Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/090153

A Dependable Computing in Database for Managing Mass Connection Systems

*Chung-Yang Chen and Wen-Lung Tsai**

Department of Information Management, National Central University, No. 300, Jhongda Rd., Jhongli City, Taoyuan Country 32001, Taiwan

Received: 21 Apr. 2014, Revised: 22 Jul. 2014, Accepted: 23 Jul. 2014
Published online: 1 Jan. 2015

Abstract: Most operations in database (DB) is based on the deferred update replication technique, especially in distributed environments. In deferred update replication, transactions are executed on a single host and broadcast to other hosts at commit time. Upon delivering a transaction, each host confirms it to ensure a globally serializable execution. The mechanism is subject to CAP theorem in order to ensure data consistency and availability. This paper proposes a dependable cache query mechanism in distributed relational database (DRDB). The key insight of the cache query mechanism provides high performance, strong consistency and availability for mass connection systems. In addition to presenting the cache query mechanism, we elaborate its implementation, and provide an experiment and analysis of its performance.

Keywords: CAP theorem, cache query mechanism, DRDB, mass connection systems

1 Introduction

DB is a basic component of the information industry, its development has, at the very least, spanned more than 50 years [16]. For about 40 years, the bulk of DB was in Structured Query Language (SQL) language until the development of the relational database (RDB) model [6], which greatly simplified the difficulty of application design. Until the 21st century, DB technology kept up with demand because of the spread of high-speed networks, the low cost of storage hardware, which made data storage more convenient. However, DB does not just emphasis data storage [5] and processing stability but also looks forward to making good use of more resources to serve a wider range of applications. Fields in RDB must find a new way, because too many applications depend on their development.

In the most serious challenge for RBD, distributed DB is bound to determine the modern direction of the field. DRDB is complex, and building it is difficult. Therefore, any way that helps related system stakeholders conceive the tradeoffs involved in developing DRDB is beneficial. CAP theorem [1][7] is basically the tradeoff between consistency and availability in a distributed environment is a representative of the general tradeoff between correctness and instantaneity in an unreliable system. This concept that it is impossible for a system to achieve both properties has played a key role in dependable computing in distributed environments.

CAP stands for consistency, availability, and partition tolerance. A distributed system cannot achieve all of these three characteristics but, depending on the requirements of a project, two can be selected. Therefore, only CA system (consistent and highly available, but not partition-tolerant), CP system (consistent and partition-tolerant, but not highly available), and AP system (highly available and partition-tolerant, but not consistent) are possible [7]. For example, shows that Oracle RAC [10] represents traditional RDB, which selected consistency and availability and dropped partition tolerance. NoSQL DB [4], on the other hand, drops consistency to achieve partition tolerance and availability. We can learn through this theory that, in reality, the development of a distributed system is bound to demand choices, and such the tradeoff is actually no better than one another, given a comprehensive look at the full set of applications. Cognitive sciences demand a focus on the development of dependable research and technology.

* Corresponding author e-mail: tswelu@gmail.com

This paper would focus on a cache mechanism in DRDB, which is compatible with traditional RDB and uses CAP perspectives to increase its load capacity. Priority is given to consistency. The nature of the so-called consistency of RDB is ACID transaction security. ACID stands for atomic, consistent, isolated, and durable; these four characteristics must be balanced. This requirement is fairly common in ticket systems [8]. This paper aims to improve the performance of DB in order to help promoting ticket systems and achieve sustainable results.

2 The proposed cache query mechanism

In a distributed system, a cache mechanism [9][15] is very important. Because the distributed system architecture of the network environment, connections, and network communication costs produce high computation costs, a priority is placed on cost reduction methods that make the best use of cache mechanism to reduce unnecessary network communications requests.

