Future Generation Computer Systems 74 (2017) 232–240

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Research and implementation of a distributed transaction processing middleware

Jianjiang Li [a], Qian Ge [a], Jie Wu [b], Yue Li [a,*], Xiaolei Yang [a], Zhanning Ma [a]

[a] Department of Computer Science and Technology, University of Science and Technology Beijing, China

[b] Department of Computer and Information Sciences, Temple University, USA

HIGHLIGHTS

- A middleware-level distributed system is complemented for improving the performance of transaction processing.
- By making partition extension to Berkeley DB, this paper overcomes the disadvantage of non-support parallel writing across multiple nodes.
- Monitoring nodes of the distributed database system by the middleware ensures correct execution and migration of transaction.

ARTICLE INFO

Article history:
Received 1 September 2015
Received in revised form
8 January 2016
Accepted 30 January 2016
Available online 8 February 2016

Keywords:
Distributed system
Transaction processing
Middleware
Partition replication body

ABSTRACT

Currently, increasingly transactional requests require high-performance transaction processing systems as support. The performance of a distributed transaction processing system is superior to that of traditional single-node transaction processing system, and the characteristic of multi-node determines that distributed transaction processing systems should pay more attention to availability. For example, in traditional single-node systems, the performance of Berkeley DB is high, but its shortcoming of not supporting parallel writing across multiple nodes is weakening its availability and scalability in the distributed environment. This paper has designed and implemented a middleware-level distributed transaction processing system called POST, including a distributed database system called POSTBOX which is based on Berkeley DB and data partition, and a distributed transaction processing middleware called POSTMAN. POSTBOX inherits the availability of highly available Berkeley DB, and expands it with data partition. By Partition Replication Body (PRB), POSTBOX overcomes the native weakness of highly available Berkeley DB, which indicates that highly available Berkeley DB does not support parallel writing across multiple nodes; POSTMAN is a middleware adapting PRB. POSTMAN monitors POSTBOX in real-time via Partition Replication Body State Array (PRBSA), and ensures the correctness of transaction processing and transactions migration in the case of node failure. The actual test results show that POST possesses high availability, and has an obvious improvement of write performance compared with highly available Berkeley DB.

1. Introduction

Historically, OnLine Transaction Processing (OLTP) [1] refers to submitting traditional transactions such as ordering goods or transferring payments to the OLTP system, based on Relational DataBase Management System (RDBMS). With the rapid development of Internet and Internet application, transaction occurs some changes, one of the most significant features of which is the explosive growth of transaction throughput [2]. For instance, excellent multi-user game based on the web can produce a large amount of interactions within one second, and the growth of smart phone use and other mobile terminals has given rise to the development of mobile transaction. These Internet applications produce more transaction requests than the capability of the traditional OLTP system, and it is difficult for RDBMS to deal with high concurrent transaction requests. In addition, RDBMS cannot support expansion without offline and distribution very well. For example, SQL query [3] of a table with massive records in a relational database management system will cost an amount of time. Although it can be solved by data segmentation and table segmentation, it also increases the difficulties of programming, data backup, database

* Corresponding author.
E-mail addresses: lijianjiang@ustb.edu.cn (J. Li), greenday0925@gmail.com (Q. Ge), jiewu@temple.edu (J. Wu), liyuepkoneal@outlook.com (Y. Li), chinayangxiaolei@163.com (X. Yang), ningzhanma@163.com (Z. Ma).

http://dx.doi.org/10.1016/j.future.2016.01.021
0167-739X/

expansion and some other issues. In order to enhance the performance of system, the most direct solution is to purchase a machine which has stronger performance, but its higher cost is often prohibitive for most enterprises.

Distributing data and loading it to multiple nodes by using distributed database systems [4,5] is an effective method to improve the performance of the transaction processing system. Currently, application and deployment of a distributed system are becoming known more and more widely, especially in the background of expansion of cloud computing and big data [6,7]. Cloud computing [8,9] requires using low-cost servers instead of expensive machines as a hardware infrastructure platform, and obtains high availability and scalability through redundancy between nodes. Berkeley DB [10] is a powerful key/value database engine: its high availability version (referred to highly available Berkeley DB) provides a distributed database solution based on master–slave replication, with high availability and better reading scalability. Berkeley DB provides full ACID [11] transactional guarantees. This ensures that highly available Berkeley DB can be applied not only to lower requirements for data consistency (for example, state updates of social network users do not need to immediately synchronize to the entire application), but also to higher requirements for data consistency, such as financial systems or order processing systems, because these systems are intolerable to abandoning transaction and data consistency [12].

Highly available Berkeley DB supports high availability and read scalability, it does not have write scalability, that is to say, it does not support parallel write cross multiple nodes. In addition, compared to building a centralized or client/server system, it is quite difficult to build a truly distributed database system, because distributed database systems may have multi-node failures and problems with security of data storing [13,14], inter-node communication is relatively complex. Middleware is an effective way to solve the fault tolerance of distributed database systems, communication difficulties, and other problems. An effective distributed transaction processing middleware [15–18] can effectively manage a distributed database system, and reduce the programming difficulties.

In response to these problems, this paper designs and implements a distributed transaction processing middleware system called POST, which consists of a distributed database system called POSTBOX based on Berkeley DB and data partition, and a distributed transaction processing middleware called POSTMAN. POSTBOX makes partition extension to highly available Berkeley DB, and overcomes the problem that Berkeley DB does not support parallel write cross multiple nodes via Partition Replication Body (PRB). POSTMAN, which is deployed on top of POSTBOX, and fully adapted to the PRB of POSTBOX, can provide an access interface for the interaction application and POSTBOX. POSTMAN monitors the status of each node of POSTBOX by Partition Replication Body State Array (PRBSA), and ensures correct execution and migration of transaction when a node fails through an efficient scheduling mechanism.

