ is the unique identification of the tenant, $rowKey$ is the primary key of the wide table, and $columnKey$ and $columnValue$ are the key/value of extended vertical metadata.

In the context of multiple wide tables, tenants’ data are spread over different single wide tables. That is to say that multiple wide tables with gradient distribution replace several single wide tables, meeting the demand on dynamic storage requirements of tenants. The tenants’ data are determined in either the core horizontal part or the extended vertical part by the requirements on the tenants’ customization.

3.3. Vertical Scalability. In order to personalize the data of tenants, we design the vertical scaling method to solve the sparse and schema null issues of a single wide table. That is to say that we extract the personalized data from the wide table and then describe it using vertical metadata. As depicted in Figure 2, extended vertical metadata reduces the waste of data resources efficiently. Each row in the extended vertical metadata is a key/value pair, which is used to store the personalization of tenants to fulfill the requirements of different tenants. In the case that the personalization of tenants is the same, the extended vertical metadata can be omitted.

Although the extended vertical metadata reduces schema null values effectively, it increases the computational complexity. For the purpose of evaluating whether the data are stored in the core horizontal part or the extended vertical part, we give the evaluation function to determine whether the extended vertical part is worth adopting. We describe the evaluation function of column $k$, $F_k$, as

$$F_k = \mu_k \alpha_k \beta_k + (1 - \mu_k) \gamma_k,$$

(2)

where

(i) $\mu_k$ is the proportion that different tenants customize column $k$;
(ii) $\mu_k = (\sum_{i=1}^{n} (k \geq C_i) \cdot 1) / n$;
(iii) $\alpha_k$ is the access number of the $k$th column;
(iv) $\beta_k$ is the access number of the tables containing column $k$;
(v) $\gamma_k$ is the service factor that the tenant serves column $k$.

It is shown that the larger $F_k$ is, the less appropriate column $k$ is in the extended vertical metadata.

Since the data type of $ColumnValue$ is weak, we take advantage of the families of multiple wide tables with different data types. Suppose there are three column families: family 1 just has a varchar attribute, family 2 just has a timestamp attribute, and family 3 just has a digit attribute. The table using hybrid representation is described in Figure 3.
3.4. Table Partition. This section discusses the reasonable partition of multiple wide tables with vertical scalability. Consider that tenant \(i\) has \(C_i\) customizing columns, and the maximum value of customizing columns is \(\text{MAX}(C_i)\). Tenants’ data are assigned to wide table \(j\), which has \(\Delta_j\) reserved columns. There is no need for tenant \(i\) to adjust the schema, as long as \(C'_i \leq C_i + \Delta_j\). If \(C'_i > C_i + \Delta_j\), tenant \(i\) need to transfer the data to wide table \(j + 1\). Therefore, reasonable amount of reserved columns and partition can reduce the probability of schema evolution.

The statistical analysis of the customizing requirements shows that customizing columns of tenants approximately corresponds to the normal distribution \(N(\mu, \sigma)\), where \(\mu\) is the average amount of customizing columns of tenants, and \(\sigma\) reflects the differentiation of the amount of customizing columns. We use the normal distribution to fit the frequency count of customizing columns of tenants. Frequency histogram is partition basis of multiple wide tables. According to three-sigma rule [19], almost all (99.73\%) of the values lie within 3 standard deviations of the mean. We divide \([\mu - 3 \times \sigma, \mu + 3 \times \sigma]\) into \(k\) intervals, and each interval corresponds to a kind of wide table. A mass of statistical analysis indicates two facts. First, we should make thinner granular partitions of intervals (the number of common columns is close to \(\mu\)) with smaller \(C_i\) and \(\Delta_j\). Second, we should make coarser granular partitions of intervals (the number of common columns is far greater than \(\mu\)) with larger \(C_i\) and \(\Delta_j\). Furthermore, another two facts are concluded from the statistics. First, the amount of partition intervals \(k\) is set at 5 to 6 where \(\text{MAX}(C_i) \leq 50\). Second, the amount of partition intervals \(k\) is set at 10 to 20 where \(\text{MAX}(C_i) > 50\). For example, when \(\text{MAX}(C_i) \leq 50\), the intervals \([\text{MAX}(\mu - 3 \times \sigma, 0), \mu - 1.25 \times \sigma], [\mu - 1.25 \times \sigma, \mu - 0.25 \times \sigma], [\mu - 0.25 \times \sigma, \mu + 0.25 \times \sigma], [\mu + 0.25 \times \sigma, \mu + 1.25 \times \sigma], [\mu + 1.25 \times \sigma, \mu + 3 \times \sigma]\) are reasonable partitions.