Each cache mechanism in DB on the market only serves statements of the SELECT cacheable behavior for the obvious reason that the SELECT statement does not change the contents of the DB [11]. This paper addresses the technical challenges of the UPDATE cache. Intuitively, the UPDATE statement would seem to change the contents of the DB, so the cache is not meaningless. However, after careful study, it was seen that the UPDATE statement did not always cause DB content changes.

```
UPDATE table_name
SET { column_name = {expression } [, ...]}
[ WHERE condition ]
```

Fig. 1: Excerpt of UPDATE statement [14]

Fig. 1 contains the UPDATE statement in which the assignments in the SET clause is executed, changing the DB content. The key to determine whether the content of the SET has been executed is to establish that the condition of WHERE. When that condition is not established, the content of the SET would not be executed, and the DB would not be changed in any way, making its features cacheable. This paper studies and proposes its own cache mechanism for this kind of query in order to obtain better system performance.

In DB, the natural processing of information is the most important and time-consuming work. Clear queries can enable the cache mechanism to reduce unnecessary requirements for DB processing, leaving more resources to deal with the real work and ultimately resulting in higher service performance.

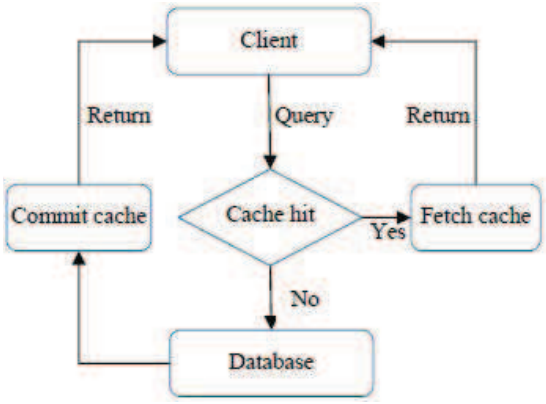

Fig. 2: Cache execution flow

The flow in Fig. 2 presents the most critical component, the commit cache, which determines what kind of results are in line with the conditions of the cache mechanism. Usually, these conditions are unchangeable and placed in the systems cache memory, so that the next time that query is made, the system quickly complete its response, reducing the expenditure of resources.

For the UPDATE statement, a logical ding of the condition when it is not cacheable is hard to achieve, because the SET problem is NP-complete. Therefore, the most efficient way to enter the DB is to actually execute one and, through the results of the implementation, judge whether to comply with the conditions of the cache mechanism.

The UPDATE statement is executed based on the standard SQL Front-end/Backend protocol [13], which, as we can see, in addition to the general system error, returns the contents of the packet format as follows: UPDATE rows

Where in the rows is a real number of lines changed, the only possible is when the rows = 0 alerts, the DB does not produce any changes, the DB does not produce changes, and the condition is not established. The results fed into the cache memory, keeping them available for the next time the same query is received.

Table 1: Comparison between cache on and cache off

	Cache on	Cache off	Ratio
UPDATE 0	1.85 sec	17.83 sec	10.38%
UPDATE 1	40.34 sec	37.51 sec	107.54%

The results of the prior implementation in Table 1 do not reflect the DB-level performance of the main test cachet. UPDATE 0, as the execution result, is not in the DB updated content, and as a result, UPDATE 1 is only the profile of an item, in order to minimize the cost of the disk write, so as not to interfere with the rating values.

The UPDATE 0 test comparison details the considerable savings of the resourcesan increase of nearly 90% in the resources availablewhich prove that the effectiveness of the cache is very significant. While the UPDATE 1 test must not be used for the cache, there would still be open caching inspection procedures, whose results reflect the cost of their implementation, representing 7.54%. These comprehensive comparison results indicate the application of the system to assess such situations and the suitability of applying this technology.

3 Experiment

3.1 Experiment subject: a ticket system

The ticket system obviously encounters the mass connection problem. DB can speed up the response to the requirements of the query. In addition to this technical problem, other practical considerations also exist.

3.1.1 Resource allocation [17]

This problem usually occurs when there is a large number of the number of instant connections that exceed the average load reserve of the unprepared system. Normality improves the overall amount of load to prepare, or permission to reduce the probability of the occurrence of this problem, but the cost of upgrading these resources needed to alleviate this fear is high.

