International Journal of Database Theory and Application
Vol.8, No.5 (2015), pp.183-194
http://dx.doi.org/10.14257/ijdta.2015.8.5.15

AtMe: An Online Multi-tenant Social Networking Service in Campus

Kun Ma, Zijie Tang, Zhe Yang and Shuwei Yao

Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan 250022, China
ise_mak@ujn.edu.cn

Abstract

Recently, more and more undergraduates have turned Internet to find support and take part in society in campus. However, current SNS systems are not oriented to a specific campus. In this paper, we design an online multi-tenant social networking service in campus, including At Helper and At Society. At Helper aims at improving the success ratio of the help process with a multi-tenant architecture, and At Society provides social services using popular instant messaging mobile application. Both modules change the traditional operation and maintenance to rent services. Besides, the key innovative design, such as multi-tenancy and access-aware data cache, are discussed in detail.

Keywords: *Social networking; multi-tenancy; online Help; helper recommendation; feed propagation; big data; cloud computing*

1. Introduction

Recently, online systems support social networking services (SNSs) for undergraduates in campus. Users are used to be immersed in the virtual world to share their experiences in trouble and take part in society online. Several well-known applications enable online help process, such as Facebook and Twitter [1]. There are some insufficiencies of these applications. First, all the SNS systems are not oriented to a specific campus. Thus, the success rate of current SNS-like online mutual help system is relatively low. Users are accustomed to say their anxieties, needs and feelings, but would not like to offer help actively. Second, all the SNS systems are not providing society management in campus.

To address these limitations, we design a multi-tenant social networking service in campus. This system is mainly divided into both At Helper and At Society modules, which provide multi-tenant [2] cloud services to tenants. This way changes the traditional operation and maintenance into service renting. In the context of both At Helper and At Society, a tenant is a campus owning its undergraduates.

At Helper have several new features. First, both pre and post recommendation methods are proposed to strengthen the success of the help process. On one hand, a helper recommendation algorithm is proposed to find competent helpers. On the other hand, candidates with similar user behaviors and interests are recommended to users based on the comparisons with the published help feeds. Second, help feed propagation is used to notify potential helpers combing with both push and pull techniques.

At Society have several new features. First, we design a society initiation module using popular instant messaging mobile application. Users can submit a joining request anywhere. Second, an automatic scheduling of a society activity is designed using school timetables extracted from another third-party educational management system.

The remainder of the paper is organized as follows. The related work is discussed in Section 2. In Section 3, we propose the online multi-tenant social networking service in campus. This system is mainly divided into At Helper and At Society modules. More technical details (multi-tenancy and access-aware data cache) are discussed in Section 4.

Section 5 presents the application prospect of this system. Brief conclusions are outlined in the last section.

2. Related Work

2.1. Social Networking Service in Campus

Mutual help origins from offline human communication. Along with the development of Internet techniques (Web 2.0 and interactive HTML5), online mutual help is topic-oriented and procedure-oriented application. Compared with offline help, online help is an effective way to present the right information to the right user at the right time in most efficient form. Currently, most online help system is a SNS-like system that gives assistance with offline help. That is to say that online help acts as an extension of offline help by providing quick answers to supplicants. There are many advantages of online mutual help, such as instantaneity and expansibility. In this paper, we take advantage of online mutual help to increase offline mutual understanding. Before the release of a help feed, we will seek some related solutions by analyzing the historical knowledge base automatically. After the release of a help feed, some referees are recommended.

Our At Helper system provides a feed of the status of inserts, updates and deletes. There are two categories of information transfer: push and pull patterns [3]. The first approach is push pattern. In this solution, the help feeds are pushed to the referees who offer assistance. It will create several records in case of one post. Although the feeds are well partitioned using shardings, the pushing amount of the feeds is very huge in an instant of an application. Another approach is pull pattern. In this solution, only one feed and its relation are created. Users can pull the feeds actively. Although the feeds are stored in the cache, the frequent query of the cache brings tremendous pressure to the application. The pull pattern is less efficient than the push pattern in the field of query performances. Another improvement of time-partitioned pull pattern. This pattern stores the feeds over a period of time in different partitions. The historical help feeds are archived in the historical database. Frequent users will access the latest feeds. Compared with the current approaches, we benefit the help feed propagation from the integration of push and pull.