3.5. Correctness Analysis. In this section, we apply equivalence analysis between multitenant multiple wide tables with vertical scalability and traditional relational table.

**Theorem 1.** Multitenant multiple wide tables with vertical scalability are equivalent to a relational table.

(1) A relational table can be converted into multitenant multiple wide tables with vertical scalability.

We denote a relational table using the mathematical relation. This relational table has \(n\) customizing columns, \(m\) reserved columns, and \(r\) personalized columns:

\[
R(tenantID, rowKey, CS, PS, ES),
\]

where

(i) \(tenantID\) is the unique identification of the tenant;
(ii) \(rowKey\) is the primary key of a relational table;
(iii) \(CS\) is a set of customizing columns, \(PS\) is a set of reserved columns, and \(ES\) is a set of personalized columns;
(iv) \(CS = \{C_i | C_i \text{ is a customizing column}\}, \text{where } 1 \leq i \leq n;\)

\[
\text{SELECT * FROM (}
\text{extended UNPIVOT (}
\text{MAX(columnValue)}
\text{FOR columnKey IN (SELECT DISTINCT columnKey}
\text{FROM extended)}
\text{))};
\]

**Algorithm 1**

(v) \(PS = \{P_j | P_j \text{ is a reserved column}\}, \text{where } 1 \leq j \leq m;\)

(vi) \(ES = \{E_k | E_k \text{ is a personalized column}\}, \text{where } 1 \leq k \leq r.\)

If attribute \(A\) is contained within \(CS (A \in CS)\), attribute \(A\) is assigned to store customizing data in core horizontal part of the multiple wide tables.

If attribute \(A\) is contained within \(PS (A \in PS)\), attribute \(A\) is assigned to store reserved data in core horizontal part of the multiple wide tables.

If attribute \(A\) is contained within \(ES (A \in ES)\), attribute \(A\) is assigned to store personalized data in extended vertical part of the multiple wide tables. The mapping from \(A\) to extended vertical part of the multiple wide tables is called UNPIVOT [20], denoted as

\[
extended = \bigcup_{i=1}^{m} \left( \pi_{tenantID, rowKey, A', R'} \right)
\]

where

(i) \(R' = \sigma_{A \in \{E_1, E_2, ..., E_m\}} R;\)
(ii) \(\pi\) is a relational projection operator;
(iii) \(\sigma\) is a relational selection operator.

The mapping rule is also described in Algorithm 1.

(2) Multitenant multiple wide tables with vertical scalability can be converted into a relational table. We use core horizontal part and extended vertical part of multiple wide tables to refactor the relational table.

First, we pivot the extended vertical metadata \(extended\) to the horizontal storage, denoted as

\[
extended' = \pi_{tenantID, rowKey, columnValue,R'},
\]

where

(i) \(R' = \sigma_{columnKey = \cdot A', extended};\)
(ii) \(\pi\) is a relational projection operator;
(iii) \(\sigma\) is a relational selection operator;
(iv) \(\circ\) is a relational connection operator.

The mapping rule is also described in Algorithm 2.
Next, relation $S$ is calculated by the connection of $extended'$ and core by tenantID and rowKey, denoted as

$$ S = σ_{tenantID=tenantID}(rowKey=rowKey)F, \quad (6) $$

where

(i) $F = corecoextended'$.

Finally, relation $R = π_A S$ is the refactored relational table, where

(i) $A = \{ a \mid a \in (columns\{reservedColumns\} - \{rowKey\}) \}$.

3.6. Implementation. Multiple wide tables with vertical scalability are composed of various single wide tables with vertical scalability, and each wide table is composed of two parts: core horizontal metadata and extended vertical metadata. We use MySQL 5.6 GA to store both parts.

Figure 4 illustrates a running example that shows the mapping between multitenant wide tables with vertical scalability and a relational table. This operation is transparent to end users with the help of the transformation view. We use the example of Figure 4 to describe the creation and read process. We take column room and temperature in the core horizontal part, and light, smoke and humidity in the extended vertical part.

When we insert three records {room : 101, temperature : 30, light : 1, smoke : 0}, {room : 102, temperature : 26, light : 1, smoke : 0}, and {room : 201, temperature : 28, humidity : 50%}. Columns room and temperature are extracted to store in the core horizontal part, while columns light, smoke, and humidity are extracted to store in the extended vertical part.

We provide a read-only logical relational view for query using Algorithm 3.

Therefore, the developers can query the relational view regardless the actual storage, which is similar to a virtual wide table. This operation is transparent to the developers.