The rest of this paper is organized as follows. Section 2 describes the highly available of Berkeley DB. Section 3 describes the system architecture of POST, and introduces the distributed database system called POSTBOX based on Berkeley DB and data partition, and distributed transaction processing middleware called POSTMAN. Section 4 provides analysis of the availability and performance of the POST system. Section 5 provides experimental results and analyses. Section 6 introduces related work. Finally, this paper makes a summary in Section 7.

2. High availability of Berkeley DB

Berkeley DB achieves high availability by the replication group. Replication group is a collection of Berkeley DB environments

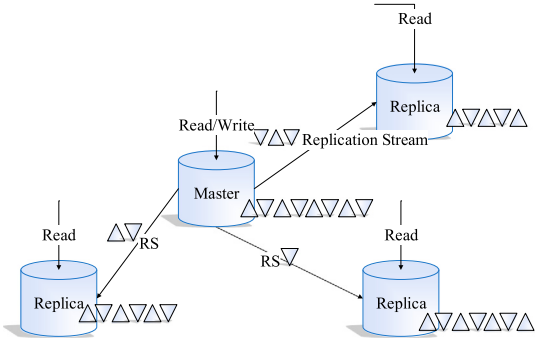

Fig. 1. Replication stream of highly available Berkeley DB.

distributed on different physical nodes. Nodes in replication group have the following three states:

(1) Master: "Master node" is chosen by a simple majority of electable nodes. It can process both read and write transactions.
(2) Replica: "Replica node" is in communication with a Master node via a replication stream which is used to keep track of changes made at the Master node. It only supports read transactions.
(3) Detached: "Detached node" has been shut down. It is still a member of the replication group, but is not an active node. A node that is not in the detached state is also referred to as being active.

There is only one Master node in the replication group. The Master node can read and write data, and the Replica node can only read data. The Master node and Replica node both belong to active nodes in the replication group. One node in a detached state means that this node is not an active node, and it is still a member of a replication group. The following introduces a replication group in two aspects of the replication stream and generation mechanism of the Master node.

2.1. Replication stream

Write transactions executed on a Master node are replicated to Replica nodes by a logical replication stream established on a TCP/IP connection. This connection is a dedicated connection between a Master node and each Replica node. Replication stream contains some descriptions of the logical changes. These logical operations are computed from log entries of the current Master node and are replayed at each Replica node by an efficient internal replay mechanism.

As shown in Fig. 1, there are four nodes in a replication group, including a Master node and three Replica nodes. The Master node has completed eight write operations (the log entry is represented by a white and gray triangle), and because the progress of the three Replica nodes is behind the Master node, the Master node sends log entries to all Replica nodes by a replication stream (dashed line with an arrow), so that all Replica nodes can replay according to the log entry to keep up with the progress of the Master node. In a replication group, Replica nodes are distributed to multiple physical machines, so this can ensure that a single failure cannot affect other nodes.

2.2. The generation process of master node

In a replication group, only the Master node has write access. When it is shut down, it is essential for the replication group to regenerate a new Master node. The generation of a Master node is influenced by two factors: the log progress and the priority of the

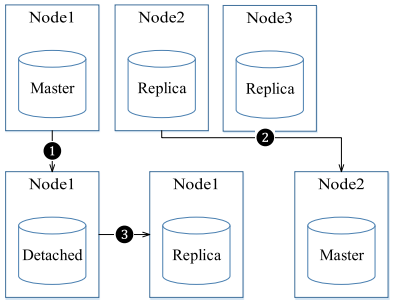

Fig. 2. The generation process of Master node of highly available Berkeley DB.

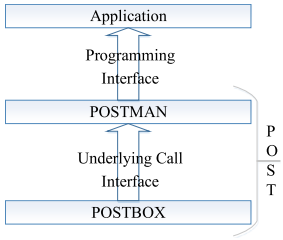

Fig. 3. System architecture of POST.

Replica nodes. A replication group generates a new Master node by comparing the priority of nodes (highly available Berkeley DB stipulates that priority is a non-negative integer. The node with the maximum priority will become the Master node, and 0 indicates that the node does not participate in the election). Shown in Fig. 2, the solid line with the arrow indicates a change of node status (replication group uses priority layout $[3 \quad 2 \quad 1]$, which means that node 1 has the highest priority 3).

(1) In the initial state, node 1 is the Master node. When node 1 is shut down, it falls into a separated state.
(2) The replication group elects a new Master node from the remaining two active nodes (node 2 and node 3). Because the priority of node 2 is greater than that of node 3, node 2 becomes the new Master node.
(3) After the recovery of node 1, the replication group already has a Master node, so that node 1 will not preempt the master status of node 2; even if the priority of node 1 is greater than that of node 2, node 1 can only become a Replica node.

Berkeley DB stipulates that when the Master node is shut down, the number of Replica nodes participating in the election must be greater than or equal to two. That is to say, for a replication group with N nodes, when $N-1$ nodes including the Master node are shut down simultaneously, the only remaining node cannot become the Master node (the replication group loses write permission).

3. POST

As shown in Fig. 3, POST is composed of distributed database system called POSTBOX based on highly available Berkeley DB and data partition, and distributed transaction processing middleware called POSTMAN. POSTMAN provides an application programming interface for the application program, and POSTBOX provides the underlying call interface for POSTMAN.