3.1.2 Compatibility

In a purely commercial system such as the ticket system, any sales logic changes can cause a chain reaction, causing considerable distress to the system programmers. The DB can provide more methods to use and be compatible with the existing method.

Practically, there are a variety of different types of ticket systems, but for purposes of the DB, a defined-context, set-simulated system is the most critical operating mode. Overall, system design is also a critical factor. This paper tries to reduce the scope of the computation's application in order to understand the benefits of DB in this context. The first definition of the application of the system and the test scenarios are as follows:

–100 vacancies booked by 10,000 people
–Authentication violence into the DB and higher than practical environment strength
–Tickets that reserve a specific seat
–100 attempts allowed for each ticket; once exceeded, that seat ends
–Programming language, PHP [12]; web server, Apache HTTP Server [3]
–Client's simulated test software, for client ab [2]

Fig. 3 is the pseudo-code for the core program. According to the pseudo codethe following statement test should be used:
$ ab -n 200 -c 200 http://server-hostname/ticket.php
This statement can generate 200 simultaneous connections and produce test results. The statement changes the n -c parameter to complete the experiment.

```
$min_seat_id = 1;
$max_seat_id = 100;
$start_seat_id = rand(1, 100);
$user_id = "A random user name";
$psql = pg_connect("myDB with UPDATE query cache");
$sql_template = "UPDATE reserved SET user_id = '%s' "
              ."WHERE seat_id = %d and seat_id is not null";
$my_seat = false;

for($i = $min_seat_id; $i <= $max_seat_id; $i++) {
  $target_seat_id = ($start_seat_id + $i)
                  % ($max_seat_id - $min_seat_id + 1)
                  + $min_seat_id;
  $sql = sprintf($sql_template, $user_id, $target_seat_id);
  $result = pg_query($psql, $sql);
  if (pg_affected_rows($result) > 0) {
    $my_seat = $target_seat_id;
    break;
  }
}
if ($myseat) {
  print $user_id. " got ".$my_seat."\n";
} else {
  print $user_id. " can't get a seat.\n";
}
```

Fig. 3: Pseudo code of Simulated ticket system (ticket.php)

3.2 Result and discussion

Programs of the experiment and DB links use only a query statement to minimize the application process and meet the test requirements in this section for an UPDATE statement. When the application needs more time for processing, the density of work for DB is reduced and could provide even better work efficiency.

Table 2: Experiment result of simulated ticket system

Connection	Cache on	Cache off	Saved
100	3.34	2.54	-31.50%
200	5.99	7.18	16.57%
1000	23.57	42.58	44.65%
2000	45.67	89.77	49.13%
10000	262.33	455.39	42.39%

In Table 2, when there were only 100 connections, the Remarks cache mechanism failed to ensure that seats

were available and became a burden. Its contribution, however, increased along with the number of connections. In this test, stabilization saves more than 40% of the operating costs. Compared with Table 1, the pure DB test saves 90% of the cost, but in this test, sharpness is reduced to 40%. The difference is that this test increases the processing of the application. Even if the application of this test is very simple, it still accounts for high operating costs. Therefore, the practical design, not just the DB performance tuning and application design has a decisive impact on the overall efficiency of the implementation. Two of three must be implemented to ensure success.

4 Conclusion and future work

In related distributed systems, a cache mechanism is a crucial component. Through the proposed cache query cache mechanism for UPDATE, This paper challenges the practicality of implementing this technology, and the results successfully expanded the field of view of the development of basic technologies. Given its important position in the cache system, the cache must perform part of the job of the distributed system. To this end, our near future work will develop cache memory cluster, a faster and more stable DB cache mechanism.

For distributed DB, data distribution algorithms are a future major research focus. Consideration of the parameters of applications could support more efficient resource allocation and ultimately yield better results.