4. Performance Evaluation

The experiments were performed on a server with the following configurations: an Intel(R) Core(TM) CPU 15-2300 3.0 GHz server with 8 GB RAM, a 100 M Realtek 8139 NIC in 100 M LAN. This server was deployed with the operating system CentOS 6.4 x 64. We have done four experiments to evaluate the performance of our multiple wide tables with vertical scalability. We take advantage of the actual sensor data collected from a WebSocket-based real-time monitoring system for remote intelligent buildings [14], which serves different cloud tenants. We use multiple wide tables with vertical scalability to store different tenants' data. We use MySQL 5.6 GA to store the core horizontal part as well as the extended vertical part.

4.1. Spatial Intensity. We select cloud sensor data generated by 20 different tenants to do the first special experiment. We compare single wide table with 50 columns, with our multiple wide tables with vertical scalability. Consider that tenants consume wide table from left to right. The amount of customizing column of each tenant is as $C = \{4, 6, 8, 9, 10, 12, 15, 16, 17, 19, 20, 22, 25, 28, 35, 40, 46, 48\}$, where $C_i$ is the number of customizing column of tenant $i$. As shown in Figure 5, $C_i$ is in accordance with the normal distribution. The average amount of customizing column is 21.3, and its standard deviation is 11.5. To measure the data intensity, we suppose that each tenant has the same number of rows. Since $\text{MAX}(C_i) \leq 50$, we divide the partition of customizing columns into five intervals: $[0, 7]$, $[8, 18]$, $[19, 24]$, $[25, 36]$, and $[37, 56]$. Each interval reflects a kind of a wide table, which is shown in Table 1. If the single wide table solution is adopted, the overall data intensity of the single wide table is 0.38. If multiple wide tables with vertical scalability solution is adopted, the data intensity is 0.90. The experimental results show that multiple wide tables with vertical scalability can enhance the degree of data intensity, and reduce schema null values. We can use a finer-grained partition to enhance the intensity when the tenants’ requirements are fixed. With the changes of tenants’ requirements, as long as the length of customizing columns is no more than the sum of customizing and reserved columns, multiple wide tables work without the adjustment of the schema. Therefore, the probability of schema evolution is reduced.

4.2. Read Performance. We have done the second experiment to evaluate the read performance. Consider that there are five columns ($E, D, C, B,$ and $A$) that are stored in a wide table. To change the selectivity of predicates, we use different distributions for the column values. Among 20,000 sensor data, there are only 10 records for which column $A$ is defined; then there are 100, 1,000, 10,000, and 100,000 records for which columns $B$ through $E$ are defined, respectively, $A$ and $B$ in this example, which are both rarely selective. We have made indexes on tenantID, rowKey, columnKey and columnValue to optimize the query.

We have executed six queries on the sharing multi-tenant sensor data with the same query condition. The queries are to extract data from columns $A, B, C, D, E,$ and all columns. The query time of each query of three solutions is shown in Figure 6. Solution 1 is single wide table with vertical scalability, where columns $E, D,$ and $C$ are stored...
### Extended vertical metadata

| TenantID | RowKey | ColumnKey | ColumnValue |
|----------|--------|-----------|-------------|
| 1        | 1      | Light     | 1           |
| 1        | 1      | Smoke     | 0           |
| 1        | 2      | Light     | 1           |
| 1        | 2      | Smoke     | 0           |
| 2        | 3      | Humidity  | 50%         |

### Pivot/Unpivot

| TenantID | RowKey | Light | Smoke | Humidity |
|----------|--------|-------|-------|----------|
| 1        | 1      | 1     | 0     | Null     |
| 1        | 2      | 1     | 0     | Null     |
| 2        | 3      | Null  | Null  | 50%      |

### Wide table

| TenantID | Room | Temperature |
|----------|------|-------------|
| 1        | 101  | 30          |
| 1        | 102  | 26          |
| 2        | 201  | 28          |

| TenantID | Room | Temperature | Light | Smoke | Humidity |
|----------|------|-------------|-------|-------|----------|
| 1        | 101  | 30          | 1     | 0     | Null     |
| 1        | 102  | 26          | 1     | 0     | Null     |
| 2        | 201  | 28          | Null  | Null  | 50%      |