3.1. POSTBOX

POSTBOX makes use of data partitioning to expand the highly available Berkeley DB, and firstly introduces the concept of Partition Replication Body (PRB). PRB overcomes the weakness that highly available Berkeley DB does not support parallel write across

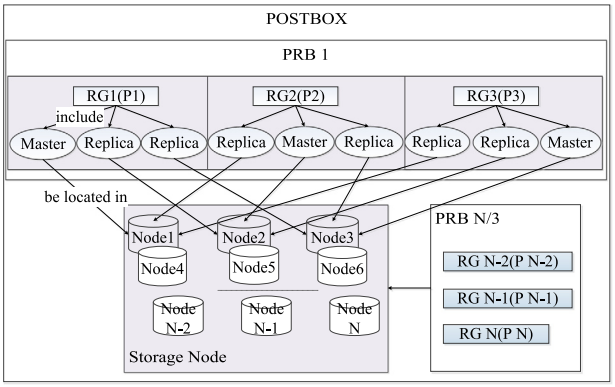

Fig. 4. System structure of POSTBOX.

multiple nodes, thereby improving the write performance of this system.

Data partition is not only the theoretical basis of the implementation of distributed data storage and distributed transaction processing, but also an important means to improve transaction processing performance. POSTBOX uses formula (1) to store *key-value* pairs on different nodes based on the hash of keys. All nodes are numbered from 1 to N, and *key-value* pairs with the same partition number are divided into the same partition, and are stored on the node with the corresponding number. POSTBOX forms the structure with PRB as a basic unit as shown in Fig. 4 by constructing a replication group for each data partition. The following will focus on PRB.

$$PartitionNum = (Hash(key) \bmod N) + 1. \tag{1}$$

In formula (1), *key* is key data. N is POSTBOX nodes (partitions), and *PartitionNum* is key corresponding partition number.

PRB is the integration of data partition and replication group, it is a two-dimensional concept, it has the horizontal physical scalability of replication group, and achieves a logical extension in the vertical direction by data partition, that is to say, multiple replication groups run within a set of nodes, and each replication group only assumes data belongs to a certain partition. PRB is the basic unit of the POSTBOX. As shown in Fig. 4, POSTBOX uses the 3×3 PRB, that is to say, PRB is composed of three nodes, on which three replication groups are responsible for the data storage of three partitions that are run, and POSTBOX stipulates that replication group N is responsible for storing the data of partition N.

Now firstly introduce two key concepts of PRB: Node State Vector (*NSV*) and PRB State Matrix (*PRBSM*). *NSV* is the concept of storage node of POSTBOX, that is, each storage node of POSTBOX maintains a column vector NSV_n (n is the node number) of length 3, which identifies the relationship between nodes and three replication groups. *PRBSM* is the concept of POSTMAN, which is one of the 3×3 matrices. If PRB m contains storage node i, j, k, its *PRBSM* conforms to formula (2).

$$PRBSM_m = [NSV_i \; NSV_j \; NSV_k]. \tag{2}$$

In formula (2), m is the number of PRB. NSV_i, NSV_j, NSV_k represent *NSV* of nodes contained by "PRB m" respectively.

As shown in Fig. 4, "PRB 1", which is responsible for storing data of partitions 1, 2, and 3, is composed of nodes 1, 2, and 3 and includes replication groups 1, 2, and 3. In POSTBOX, PRB uses Priority Matrix $PM = \begin{bmatrix} 3 & 2 & 1 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{bmatrix}$, so in the initial state, PRB 1 is satisfied with $PRBSM_1 = \begin{bmatrix} M & R & R \\ R & M & R \\ R & R & M \end{bmatrix}$, $NSV_1 = [M \quad R \quad R]^T$, $NSV_2 = [R \quad M \quad R]^T$, and $NSV_3 = [R \quad R \quad M]^T$. To facilitate the presentation, this paper stipulates that "1" indicates the state of

master-"M", and "0" indicates the state of replica-"R", "\" indicates node failure of NSV field, and "−1" indicates node failure of *PRBSM* field. For "PRB 1", in the initial state, Node 1 is the Master node of replication group 1 (with read and write permissions to data of partition 1), and Replica node of replication groups 2, and 3 (with read access to data of partition 2 and 3), the corresponding $NSV_1 = [1 \quad 0 \quad 0]^T$; Node 2 is the Master node of replication group 2 (with read and write permissions to data of partition 2), and Replica node of replication groups 1, and 3 (with read permission to data of partition 1 and 3), the corresponding $NSV_2 = [0 \quad 1 \quad 0]^T$; Node 3 is the Master node of replication group 3 (with read and write permissions to data of partition 3), and Replica node of replication groups 1, and 2 (with read permission to data of partition 1 and 2), the corresponding $NSV_3 = [0 \quad 0 \quad 1]^T$, so $PRBSM_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, which is generated by NSV_1, NSV_2 and NSV_3. More complex information regarding NSV and PRBSM will be described in detail in subsequent sections.

3.2. POSTMAN

POSTMAN mainly achieves routing based on monitor and POSTMAN transaction processing. The function of the monitor is the tracking state of each PRB in POSTBOX. According to the state information, the routing process can route write requests to correct nodes in POSTBOX. The monitor needs to monitor the following two types of state:

(1) Changes of Master node of each replication group in POSTBOX.
(2) Changes of node status of each replication group in POSTBOX, that is, whether the node is an active node.

The monitor mainly monitors the status of POSTBOX by N-dimensional $PRBSA[N]$, N is the node number (partition number) of POSTBOX, the array index is associated with the partition, the corresponding element value is associated with the node, its meaning can be described by formula (3), which means that the node *NodeNum* has write permission of partition *PartitionNum*, the *PRBSA* uses "−1" to indicate that the PRB loses write permission of a partition.

$$NodeNum = PRBSA[PartitionNum - 1]. \tag{3}$$

In formula (3), *PartitionNum* is the partition number, and *NodeNum* is the node number.