2.2. Key Techniques

2.2.1. Multi-Tenancy

Instead of multi-instance architectures, multi-tenancy refers to a principle in Software as a service (SaaS) architecture where a single instance of the software runs on a server, serving multiple tenants [4]. With a multi-tenant architecture, a software application is designed to virtually partition its data as well as configurations. Each tenant works with a virtual application instance, and all tenants share the same physical application instance. There are many approaches for data storage in a multi-tenant deployment [2, 5-6], such as separate schema in the same database instance and shared schema in the same database. Separate schema in the same database instance houses multiple tenants in the same database but with different schemas. The typical feature of this solution is that each tenant has its own set of tables, views, functions and stored procedures separately. Another approach is shared schema in the same database, where all the tenant data are mixed together with the same schema. For example, a given table can include records from multiple tenants stored in any order. A tenantID column associates each record with the appropriate tenant. This approach has the lowest hardware and backup costs. However, this approach may incur additional effort in the area of privacy and security to ensure that tenants can never access other tenants' data. After evaluating the pros and cons of two approaches for data storage, we decided to pursue the schema sharing solution.

There are some data storage models for multi-tenancy. The obvious model is extension table [7]. The sharing data are stored in the public storage (basic tables), and the separate data are stored in the extension tables. Because multiple tenants may use the same extensions, the extension tables as well as the base tables should be given a tenant column. However, this approach has not yet resolved the expansion problem. Mapping the basic and extension tables into one logical table is essential. Another model is wide table [8-9], which is usually highly sparsely populated so that most data can be fit into a line or a record. However, this model will create a fatal issue called schema null, since too many null values exist in this storage model. Moreover, it cannot provide dynamic customization capabilities since the reserved column is still limited in numbers. Compared with current two approaches, we propose an integrated data model to solve schema null issues by the extension of the wide table.

2.2.2. Data Cache

Cache techniques are deserved methods for the storage of big data. Generally, Relational Database Management System (RDBMS) is best fitted for both On-Line Transaction Processing (OLTP) and On-Line Analytical Processing (OLAP) systems [10]. However, RDBMS is approaching a bottleneck in the query performance issues once the data to be processed or queried become larger [11].

The general cache solution is RDBMS with query cache [12-14] (hereinafter RDBMS + Query Cache). It increases the performance of duplicated RDBMS-based queries to minimize database load. The existence of cheap memory resources is an opportunity to scale the applications even further. For example, Hibernate ORM framework [15] uses EhCache to implement a second level cache of the query results. TxCache [16] is a transparent caching framework that supports transactions with snapshot isolation. It is designed for RDBMSs that supports multi-version concurrency control, and extends them to produce invalidation tags in the presence of updates. However, one of the biggest challenges of these approaches is data consistency between the cache and physical data, which makes it difficult to guarantee the hit rate of the cache.

3. Multi-Tenant Social Networking Service

3.1. At Helper

At Helper is the core business of the online multi-tenant social networking service in campus. We attempt to optimize the traditional online help process from two perspectives, which is different from the current online help products.

3.1.1. Helper Recommendation

Before the supplicant posts a help feed, the helper recommendation algorithm firstly compare the feed with the historical successful knowledge base. To improve the success rate of matching, we tag every successful mutual help knowledge. We use Levenshtein distance algorithm [17] to measure the differences between help contents with the knowledge base. And then, the help cases with over 0.7 similarity are provided with the supplicant as a reference. To some extent, the method can stop the post of help feed, since this help is solved using historical experiences. Compared with current similar products, this strategy sometimes overcomes the difficulty in advance.