**Figure 4:** A mapping example between the multi-tenant wide table and relational table.

**Table 1:** Partition and intensity of multiple wide tables with vertical scalability.

| Number | Customizing | Reversed | Partition | Intensity |
|--------|-------------|----------|-----------|-----------|
| 1      | 4           | 3        | 4, 6      | 0.57      |
| 2      | 8           | 10       | 8, 9, 10, 12, 15, 16, 17 | 0.44     |
| 3      | 19          | 5        | 19, 20, 22, 22 | 0.79     |
| 4      | 25          | 11       | 25, 28, 35 | 0.69     |
| 5      | 37          | 19       | 40, 46    | 0.66     |

**Algorithm 3**

```sql
CREATE VIEW v_wide AS
SELECT * FROM (
    SELECT *
    FROM core w
    LEFT JOIN extended e
    ON w.tenantID = e.tenantID and w.rowKey = e.rowKey
) PIVOT(MIN(ColumnValue)
FOR ColumnKey IN (SELECT DISTINCT ColumnKey
FROM ExtendedData)
);
```

In the core horizontal part, while $B$ and $A$ are stored in the extended vertical part. Extended vertical metadata in solution 1 is pivoted to an inner join of the wide table. Solution 2 is single wide table without vertical scalability, where columns $E$, $D$, $C$, $B$, and $A$ are stored in a wide table. Solution 3 is our multitenant multiple wide tables with vertical scalability, where extended vertical part is mapped into a logical relational view for further relational query.

By contrast, solution 1 is more time consuming than solutions 2 and 3. That is due to the cost of transformation from the extended vertical data to the core horizontal data in solution 1. After all the data has been assembled, the performance difference between solutions 1, 2, and 3 is the execution cost of PIVOT. Then, another index seek is
performed on the rows to make the performance of PIVOT close to the wide table. Compared with the deficiency of read performance, solution 3 is better at strong spatial intensity than solution 2.

Next, we have made the third experiment to observe the effect of concurrent transactions on read performance. We use multithreads to simulate a large number of requests to wide tables with 10,000 records. We issue a query “query the monitoring sensor data in the past three months” for 10 times and record the average query time. We have added solution 4 for the sake of further comparison. In solution 4, multitenant sensor data are stored in the form of document stores with free and dynamic schema. Figure 7 shows the average query time of each query with different solutions. We begin by contrasting four solutions in the last paragraph.

Due to the data refactor of extended vertical part, query time of solution 1 is always the largest. Since solution 3 adopts a relational view for query, the effect of concurrent transactions on read performance is small. When the amount of concurrency is less than 1,200, the query time of solution 3 is larger than solution 2. When the amount of concurrency is more than 1,200, the query time of solution 2 is rising sharply. The query time of solution 3 is close to solution 4, but solution 4 cannot provide the support of SQL. It is difficult for the legacy sensor applications to utilize this solution transparently.

4.3. Write Performance. We have made the last experiment to observe the effect of concurrent transactions on write performance. We measure the number of writes per second. In solutions 1 and 3; writing one record will generate more than one record in the database because of the vertical extension. We have inserted 100,000 sensor records to observe the throughput of different solutions. Figure 8 shows the write performance. When the amount of concurrency is less than 600, the throughput of solution 3 is the largest. In the case of high concurrency, solution 3 is a little worse than solution 4. Among wide table-like solutions (solutions 1, 2, and 3), our solution has ensured optimal write performance. Future work is target to explore read/write splitting to improve the write performance in the case of high concurrency.

5. Discussion

Tenant customizing and its performance are two interdependent elements. Single wide table is basic sparse table to store tenant’s data. Due to the sparsity feature, this model causes too much space. Although single wide table with vertical scalability addresses schema null issue, it causes an
Motivated by sharing multitenant storage in sensor cloud system, this paper attempts to propose a better solution to store multitenant sensor data in the context of cloud computing. Compared with current multitenant data models, we propose multitenant multiple wide tables with vertical scalability of the sensor cloud system. This model consists of two parts: core horizontal part and extended vertical part. Core horizontal part is used to store customizing data, while extended vertical part is used to store personalized data. To address the schema evolution issue, we divide a wide table into multiple clusters that we called multiple wide tables. On one hand, this model mainly focuses on solving schema null and evolution issues with high scalability. On the other hand, our proposed model meets the demand on tenants’ personalization. Further, the partition of multiple wide tables with vertical scalability is discussed in detail. Besides, we illustrate the equivalence analysis of the multitenant multiple wide tables with vertical scalability and traditional relational table. A running example of the transformation is presented at the end. The experimental results show that our multiple wide tables with vertical scalability is superior to single wide table and single wide table with vertical scalability in the aspects of spatial intensity and read/write performance.