PRBSA is generated with unit of PRB (described in Algorithm 1), for the node i:

- If i cannot be divided by 3, then this represents that node scanning does not reach the boundary of PRB. At this point, it only needs to determine the state of the ith node, and if the ith node is normal, then it will get normal *NSV* from the ith node, otherwise it will get a failure from the ith node.
- If i can be divided by 3, then this represents that node scanning has reached the boundary of PRB. At this point, firstly determine the state of the ith node, and if the ith node is normal, then it will get normal *NSV* from the ith node, otherwise it will get failure $NSV = [-1 \quad -1 \quad -1]^T$ from the ith node; then assemble *PRBSM* with the corresponding *NSV* in current PRB, and traverse *PRBSM* by row vector mode. For each row vector, if there is element "1" in it, then a corresponding *PRBSA* value is assigned to the node number of element "1" in the row vector; otherwise a corresponding *PRBSA* value is assigned to "−1".

PRBSM and *PRBSA* satisfy the following property:

- When the failure node number is not greater than 1 in the PRB, there always is three "1" in the corresponding *PRBSM*, which means that PRB obtains write permission three partitions.

Algorithm 1 The Generation Process of *PRBSA*

Input:
nodenum_from, *nodenum_to*, *PRBSA*
Output:
the assigned *PRBSA*

```
1: Initialize 3 × 3 empty matrix M = [C1, C2, C3] for storing PRBSM
2: GetNSV(i) gets NSV from the ith storage node in POSTBOX
3: for all nodenum_from ≤ i ≤ nodenum_to do
4:   if i%3 = 0 then
5:     if the ith node is normal then
6:       C3 ⇐ GetNSV(i)
7:     else
8:       C3 ⇐ [−1 −1 −1]^T
9:     traverse M = [R1 R2 R3]^T by the row vector mode
10:    for all 1 ≤ j ≤ 3 do
11:      if exist element whose value is 1 in Rj then
12:        PRBSA[(i/3 − 1) × 3 + j − 1] ⇐ i/3 + r
13:        /* r represents that element with value of 1is located
           in the rth row of matrix M */
14:      else
15:        PRBSA[(i/3 − 1) × 3 + j − 1] ⇐ −1
16:  else
17:    if the ith node is normal then
18:      Ci%3 ⇐ GetNSV(i)
19:    else
20:      Ci%3 ⇐ [−1 −1 −1]^T
```

- There is one "1" at most in a row in *PRBSM*, which means that the write permission can only be owned by a node in PRB.
- *PRBSM* can have more than one "1" in a column, which means that a node can have write access to multiple partitions in PRB.
- If "−1" appears in the corresponding *PRBSA* of PRB, there must be two or three "−1". This means that the 3×3 PRB can tolerate one node failure. Only when two or three node failures occur, the PRB will be invalid, and the corresponding *PRBSA* appears "−1".

Monitor maintains *PRBSA* by the way of updating according to demand. As shown in Algorithm 2, the process is divided into following two steps (node number i starts from 1 to N):

(1) Determine whether current node number i is greater than N. If not, continue to step 2, otherwise set node number to 1.
(2) Monitor detects all storage nodes of POSTBOX by probe (probe does not do any actual data operation, and it is only responsible for detecting the state of storage node in POSTBOX). If current node fails, only call the *PRBSA* generation algorithm to complete partial updates of *PRBSA*, and skip all the nodes of the PRB which current node belongs to, then detect the first node of next PRB. When the failed node number is multiple of 3, call form is *Generate_PRBSA*(i-2, i, *PRBSA*); Otherwise call form is *Generate_PRBSA*($(i/3) \times 3 + 1$, $(i/3 + 1) \times 3$, *PRBSA*). If the node is normal, then detect next node.

3.3. POSTMAN transaction processing

This paper mainly involves three related transaction concepts: atomic transaction, Single Partition Transaction (SPT), and POSTMAN transaction. The atomic transaction refers to one add, delete or modify operation to a key/value pair in Berkeley DB. SPT consists of one or more atomic transactions, and data operated by atomic transaction belongs to a partition. The transactional property of SPT is guaranteed by the Berkeley DB. POSTMAN transaction consists of one or multiple atomic transactions which may belong to different

Algorithm 2 Maintenance of *PRBSA*

Input:
i: node number

```
1: loop
2:   if i>N then
3:     i ⇐ 1
4:   else if Node i is failure then
5:     1) call partial generation algorithm of PRBSA for PRB
       including the ith node
       2) i ⇐ the number of first node of next PRB
6:   else
7:     i ⇐ i + 1
```

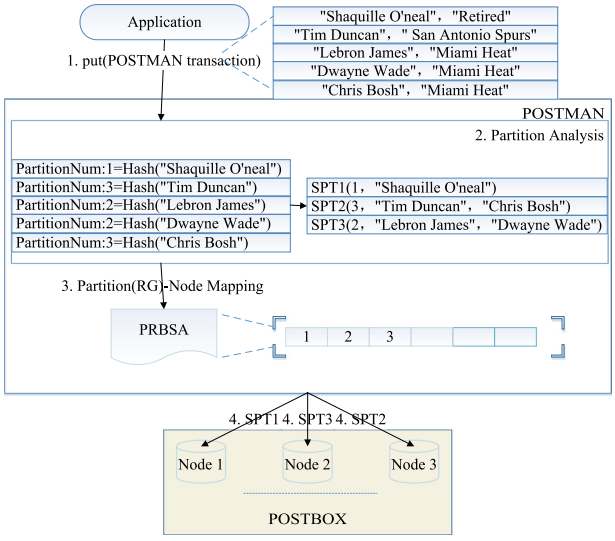

Fig. 5. POSTMAN transaction processing.

partitions. The transactional property of POSTMAN transaction is guaranteed by POSTMAN.