In addition, many of the applications deployed on APPs of smart phone have different properties than traditional application and database services like search engines and on-line shopping websites. For instance, they are heavily influenced by propinquity and security; they are organized around all kinks of social interactions; and privacy considerations are more immediate. By reviewing CAP in the context of DRDB, we might better understand the unique tradeoffs that occur in these types of scenarios.

Acknowledgement

The authors are grateful to the anonymous referee for a careful checking of the details and for helpful comments that improved this paper.

References

[1] D. Abadi, Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story, Computer, **45**, 37-42, 2012.

[2] The Apache Software Foundation, ab - Apache HTTP server benchmarking tool - Apache HTTP Server, http://httpd.apache.org/docs/2.2/programs/ab.html.

[3] The Apache Software Foundation, Welcome! - The Apache HTTP Server Project, http://httpd.apache.org/.

[4] E. Brewer, CAP twelve years later: How the ”rules” have changed, Computer, **45**, 23-29, 2012.

[5] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, Bigtable: A distributed storage system for structured data, ACM Transactions on Computer Systems, **26**, 4:1-4:26, 2008.

[6] E. F. Codd, A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, **13**, 377-387, 1970.

[7] S. Gilbert, N. Lynch, Perspectives on the CAP Theorem, Computer, **45**, 30-36, 2012.

[8] B. Gregg, Thinking Methodically about Performance, Communications of the ACM, **56**, 45-51, 2013.

[9] O. Khan, M. Lis, Y. Sinangil, and S. Devadas, DCC: A Dependable Cache Coherence Multicore Architecture, IEEE Computer Architecture Letters, **10**, 12-15, 2011.

[10] OracleR, Oracle Real Application Clusters, Available: http://www.oracle.com/us/products/database/options/real-application-clusters/overview/index.html, October 26, 2012.

[11] Pgpool Global Development Group, pgpool-II Tutrial On Memory Query Cache, Available: http://www.pgpool.net/pgpool-web/contrib_docs/memqcache/en.html, November 14, 2012.

[12] PHP Group, PHP: Hypertext Preprocessor, http://www.php.net/.

[13] The PostgreSQL Global Development Group, PostgreSQL: Documentation: 9.2: Message Formats, Available: http://www.postgresql.org/docs/9.2/static/protocol-message-formats.html, November 14, 2012.

[14] The PostgreSQL Global Development Group, PostgreSQL: Documentation: 9.2: UPDATE, Available: http://www.postgresql.org/docs/9.2/static/sql-update.html, November 14, 2012.

[15] M. H. Yeo, Y. S. Min, K. S. BOK, J. S. Yoo, The Optimization of In-Memory Space Partitioning Trees for Cache Utilization, IEICE TRANSACTIONS on Information and Systems, **E91-D**, 243-250 2008.

[16] Wikipedia, Database management system, Available: http://en.wikipedia.org/wiki/DBMS, October 25, 2012.

[17] W. Zhang, L. Ruan, M. Zhu, L. Xiao, J. Liu, X. Tang, Y. Mei, Y. Song, Y. Sun, SLA-Driven Adaptive Resource Allocation for Virtualized Servers, IEICE TRANSACTIONS on Information and Systems, **E95-D**, 2833-2843, 2012.

Chung-Yang Chen is an Associate Professor in the Department of Information at National Central University (NCU), Taiwan R.O.C. He received his Ph.D. degree in Industrial Engineering and Engineering Management at Arizona State University (ASU) in 2002. He is also the director of Computing Center in the School of Management. His current research and teaching interests include the areas of software engineering, BPR (Business Process Reengineering), CMMI, project management methods, and information quality.

Wen-Lung Tsai is a Ph.D. candidate of Information Management at National Central University (NCU), Taiwan R.O.C. He is also an engineer in Innovative DigiTech-Enabled Applications & Services Institute, Institute for Information Industry (III), Taiwan R.O.C. His research interests include cloud computing, distributed database, software engineering and project management.