Multiple wide tables with vertical scalability that we have proposed formulate a kind of sharing multitenant sensor data storage model. We attempt to store tenants’ data together with the same schema. In this solution, the cloud sensor data are maintained for centralized management. Therefore, this method has some limitations in the distributed environment. Future work is targeted to explore some distributed techniques to optimize our method, such as data sharing, data partition, and read/write splitting.

**Conflict of Interests**

The authors declare that there is no conflict of interests regarding the publication of this paper.

**Acknowledgments**

This work was supported by the Doctoral Fund of University of Jinan (XBS1237), the Teaching Research Project of University of Jinan (J1344), the Technology Development Program of Shandong Province (2011GGX10116), and the National Key Technology R&D Program (2012BAF12B07).

**References**

[1] M. Armbrust, A. Fox, R. Griffith et al., “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] C. D and H. G, “Cloud computing: benefits, risks and recommendations for information security,” *Communications in Computer and Information Science*, vol. 72, no. 1, 17 pages, 2009.

[3] M. J. Turner, "Worldwide cloud systems management software 2013c2017 forecast," Research Report 240276, International Data Corporation, 2013.

[4] A. Alamri, W. S. Ansari, M. M. Hassan, M. S. Hossain, A. Alelaiwi, and M. A. Hossain, "A survey on sensor-cloud: architecture, applications, and approaches," *International Journal of Distributed Sensor Networks*, vol. 2013, Article ID 917923, 18 pages, 2013.

[5] K. Park, H. Won, and S. Hur, “SaaS tagging platform: integrating and customizing on-demand applications supporting multi-tenancy,” in *Proceedings of the 14th International Conference on Advanced Communication Technology (ICACT’12)*, pp. 961–964, 2012.

[6] J. Lee, S. Kang, and S. JinHur, "Web-based development framework for customizing Java-based business logic of SaaS application," in *Proceedings of the 14th International Conference on Advanced Communication Technology (ICACT ’12)*, pp. 130–1313, 2012.

[7] K. Ma, Z. Chen, A. Abraham, B. Yang, and R. Sun, "A transparent data middleware in support of multi-tenancy," in *Proceedings of the 7th International Conference on Next Generation Web Services Practices (NWeSP ’11)*, pp. 1–5, 2011.
[8] K. Ma, B. Yang, and A. Abraham, “A template-based model transformation approach for deriving multi-tenant SaaS applications,” Acta Polytechnica Hungarica, vol. 9, no. 2, pp. 25–41, 2012.

[9] R. Woollen, “The internal design of salesforce.com’s multi-tenant architecture,” in Proceedings of the 1st ACM Symposium on Cloud Computing, p. 161, Indianapolis, Ind, USA, 2010.

[10] H. Tirri, “Indexing the real world: sensing, big data and mobility,” in Proceedings of the 19th Annual International Conference on Mobile Computing & Networking, pp. 415–416, 2013.

[11] E. Chu, J. Beckmann, and J. Naughton, “The case for a wide-table approach to manage sparse relational data sets,” in Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 821–832, 2007.

[12] K. Ma, B. Yang, Z. Chen, B. Ma, and Q. Li, “From the modern web applications to the multi-tenant SaaS solution,” Chinese Journal on Communications, vol. 32, no. 9, pp. 133–138, 2011.

[13] S. Wu, S. Zhang, and L. Kong, “Schema evolution via multiversion metadata in SaaS,” Procedia Engineering, vol. 29, pp. 133–138, 2012.

[14] K. Ma and R. Sun, “Introducing websocket-based real-time monitoring system for remote intelligent buildings,” International Journal of Distributed Sensor Networks, vol. 2013, Article ID 867693, 10 pages, 2013.

[15] S. Aulbach, D. Jacobs, A. Kemper, and M. Seibold, “A comparison of flexible schemas for software as a service,” in Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD ’09), pp. 881–888, 2009.

[16] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, Addison-Wesley, Upper Saddle River, NJ, USA, 2010.

[17] R. Cattell, “Scalable SQL and NoSQL data stores,” ACM SIGMOD Record, vol. 39, no. 4, pp. 12–27, 2010.

[18] B. Yang, W. Qian, and A. Zhou, “Using wide table to manage web data: a survey,” Frontiers of Computer Science in China, vol. 2, no. 3, pp. 211–223, 2008.

[19] F. Pukelsheim, “The three sigma rule,” The American Statistician, vol. 48, no. 2, pp. 88–91, 1994.

[20] B. Jelen and M. Alexander, Pivot Table Data Crunching, Que, New York, NY, USA, 2010.
Submit your manuscripts at
http://www.hindawi.com