The Levenshtein distance between two strings a and b is given by $lev_{a,b}(i,j)$ where $0<i<length(a)$, $0<j<length(b)$, and

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & if\ min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+[a_i \neq b_i] \end{cases} & otherwise \end{cases}$$

The first element in the minimum corresponds to a deletion (from a to b), the second corresponds to an insertion, and the third corresponds to a match or mismatch. The similarity of both two help feeds is *similarity=1 - lev*$_{a,b}$*(length(a),length(b))/max(length(a),length(b))*. We have made some tests to conclude that the threshold of the similarity is 0.6. The help cases with over 0.6 similarity are provided with the supplicant as a reference. The system we design will remind the supplicants that they can stop this process if the system matches a reference solution automatically. Compared with current similar products, this strategy sometimes overcome the difficulty in advance.

After the supplicant posts a help feed, a helper recommendation algorithm based on user affinity (user relationship) is proposed to find potential helpers who have similar habits and behaviors. We define user affinity with the integration of basic affinity, interactive affinity, and similarity affinity. The basic affinity is defined as

$$B(X,Y) = \sum_{i=1}^{n} w_i \times B_i,$$

Where w_i is the coefficient of the aspect B_i. For instance, mutual friends and help frequentness are the influence factors. We use social networking graph *G=(N, E, W)* to define interactive affinity reflecting the behaviors of users. *N* is a set of user nodes, *E* is a set of directed behavior edges, and *W* is the weight of the edges *E*. We consider the minimum weight as the interactive affinity, denoted as

$$E(X,Y) = min(W(X,Y), W(Y,X)).$$

The user nodes without the direct link may also has similarity affinity, denoted as

$$S(X,Y) = \sum_{i=1}^{n} w_i \times S_i$$

For example, user habits and personal information are both influence factors. We define user intimacy depending on the summation of the above factors, denoted as

$$F(X,Y) = \alpha \times B(X,Y) + \beta \times E(X,Y) + (1-\alpha-\beta) \times S(X,Y),$$

Where a $(0<a<1)$ and β $(0<\beta<1)$ are the important factors. Next, the ratings of the potential helpers are ranked by the user affinity. Those ordered in the front are recommended to the supplicants. Compared with current similar products, this strategy improves the success rate to some extent.

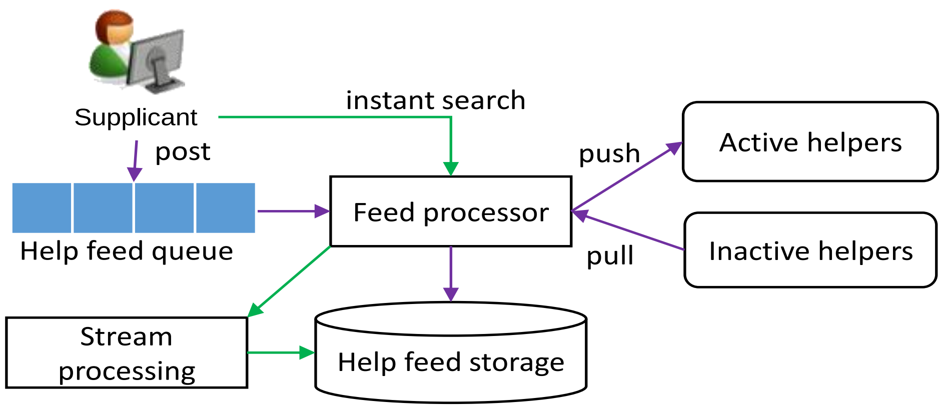

Figure 1. Help Feed Propagation Architecture

3.1.2. Help Feed Propagation

We use the integration of push and pull to assist help feed propagation. The architecture is shown in Figure 1 After the post of a feed, the feed is put in the post help queue. We use the key/tuple stores as the cache of help feeds to accelerate read/write efficiency. Next, the push (push help feeds to potential helpers) and pull (inactive potential helpers pull help feeds actively) strategies are applied to the help feed. This integration strategy greatly reduced the amount of needless pushing and pulling, decreasing the performance loss. Another new feature is that we apply stream processing techniques to implement instaneous processing.

Next, we propose an optimization scheme to the feed propagation. We classify the feed propagation into two categories.