POSTMAN transaction highly depends on *PRBSA*. This mainly reflects in two aspects:

(1) By accessing *PRBSA*, POSTMAN is able to ensure that all SPTs constituting the POSTMAN transaction are sent to the correct node under normal circumstances.
(2) By accessing *PRBSA*, POSTMAN is able to ensure that SPT, which the failure node is responsible for, is correctly migrated to the other node when a node fails.

POSTMAN transaction process is divided into four steps (as shown in Fig. 5):

(1) Build POSTMAN transaction. The application builds POSTMAN transaction which contains five atomic transactions. The five atomic transactions insert key/value pair:
$\langle Shaquille\ O'neal|Retired\rangle$ $\langle Tim\ Duncan|San\ Antonio\ Spurs\rangle$
$\langle Lebron\ James|Miami\ Heat\rangle$ $\langle Dwayne\ Wade|Miami\ Heat\rangle$
$\langle Chris\ Bosh|Miami\ Heat\rangle$.
(2) POSTMAN makes partition analysis of POSTMAN transaction. Parser hashes the key of each atomic transaction constituting the POSTMAN transaction, then gets the partition corresponding to atomic transaction, and according to the partition, it builds the single partition transaction:
$SPT_3\ \langle 2|Lebron\ James|Dwayne\ Wade\rangle$
$SPT_2\ \langle 3|Tim\ Duncan|Chris\ Bosh\rangle$ $SPT_1\ \langle 1|Shaquille\ O'neal\rangle$.
(3) POSTMAN inquires *PRBSA* to determine the corresponding Master node of partition. Depending on the nature of *PRBSA*, the Master node of partition 1 is node 1, the Master node of partition 2 is node 2, and the Master node of partition 3 is node 3.
(4) POSTMAN sends SPT_1, SPT_2 and SPT_3 to POSTBOX to complete the insertion of data.

Algorithm 3 POSTMAN transaction submission

Input: *Threshold*

```
1: Loop_Count ⇐ 0
2: if Loop_Count > Threshold then
3:   rollback
4: else if PRBSA of SPT contains -1 then
5:   rollback
6: else
7:   SPTs are transmitted to the corresponding node to execute
8: if execution of the corresponding node is failure then
9:   1) Loop_Count ⇐ Loop_Count + 1
       2) Filter losing SPT
       go to Line 2
10: else
11:   submit
```

POSTMAN transaction processing improves the classic two-phase submission by loop detection and failure single-partition transaction filtering. It mainly improves the preparation phase of two-phase submission to avoid the rollback of the entire distributed transaction when a child transaction of distributed transaction fails. POSTBOX has a certain tolerance on node failure, 3×3 PRB can tolerate a node failure, which ensures that POSTMAN can complete the migration of failure single-partition transaction by loop detection when a node fails.

As shown in Algorithm 3, the loop detection refers to that POSTMAN sets up a loop counter – *Loop_Count* and a loop number threshold – *Threshold* for the preparation phase of the POSTMAN transaction. If a node fails during the preparation phase of the POSTMAN transaction, *Loop_Count* adds 1. Failure single partition transaction filtering indicates only filtering the single partition transaction executed on the failure node, and migrating it to the new node to execute in next preparation phase.

The preparation phase of POSTMAN transaction has two termination conditions, and meeting any one will lead to POSTMAN transaction rollback. The first is that *Loop_Count* is greater than *Threshold*, the second is that the values in *PRBSA* exist -1. The second condition is a forced termination condition, because the occurrence of the second condition indicates a failure of PRB, and the system loses write permission to the corresponding partition; the first condition is the optional termination condition, and it is primarily for the situation such as that the POSTMAN transaction consists of *N* single partition transactions, which belong to *N* PRBs (does not exist that any two single-partition transactions belong to a PRB). Assume that the value of *N* is 10, then the preparation phase of this POSTMAN transaction can tolerate up to 10 times the node failures and still continues to perform (without considering node recovery). This paper considers that this situation should be avoided, because the system does not have high availability.

4. Analysis of POST

CAP [19,20] was proposed by Brewer. As shown in Fig. 6, CAP theory suggests that, any distributed system cannot satisfy Availability, Consistency and Partition Tolerance at the same time. When improving any two of them, the third must be sacrificed. POST follows the CAP theory, meets the high availability and partition tolerance, and supports eventual consistency. This consistency model does not guarantee that all Replica nodes are consistent at the moment when the Master node submits. However, it is

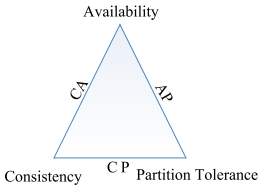

Fig. 6. CAP theory.

possible to ensure that if without generating a new update in a sufficiently long period of time, all the updates are propagated to all member nodes, and eventually all Replica nodes share a consistent view with the Master node. The following focuses on the analysis of POSTBOX availability and performance comparison of POST and highly available Berkeley DB.

4.1. Analysis of POSTBOX availability

During the execution of a distributed transaction, there are two situations which can cancel the transaction: deterministic case and non-deterministic case. Node failure is a typical non-deterministic case, because when a distributed database system is running, the node failure is random, and the user cannot determine which node will have problems. The deterministic case is usually controlled by the transaction logic, such as in an order management process; when the stock is less than zero, the transaction will be forced to cancel. POSTBOX availability is mainly for the non-deterministic case. PRB is the basic unit of high availability of POSTBOX. Availability of PRB reflects the availability of POSTBOX. PRB has the following properties:

(1) The number of failure node which PRB can tolerate has a linear growth as the expansion of scale of PRB. $N \times N$ PRB can tolerate $N - 2$ node failures.
(2) The availability of PRB has a monotonous growth as the expansion of scale of PRB.
(3) The growth of the availability of PRB decreases as the expansion of scale of PRB, which means that with the expansion of scale of PRB, the availability benefits brought by the unit expansion of scale of PRB gradually decrease.
(4) Copy number of $N \times N$ PRB is $N - 1$, that is, with the expansion of scale of PRB, copy number has a linear growth.

It can be obtained from the above properties that the improvement of availability will reduce system performance, because copy operation will take up a lot of CPU and I/O resources. In this paper POSTBOX uses 3×3 PRB, due to consideration of the system availability benefits and performance loss.

The following is the proof of properties 2 and 3:

Assume that the probability of node failure in POSTBOX is P (P is a decimal and is much smaller than 1. Currently it can generally be guaranteed to be 0.001), then according to the property 1, the availability of $N \times N$ PRB is $f(N)$, which satisfies

$$f(N) = 1 - C_N^{N-1} \times P^{N-1} \times (1 - P) - P^N; \quad N \geq 3.$$

That is, the case in which there are N nodes failures and any $N - 1$ nodes failures are excluded.

The availability of $(N + 1) \times (N + 1)$ PRB is $f(N + 1)$, which satisfies

$$f(N+1) = 1 - C_{N+1}^{N} \times P^N \times (1 - P) - P^{N+1}; \quad N \geq 3$$

$$\begin{aligned} f(N+1) - f(N) &= C_N^{N-1} \times P^{N-1} \times (1 - P) \\ &\quad + P^N - C_{N+1}^{N} \times P^N \times (1 - P) - P^{N+1} \\ &= P^N \times (1 - P) + N \times P^{N-1} \times (1 - P) \\ &\quad - (N + 1) \times P^N \times (1 - P) \\ &= P^N \times (1 - P) + P^{N-1} \times (1 - P) \times (N - (N + 1) \times P). \end{aligned}$$

$$\text{Set } \alpha = P^N \times (1 - P), \quad \beta = P^{N-1} \times (1 - P), \quad \text{and} \quad \gamma = N - (N + 1) \times P.$$

Then $\alpha > 0, \beta > 0$.

P is much smaller than $N/(N + 1)$ and greater than 0, so $\gamma > 0$.

Therefore, $f(N + 1) - f(N) = \alpha + \beta \times \gamma > 0$.

That is the availability of PRB has a monotonous growth as the expansion of scale of PRB.

$$\text{Set } F_N = f(N+1) - f(N), \qquad \delta = N + 1 - (N + 2) \times P,$$

$$\begin{aligned} \text{then } F_{N+1} &= f(N+2) - f(N+1) \\ &= P^{N+1} \times (1 - P) + P^N \times (1 - P) \\ &\quad \times ((N + 1) - (N + 2) \times P) \\ &= P \times (\alpha + \beta \times \delta), \end{aligned}$$

$$\begin{aligned} F_{N+1}/F_N &= P \times (\alpha + \beta \times \delta)/(\alpha + \beta \times \gamma) \\ &= P \times (\alpha/\beta + \delta)/(\alpha/\beta + \gamma) \\ &= P \times (P + \delta)/(P + \gamma) \\ &= [P^2 + P \times (\gamma + 1 - P)]/(P + \gamma) \\ &= (P + P \times \gamma)/(P + \gamma) < 1. \end{aligned}$$

That is, the growth scale of the availability of PRB decreases as the scale of PRB expands.

Assuming node failure probability $P = 0.001$ (the mean time between failures is 3 years, and the mean time to restoration is one day) in POSTBOX, then when $N = 3$, the availability of PRB $f(3) = 99.9997\%$ (this is a very high level availability), and the copy number is 2; when $N = 4$, the availability of PRB $f(4) = 99.9999996\%$, and the copy number is 3. Compared 3×3 PRB, the availability of 4×4 PRB only improves 0.0003%, but the copy number has increased by 50%. When N continues to increase, the copy number increases linearly, while the growth of the availability of PRB is almost negligible. The negative performance cost caused by the expansion of scale of PRB is much larger than the positive revenue generated by availability improvement. This article is sufficient to ensure the system gets higher performance at a better level of availability by using 3×3 PRB.

4.2. Analysis of POST performance

There are three main factors limiting the performance of POST: I/O of storage system, memory and processor, and communication between nodes. For the system related to database, the most obvious feature is that I/O is the key factor to determine the performance, so this section focuses on I/O. Here consider shared storage, shared storage indicates that all nodes of a system are mounted on a storage system, sharing I/O bandwidth of the storage system.

It takes T_{RG} for the time of highly available Berkeley DB completing N POSTMAN transaction, and T_{PRB} for the time of POST completing N POSTMAN transaction. Here we only consider the preparation phase (time consumption of the commit phase is relatively smaller than that of the preparation phase). For highly available Berkeley DB, each POSTMAN transaction will launch twice the local calls and the write operation once. For POST, the transaction is equally allocated to three nodes. Each node bears 1/3 the transactions, and data storage is well-distributed, so there will be 1/3 the transactions which initiates twice the local calls and one write operation, and 2/3 the transactions which initiates twice the remote calls and one write operation.

When I/O of the storage system is not restricted:

$$T_{RG} = N \times (2 \times T_{route\_local} + T_{write})$$

$$\begin{aligned} T_{PRB} &= N/3 \times 1/3 \times (2 \times T_{route\_local} + T_{write}) \\ &\quad + N/3 \times 2/3 \times (2 \times T_{route\_remote} + T_{write}) \end{aligned}$$

$$\begin{aligned} T_{RG}/T_{PRB} &= 3 \times (6 \times T_{route\_local} + 3 \times T_{write})/ \\ &\quad \times (2 \times T_{route\_local} + 4 \times T_{route\_remote} + 3 \times T_{write}). \end{aligned}$$

T_{route_local}: Time cost of routing the POSTMAN transaction from the local POSTMAN router to the local POSTBOX node

Table 1
Hardware configuration of experiment platform.

Hardware	Configuration
Processor	Two Intel Xeon X5670 (Six core), Frequency 2.93 GHz, 12 MB cache
Memory	32 GB
Disk	Share 6 TB
Network	InfiniBand

Table 2
Software version of experiment platform.

Software	Version
OS	RedHat Enterprise Linux 5.5 x86_64
Berkeley DB	Berkeley DB 11gR2 5.3.15

T_{route_remote}: Time cost of routing the POSTMAN transaction from the local POSTMAN router to the remote POSTBOX node

T_{write}: Time cost of the write operation of the POSTMAN transaction. Because T_{route_local} is smaller than T_{route_remote}; T_{route_local} and T_{route_remote} are smaller than T_{write}, when I/O of storage system is not restricted, $1 < T_1/T_2 < 3$, that is, compared to highly available Berkeley DB, POST gets an obvious performance improvement, but cannot reach linear expansion.

When I/O of the storage system is restricted, POSTMAN transactions initiated by all nodes of POST will line up in the storage system, and POST degenerates into highly available Berkeley DB. Because the scheduling capability of three nodes is better than that of a single node, the performance of POST will be better than that of a highly available Berkeley DB.

5. Evaluation

Write performance of POST (POSTBOX contains a PRB) and highly available Berkeley DB are compared in the experiment.

5.1. Experiment platform

Experiment platform is a high performance cluster. The experiment uses three nodes with the same configuration. Specific hardware configuration and software versions are shown in Tables 1 and 2.

5.2. Experimental method

The experiment uses POSTMAN transaction with the length of 10, that is to say, every POSTMAN transaction contains 10 atomic transactions. Every Atomic transaction completes fixed length transactional insertion operation, and each record consists of keys and fixed length values which are generated randomly and are not duplicated. Distinct key can ensure that each transactional insertion completed by atomic transaction is new, rather than transactional update operation (The actual test results show that time of update operation is much less than that of the insert operation). The data is divided into two types of 128 bytes and 512 bytes, which is used to test different performances of POST and highly available Berkeley DB under different I/O loads. At the same time, the experiment adopts four kinds of thread size (6, 9, 12, 15), which are used to test different performances of POST and highly available Berkeley DB under different CPU loads.

The experiment conducted 8 groups corresponding to four kinds of thread size and two kinds of data size, and each group performs 10 times, then gets the average. For certain thread scales M and data scales N, highly available Berkeley DB creates M threads for the Master node and executes 9999 POSTMAN transactions; Post creates M threads for every node, and each executes 3333 DOLTM transactions. For multithreading, the experiment creates a new Berkeley DB database handle and RMI handle for each thread in order to reduce the competition between the threads.

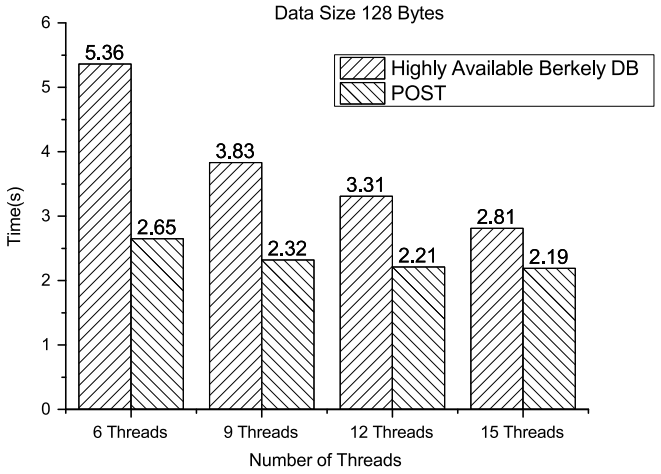

Fig. 7. Performance comparison of POST and highly available Berkeley DB (the size of data is 128 bytes).

5.3. Analysis

As shown in Figs. 7 and 8, when the data size is 512 bytes, the performance of POST declines slightly with the increase of thread number; when the data size is 128 bytes, the performance increases slightly with the increase of thread number. For highly available DB, when the data size is 512 bytes, the performance gradually stabilizes with the increase of thread number; when the data size is 128 bytes, the performance increases slightly after promoting sharply with the increase of thread number. This is because when the number of the threads is large, system I/O is limited and systems performance will be declined by some unnecessary context interaction and the hardware cache recovery.

For the data size of 512 bytes, when the number of threads is 6, the POST achieves the biggest performance improvement of 45.6% compared with highly available Berkeley DB; when the number of threads is 9%, 12% and 15%, 20.6%, 11.7% and 11.4% performance improvements can be achieved respectively. For the data size of 128 bytes, when the number of threads is 6, the POST achieved the biggest performance improvement of 102% compared with highly available Berkeley DB; when the number of threads is 9%, 12% and 15%, 65.1%, 49.8% and 28.3% performance improvement can be achieved respectively.

The experimental results show that POST achieves better acceleration effects when the data size is small, because at this time I/O limitation is light. It also shows an important feature of the database, that is, I/O is an important factor which decides the performance of database. Frequent transactional inserts lead to a large number of I/O operations, so the performance is affected seriously.

6. Related work

With the rapid development of high cost-effective computer and the higher requirement of information processing capability, distributed database and transaction systems attract more attention and are being widely used. In particular, they are more and more important with the growth of big data and cloud computing, also the appearance of other applications with massive data.

NoSQL (Not Only SQL) has become the first alternative to relational databases with higher flexibility, availability and fault tolerance in big data and cloud computing environments. Bigtable [21]

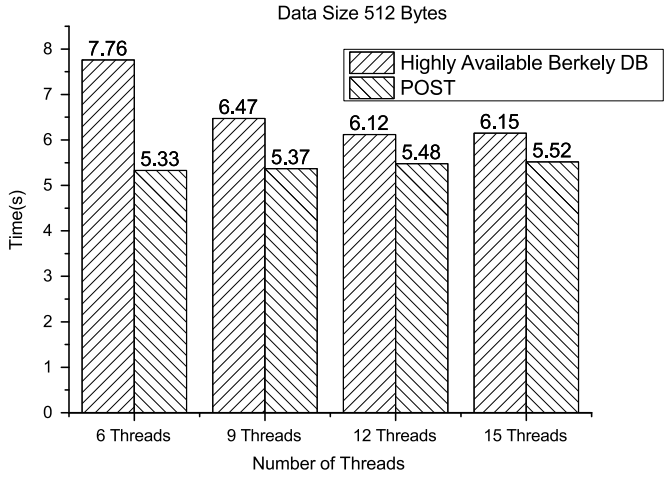

Fig. 8. Performance comparison of POST and highly available Berkeley DB (the size of data is 512 bytes).

is a typical NoSQL built on Google file system (GFS) which is a scalable distributed file system for large distributed data-intensive applications. Berkeley DB also belongs to NoSQL. Compared with Bigtable, Berkeley DB is easier for non-programmers to use because has a more flexible data model and is applied to a wider range in commerce. NewSQL (relative to NoSQL) aims to achieve the benefits of scalability and fault tolerance provided by NoSQL, meantime it is still using the relational model. Voltdb [22] is an in-memory NewSQL database with incredible high read and write speeds. The database tables and the stored procedures are partitioned and saved in multiple sites. As each site runs single thread, it eliminates the cost on locking and latching. The performance of Voltdb is improved compared with Berkeley DB, however the flexibility of Voltdb is much lower than Berkeley DB as the system should restart if there is a change in size of cluster. Even though Berkeley DB is a good data management system in general, the shortcoming that not supporting parallel writing does worsen its performance in big data and cloud computing environments. Under the background of those, this paper has designed and implemented a middleware-level distributed transaction processing system called POST which could solve this issue in most conditions. By the way, with the rapid development of cloud computing, data security [23–25] in storage should draw enough attention in the productive environment.

[26] proposes one transaction layer named Argos. Argos provides two transaction services, which can help distributed transactions conform to traditional ACID attributes, meanwhile, can support the transaction whose life cycle is longer by compensation. This is done in order to achieve high performance and high availability of data management. [27] proposes a parallel file system to improve I/O performance on the cluster. [28] makes one middleware named Sprint to extend from an independent memory database to a non-shared server cluster. Aiming at issues on high message delay and resource lock of publish/subscribe middleware, [29] proposes the middleware TOPS which is transaction-oriented, based on HTS (a middleware to introduce the transaction concept), combine transaction concept and publish/subscribe service more closely. [30] implements a middleware support copy based on PostgreSQLs duplication, provide snapshot isolation the same with PostgreSQL granularity.

[31] develops a new type of OLTP system called H-Store which runs in non-shared memory distributed clusters. It is based on Multi-single threaded engine collaboration, providing more efficient processing services for OLTP transaction. [32] proposes TTLMosaic, which could make parallel efficiency higher and scalability more excellent. [33] implements two parallel I/O strategies to speed up the system of numerical weather forecast. [34] implements a new tool which is more suitable for the parallelism in stream data.

7. Conclusion

Aiming at the shortcomings of the highly available Berkeley DB, this paper designs and implements distributed transaction processing middleware system called POST, which includes distributed database system called POSTBOX based on the Berkeley DB and data partition, and distributed transaction processing middleware called POSTMAN. The POSTBOX inherits the high availability of Berkeley DB from them, and extends the partition of the highly available Berkeley DB, so it overcomes the problem that highly available Berkeley DB natively does not support parallel writing across multiple nodes. POSTMAN is a middleware designed to adapt PRB, it monitors the state of each node in POSTBOX via *PRBSA*, and is in charge of the initiation, execution, and migration of transactions. Experimental results show that the write performance of POST is superior to that of highly available Berkeley DB. Especially when the size of transaction data is small, the acceleration of POST is obvious.

Acknowledgments

This work was supported in part by the National High Technology Research and Development Program of China (No. 2015AA01A303), Beijing Key Subject Development Project (XK10080537), NSF grants CNS 149860, CNS 1461932, CNS 1460971, CNS 1439672, CNS 1301774, ECCS 1231461, ECCS 1128209, and CNS 1138963.

References

[1] A. Pavlo, E.P.C. Jones, S. Zdonik, On predictive modeling for optimizing transaction execution in parallel OLTP systems, Proc. VLDB Endow. 5 (2) (2011) 173–182.
[2] M. Stonebraker, New opportunities for new SQL, Commun. ACM 55 (11) (2012) 10–11.
[3] M. Negri, G. Pelagatti, L. Sbattella, Formal semantics of SQL queries, ACM Trans. Database Syst. 16 (3) (1991) 513–534.
[4] L. Wang, D. Chen, W. Liu, Y. Ma, Y. Wu, Z. Deng, Dddas-based parallel simulation of threat management for urban water distribution systems, Comput. Sci. Eng. 16 (1) (2014) 8–17.
[5] M. Burrows, The chubby lock service for loosely-coupled distributed systems, in: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI'06, USENIX Association, Berkeley, CA, USA, 2006, pp. 335–350.
[6] Y. Ma, L. Wang, P. Liu, R. Ranjan, Towards building a data-intensive index for big data computing—A case study of remote sensing data processing, Inform. