Using Memcached to Promote Unified User Management System

Zuliang Zhao, Xiaodong Zhang, Lin Li, Zhe Liu, De-Hai Zhu, and Shao-Ming Li

College of Information and Electrical Engineering, China Agricultural University, Beijing, China
mrzzl1987@gmail.com, zhangxd@cau.edu.cn, lilincau@126.com, liuzhe23@vip.qq.com, zhudehai@263.net, lshaoming@sina.com

Abstract. In modern society, more and more companies have introduced different systems in corporate business sectors, furthermore, an employee may work in various systems. As a result, the employees need to record multiple usernames and passwords, which brings discomfort and lots of likelihood of errors, and posts security risks for the different systems that have their own security user authentications. Meanwhile, most of management systems are based on B/S model, where one of the foremost requirements is to provide low-latency access to frequent and large amounts of requests. To address this needs, in this paper, we firstly demonstrate the design of a web unified user management system which based on the Role-based access control (RBAC) model and Memcached. Secondly we conducted an experiment on two groups of integrated prototype systems to validate our results. Finally we draw a conclusion than it can not only monitor users in real time and configure the permissions safely and flexibly, but also reduce the response time significantly and improve the system performance.

Keywords: Memcached, caching technology, unified user management.

1 Introduction

Nowadays, with the development of the internet and the popularization of computers, more and more staff accomplish their daily tasks in computer. Here is problem: on the one hand, the employee and corporate structures have become more and more complicated. For instance, the jobs change frequently and a plenty of High multi-tasters exist in different departments in a small business. These factors require that the user management need to be more flexible. Furthermore, it should not only be customized different permission to different users, but also achieve the goal that different staff members use different functions in the integrated enterprise management system. On the other hand, the concurrency and PV (page view) have increased dramatically in the Web-based enterprise management system. With the purpose of improving system response, in this paper [6], we have optimized the design of the traditional unified user management platform: by using the Memcached caching technology. In our architecture, the user management data which doesn't

D. Li and Y. Chen (Eds.): CCTA 2013, Part I, IFIP AICT 419, pp. 175–182, 2014.

change but needs to access frequently will be stored in the memory. While the web applications exchange data with memory directly. Therefore, the read rate increases by several orders of magnitude and the user experience (UE) will be enhanced.

1.1 Unified User Management Platform [9]

The traditional unified user management platform, which is based the Lightweight Directory Access Protocol (LDAP) or database [2], manages the storage, authentications and permissions of user information uniformly among the applications in the enterprise organization. Users will be able to use the resources and services which have been authorized simply with Single sign on (SSO) [7].

1.2 Memcached and .net Cache [10]

Memcached is a general-purpose distributed memory caching system that was originally developed by Danga Interactive for LiveJournal. It is often used to speed up dynamic database-driven websites by caching data and objects in RAM to reduce the number of reading times from an external data source (such as a database or API). Therefore, it can be used to improve the speed of dynamic Web applications and the system scalability.

In the development of Unified user management platform based on asp.net, it is found that these functions can be implemented by exploiting a cache class in .net. In most cases we can improve the performance by using the Cache class, but some bottlenecks may also be introduced: One thing is that it is difficult to specify the occupied memory according to cache classes. The other thing is that it is impossible to exchange the data in the cache of one computer with that of another computer. In some cases, the system managers need to do some security control in real time, e.g., making the login users log out the system, changing the user permissions in real-time and so on. However, these functions cannot be realized by using the Cache class. Therefore, we import the distributed cache in this paper, which can be regarded as an extension of the traditional concept of cache used in a single locale. A distributed cache may span multiple servers so that it can grow in size and in transactional capacity. As a consequence, it will significantly enhance the reusability and real-time control of cached data.

1.3 Role-Based Access Control (RBAC) System Model

In most of enterprises, roles are created for various job functions. [3]The permissions to perform certain operations are assigned to specific roles. Members of staff (or other system users) are assigned particular roles, and through those role assignments acquire the computer permissions to perform particular computer-system functions. Since users are not assigned permissions directly, but only acquire them through their role (or roles), management of individual user rights becomes a matter of simply assigning appropriate roles to the user's account; this simplifies common operations, such as adding a user, or changing a user's department.

2 System Design and Implementation

2.1 System Architecture

The system architecture is object-oriented, which is consistent with the core idea of MVC. The details of this architecture are described in the following:

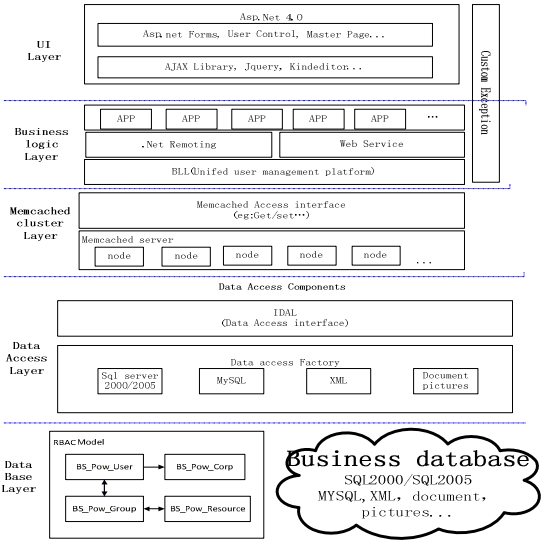

Fig. 1. System Framework

UI Layer: as illustrated in Fig. 1, it can be any output representation of data, such as a chart or a diagram. Note that some data can be represented by different views, such as a pie chart for management and a tabular view for accountants.

Controller Layer: mediates input, converting it to commands for the Memcached.

Memcached Layer: data input and request, converting it to commands for the controller or database access, Storage of the user manager data.

Database Access Layer: data input and request, converting it to commands for Memcached or data base.

Data Base Layer: storage of data.

2.2 Database Design

The database is explained in detail as follows:

BS_Pow_User table: user information, including users' personal information, username and password.

BS_Pow_Corp table: enterprise organization information by tree structure. We use department's bits (field: CorpNum) to confirm the department affiliation. This

structure is able to cope with most kinds of substantially requirements in different departments of enterprises.

BS_Pow_UserWithCorp table: user's department. In some cases, one user may belong to different departments at the same time. To solve this problem, we create a main department mark (filed: Main, value: "1" means main department; "null" means: subsidiary departments).

BS_Pow_Group table: role's information. System managers can define different roles by themselves.

BS_Pow_Resource table: we store data by tree structure. We define menus and operations as resources in platform. Menus include access paths to different systems, while operations include what users can do in different systems just as adding, deleting, changing, searching, input, output, printing and etc. They are distinguished with each other by the field "ResUrl" ("null" means it is operation and the other is resources).

BS_Pow_UserWithGroup table: relationships between users and roles.

BS_Pow_GroupWithResource table: relationships between roles and resources.

BS_Pow_LoginLog table: User's login information. System managers can check the users' login information such as IP, permissions from this table.

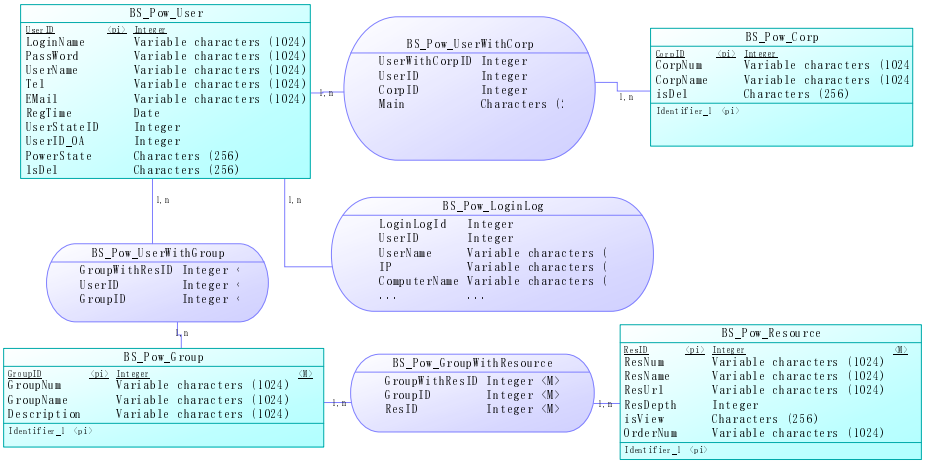

Fig. 2. Database Design [8]

2.3 Data Initialization in Memcached [13]

After Memcached configuration is completed, we need to copy a mirror file of the user-role-resource information from the database. First, we should make a serialization for the permission configuration data in the database. Serialization, represents the process of converting a data structure or object state into a format that can be stored and "resurrected" later in the same or another computer environment. When the resulting series of bits are reread according to the serialization format, it can be used to create a semantically identical clone of the original object.

Memcached supports custom data types which only require a [Serializable] mark when a new data type is defined. We create a user class; includes user information, departments, roles and role-resources. Memcached uses a key–value associative array. In this paper, we use user-id as the key and the user class instance as the value.

2.4 The Basic Process [5]

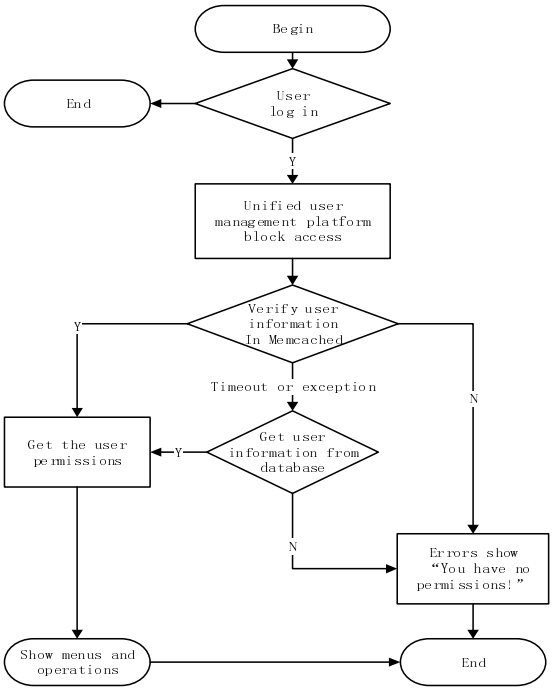

Fig. 3. The basic Process

2.5 Permission Data Read

When Memcached receives a get (incr/decr/...) request, it queries its own cache, or tries to access the database in case that the first query is missed. The algorithm runs as follows: Firstly get the data from database and post it to the user, secondly store it to Memcached's cache. If there is another post for the same key, then send this key-value back to the host. In order to prevent the system from the "Dos attacks" which works in exhaustive manner and may would result to database crash, we only try to connect the database until the Memcached is unable to connect or throw an exception.

2.6 Permission Date Modification

When Memcached receives a set (add/replace/...) request and has stored the new value to Memcached key-value successfully, it would be triggered to access the database, and then do an operation of deserialization to the changed key-values, finally store them to the database.

2.7 Memcached Persistence

Memcached runs as the Daemon model in one or more servers and accepts the connection operation of a client at any time. Since the whole data is stored in the memory, it would be lost whenever the servers rebooted. Therefore, a timer is needed to synchronous the data between the database and Memcached. The data would be lost anywhere from this. In this case, the data would not be lost even when the servers are crashed.

3 Performance Testing

Microsoft's Web Application Stress Tool, which is developed by Microsoft's website testers, provides an easy way to simulate large number of users against Web application. This tool makes it possible to make intelligent decisions about hardware and software load incurred by application and how much traffic a given machine or group of machines can handle.

3.1 Test Environment

Table 1. Test environment

Hardware environment	Software environment
Server1 CPU:XEON MP 7300, RAM 2GB	Sql2005 + iis6.0
Server2 CPU:XEON MP 7300, RAM 2GB	Sql2005 + iis6.0
Client Dual-Core 3.2*2, RAM 4GB	Microsoft Web Application Stress Tool

3.2 Test Plan

In this simulation, we consider the case when respectively 50,100 and 300 users concurrently log in, for the purpose of testing Memcached performance.

Data and server configuration instructions are described in the following: A Group Company is considered which is consisted of 6 subsidiaries, 627 departments and more than 2000 staff, including formals and temporaries. There is almost 10 MB user management data. The time that Memcached services take to start after the server reboots is Microsecond, as same as the time that Memcached completed a comparison with database.

Table 2. No Memcached (Unit: Ms)

users	average	median	90%line	min	max
50	1620	1754	1812	125	6412
100	6680	6789	7054	250	10121
300	12560	14576	15746	328	17123

Table 3. Using Memcached (Unit: Ms)

users	average	median	90%line	min	max
50	1220	1254	1312	64	3412
100	3680	3789	4054	201	6121
300	5960	6576	6746	300	9123

Illustrating:

average: Average response time: in default, is the average response time for a single request

median: Median, 50% of the user's response time
90% line: 90% of the user's response time

min: The minimum response time

max: The maximum response time

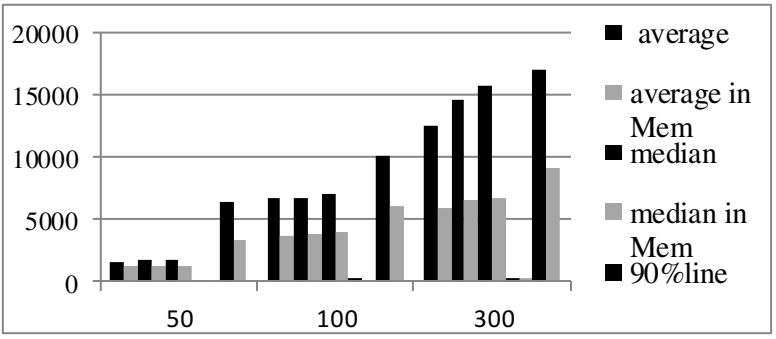

Fig. 4. Contrast Chart

Table 4. Response time and UE

Response time	UE
Within 2s	Speedy
2-5s	Accepted
5-10s	Anxiety, some users try to re-click
More than 10s	Believe system had crashed

4 Conclusions

In this paper, we optimize the traditional Unified User Management Model by using the Memcached cache technology. The results of performance test show that our proposed approach can significantly speed up the system operation and improve the performance, while the load of servers are substantially released. Moreover, our approach makes it possible to control and adjust the user permissions in real-time, which guarantees the security of the system.

Acknowledgment. This paper is supported by the "National Key Technology R&D Program under the Twelfth Five-Year plan of P.R. China" (Grant No. 2012BAK19B04-03).

References

1. Jose, J., Subramoni, H., Luo, M., Zhang, M., Huang, J., Wasiur-Rahman, M., Islam, N., Ouyang, X., Wang, H., Sur, S., Panda, D.K.: Memcached Design on High Performance RDMA Capable Interconnects. In: Proceedings of the 2011 International Conference on Parallel Processing, pp. 743–752. IEEE Computer Society, Washington, DC (2011)
2. Memcached Command Binary Protocol, http://code.google.com/p/memcached/wiki/BinaryProtocolRevamped
3. http://en.wikipedia.org/wiki/Role-based_access_control
4. Sobel, J.: Keeping Up (The Facebook Blog) (2011), http://blog.facebook.com/blog.php?post=7899307130
5. Memcached Commands, http://code.google.com/p/memcached/wiki/NewCommands
6. Memcached: High-Performance, Distributed Memory Object Caching System, http://memcached.org/
7. Appavoo, J., Waterland, A., Da Silva, E.A.: Providing a cloud network infrastructure on a supercomputer. In: Proceedings of the 19th ACM Int'l Symposium on High Performance Distributed Computing, HPDC 2010, pp. 385–394. ACM, New York (2010)
8. Vaidyanathan, K., Narravula, S., Balaji, P., Panda, D.K.: Designing Efficient Systems Services and Primitives for Next-Generation Data-Centers. In: Workshop on NSF Next Generation Software(NGS) Program; Held in conjunction with IPDPS (2007)
9. IBM, Message Passing Interface and Low-Level Application Programming Interface (LAPI), http://www-03.ibm.com/systems/software/parallel/index.html
10. Chalamalasetti, S.R., Lim, K., Wright, M., AuYoung, A., Ranganathan, P., Margala, M.: An FPGA memcached appliance. In: FPGA 2013 Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp. 245–254 (2013)
11. Jose, J., Subramoni, H., Kandalla, K., Wasi-ur-Rahman, M., Wang, H., Narravula, S., Panda, D.K.: Scalable Memcached Design for InfiniBand Clusters Using Hybrid Transports. In: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 236–243 (2012)
12. Li, F., Zhan, S., Li, L.: Research on using memcached in call center. In: 2011 International Conference on Computer Science and Network Technology (ICCSNT), December 24-26, vol. 3, pp. 1721–1723 (2011)
13. Xu, J., Zou, W.: Application Research of Memcached. Science Mosaic 7, 95–97 (2012) ISSN: 1671-4792