- In order to reduce the amount of pushing data, the system will push the help feeds to the potential helpers actively.
- The inactive potential helpers will pull the help feeds actively when they login this system next time.

This integration strategy greatly reduced the amount of needless pushing and pulling, decreasing the performance loss.

3.2. At Society

At society is another business of the online multi-tenant social networking service in campus. We attempt to optimize the traditional society management from two perspectives, which is different from current association management products.

3.2.1. Society Initiation using WeChat

Anyone who wants to join an association can submit an application using popular instant messaging mobile application - Wechat. We attempt to optimize the society initiation process as follows, which is shown in Figure 2 First, an applicant followed a WeChat public/service account using mobile client. Second, an applicant submitted a proposal with resume using WeChat instant messaging interface. Next, the message response server in At Society recorded the request, and send a reminder to the administrator. Finally, the administrator reviewed the request.

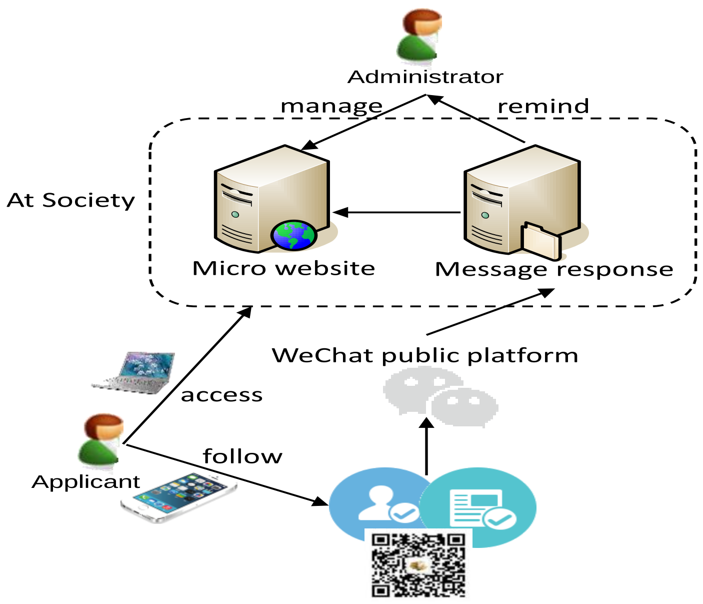

Figure 2. Society Initiation Process

3.2.2. Society Activity Scheduling

We design a scheduling module to assist the organization of each society activity. The school timetables of all society members are imported to this system first. We use OAuth 2.0 protocol to implement the extraction of school timetable from the pages of the educational management system. After that, the best time (most members have no class) is computed. This module of AtMe can greatly reduce the workload and improve the effectiveness of organizing an activity.

We design a visually-rich interactive timeline to display all the activity events of a society. Figure 3 shows the activity timeline of a society.

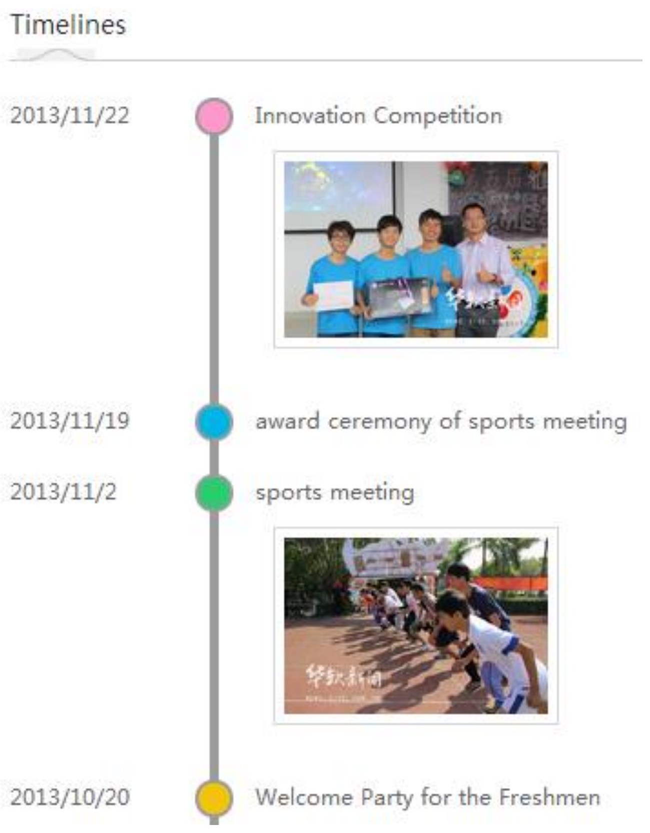

Figure 3. Activity Timeline of a Society

4. Discussion

4.1. Multi-Tenancy

The typical multi-tenant architecture of our framework is shown in Figure 4. The architecture is composed of two points of view in order to reduce the complexity: business layer and database layer. User requests with context of tenantID are split to formulate new data query and manipulation statements.

- From the top is business layer, where User requests with context of tenantID are received.
- At the bottom is database layer, where we use shared database schema with tenantID column as the storage. In the database layer, the system extracts the tenantID from the Web context, and formulate new data query and manipulation statements (insert/update/delete) statements. The formulation is shown in Figure 4. For the query statement and update/delete manipulation statements, tenantID is

connected with a criteria to formulate the new statement. For the insert manipulation statement, the value of context tenantID is as a new column.

4.2. Access-Aware Data Cache

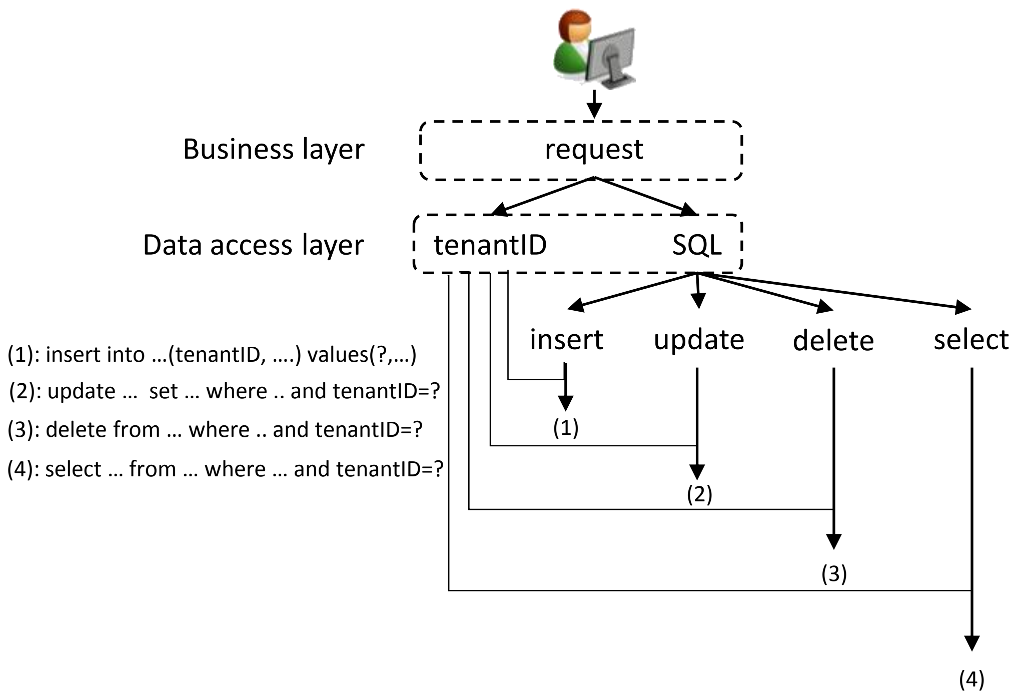
Business layer
request
Data access layer
tenantID
SQL
insert
update
delete
select
(1): insert into …(tenantID, ….) values(?,…)
(2): update … set … where .. and tenantID=?
(3): delete from … where .. and tenantID=?
(4): select … from … where … and tenantID=?
(1)
(2)
(3)
(4)

Figure 4. Multi-Tenancy Architecture

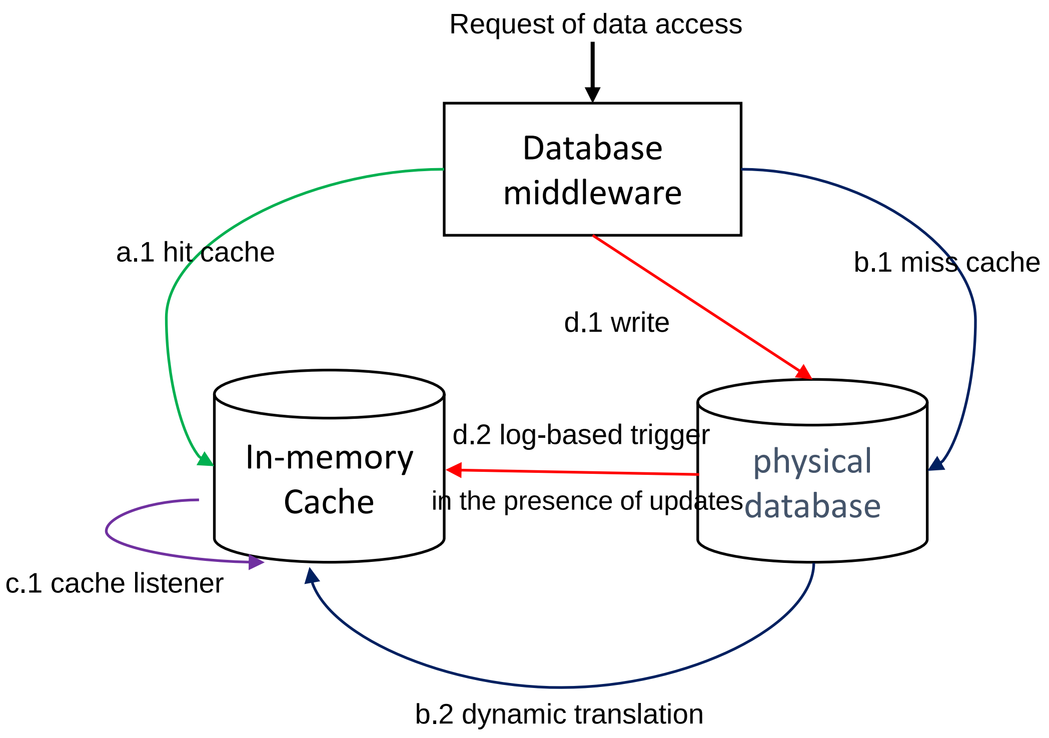

Figure 5. Access-Aware Data Cache Architecture

To accelerate the access of data, we design an access-aware data cache framework. Figure 5 shows the architecture of access-aware data cache. If the data of one column of the query miss the in-memory cache, they are obtained from the original database.

4.2.1. Cache Hit and Miss

We intercept the query statements to redirect it to our proposed access-aware data cache to determine whether the data are in the cache or not. If the data in this column of the query hit the cache (see Figure 5 a.1), this framework returns the results from the cache. On the contrary, if the data in this column of the query miss the cache (see Figure 5 b.1), the middleware returns the results from the original database. If the subtraction from the current column access frequency rate to the average one exceeds a threshold, the data of this column need to be dynamically translated into the cache (see Figure 5 b.2). In case of the hit or miss of the cache, the column access frequency count need the rectification.

4.2.2. Cache Listener

We design cache listener (see Figure 5 c.1) to observe the usage of the column in the cache. If the current column access frequency rate is smaller than the average column access frequency rate, the column access frequency count need to be abated. If the subtraction from the current column access frequency rate to the average one decides a negative threshold, we should remove the data in this column in the cache.

4.2.3. Transaction

We intercept the transaction operations (see Figure 5 d.1) to redirect it to our proposed access-aware data cache to synchronize it into the cache (see Figure 5 d.2) to ensure the consistency. In case of the insert transaction operation, we will insert the data of this column into the cache when this column exists in the cache. In case of the delete transaction operation, we will delete the corresponding cache when this column exists in the cache. In case of the update transaction operation, we will rectify the data in this column in the case of the existence in the cache.

4.2.4. Cache Consistency and Transaction Processing

We just discuss the transaction processing except schema rectifications. When the application requests a transaction operation (insert/update/delete), the access-aware data cache framework will make an initial decision about the synchronization between the physical database and the cache. If the updates of this column are located both in the cache and the RDBMS, we need to synchronize the updated into the cache. The updates will divide into three categories. In case of the update, we will rectify the corresponding cache. In case of the insert, we will insert the data of this column into the cache. In case of the delete, we will delete the corresponding cache.

We use the log-based change data capture to intercept the changes of the physical database. For the transaction session, we capture the changed data from the binary log events, and construct the corresponding cache.

Our capture engine has been designed to support extraction from different data sources. This flexibility is achieved by allowing different capturers to be developed and plugged into the capture engine. This capturer is typically an embedded thread and must follow a few semantic constraints. We talk in detail about the implementation in the next section.

In the scenario of translation from physical database to data cache, only the changed data reading from the transactional binary log are concerned. The database products often provide row-based replication (RBR) events to capture changed data from the binary log. We can use the binlog API to connect the listener to get the incremental data. By interpreting the contents of the database transaction log one can capture the changes made to the database in a non-intrusive manner.

In access-aware data cache framework, the transaction operations (create/update/delete) are guaranteed by the database itself. We only provide the transaction and fault tolerance of dynamic translation (see Figure 5 b.2) and log-based trigger in the presence of updates (see Figure 5 d.2). These two styles are abstracted into a record translation from the source database to target cache. We provide a record pool to store the temporal data to be translated. Before translation, each record is saved in this pool first. After the successful translation, the temporal record will be removed from the pool. In any case of translation failure, we can re-translate the record in the pool.

We provide two methods to deal with fault tolerance. The first method is translation replay. It is based on the temporary record pool. In case of failure, the record is taken out from the upstream pool, and played again. Another method is lineage tracking. Each record translation is guaranteed to be processed at least once by verifying the transactional unique ID. In case of failure, the translation is recovered from the checking point. In case of high demand of instantaneity, batch record translation is put in a transaction by the pipeline techniques.

5. Application Prospect

The publicity of AtMe is with the link http://www.aite5.com. This system is providing renting services to each tenant with different second-level domains. Now we have three tenants: University of Jinan (http://ujn.aite5.com), Qingdao Science and Technology University (http://qust.aite5.com), and Hunan University of Science and Technology (http://hnust.aite5.com).

6. Conclusions

As the social networking grows, it becomes important to understand how this technology can be applied to assist social networking in campus. Therefore, we have proposed a service of the online multi-tenant social networking. This system is mainly composed of At Helper and At Society.

First, we clarify the design of At Helper module. To improve the success rate of mutual help, we proposed a helper recommendation algorithm to accelerate the help process. Furthermore, we have presented the integration solution of push and pull to propagate the feeds to the potential helpers.

Second, we clarify the design of At Society module. To enhance communication among society members, we proposed society initiation using WeChat. With the help of WeChat public platform, applicants can submit a proposal using mobile WeChat, and members can arrange a society activity intelligently.

Acknowledgments

This work was supported by the Doctoral Fund of University of Jinan (XBS1237), the Shandong Provincial Natural Science Foundation (ZR2014FQ029), the Teaching Research Project of University of Jinan (J1344), the National Training Program of Innovation and Entrepreneurship for Undergraduates (201410427030), and the Student Research Training of University of Jinan (SRT2014335 & SRT2014339).

References

[1] H. Kwak, C. Lee, H. Park and Sue Moon, "What is twitter, a social network or a news media", In Proceedings of the 19th international conference on World wide web, ACM, **(2010)**, pp. 591-600.

[2] K. Ma, Z. Chen, A. Abraham, B. Yang and R. Sun, "A transparent data middleware in support of multi-tenancy", In Next Generation Web Services Practices (NWeSP), 2011 7th International Conference on,

IEEE, **(2011)**, pp. 1-5.
[3] P. Balouchian and A. Elci, "Data portability across social networks", In New Challenges in Computational Collective Intelligence, Springer, **(2009)**, pp. 181-190.
[4] R. Mietzner, F. Leymann and T. Unger, "Horizontal and vertical combination of multi-tenancy patterns in service-oriented applications", Enterprise Information Systems, vol. 5, no. 1, **(2011)**, pp. 59-77.
[5] 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, **(2012)**, pp. 25-41.
[6] K. Ma and Z. Tang, "An online social mutual help architecture for multitenant mobile clouds", International Journal of Intelligent Information and Database Systems, vol. 8, **(2014)**.
[7] R. Elmasri and S. B. Navathe, "Fundamentals of database systems", Pearson, **(2014)**.
[8] 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, **(2008)**, pp. 211-223.
[9] K. Ma and B. Yang, "Multiple wide tables with vertical scalability in multitenant sensor cloud systems", International Journal of Distributed Sensor Networks, vol. 2014, **(2014)**.
[10] H. Plattner, "A common database approach for oltp and olap using an in-memory column database", In Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, ACM, **(2009)**, pp. 1-2.
[11] S. Ahirrao and R. Ingle, "Scalable transactions in cloud data stores", In Advance Computing Conference (IACC), 2013 IEEE 3rd International, IEEE, **(2013)**, pp. 116-119.
[12] S. Ghandeharizadeh and J. Yap, "Cache augmented database management systems", In Proceedings of the ACM SIGMOD Workshop on Databases and Social Networks, ACM, **(2013)**, pp. 31-36.
[13] P. Gupta, N. Zeldovich and S. Madden, "A trigger-based middleware cache for orms", In Middleware 2011, Springer, **(2011)**, pp. 329-349.
[14] F. Dong, K. Ma and B. Yang, "Cache system for frequently updated data in the cloud", WSEAS Transactions on Computers, vol. 14, **(2015)**.
[15] F. Wolf, H. Betz, F. GropengieBer and K. U. Sattler, "Hibernating in the cloud-implementation and evaluation of object-nosql-mapping.", In BTW, Citeseer, **(2013)**, pp. 327-341.
[16] Dan R. K. P., Austin T. C., I. Zhang, S. Madden and B. Liskov, "Transactional consistency and automatic management in an application data cache", In OSDI, vol. 10, **(2010)**, pp. 1-15.
[17] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals", In Soviet physics doklady, vol. 10, **(1966)**, pp. 707.

Authors

Kun Ma, received his PhD degree in Computer Software and Theory from Shandong University, Jinan, Shandong, China, in 2011. He is a senior lecturer in Provincial Key Laboratory for Network based Intelligent Computing and School of Information Science and Engineering, University of Jinan, China. He has authored and coauthored over 30 research publications in peer-reviewed reputed journals and conference proceedings. He has served as the program committee member of various international conferences and reviewer for various international journals. He is the Co-Editor-in- Chief of International Journal of Computer Information Systems and Industrial Management Applications (IJCISIM). He is the managing editor of Journal of Information Assurance and Security (JIAS) and Information Assurance and Security Letters (IASL). He is the editorial board member of International Journal of Intelligent Systems Design and Computing and Journal of Software. He is the guest editor of International Journal of Grid and Utility Computing and Informatica. His research interests include model-driven engineering (MDE), data intensive computing, big data management, and multi-tenant techniques.

Zijie Tang, is an undergraduate student of Dr. Kun Ma, studying at School of Information Science and Engineering, University of Jinan. His research interests include software development of innovative applications, data intensive computing, and big data management.

Zhe Yang, is an undergraduate student of Dr. Kun Ma, studying at School of Information Science and Engineering, University of Jinan.

Shuwei Yao, is an undergraduate student of Dr. Kun Ma, studying at School of Information Science and Engineering, University of Jinan.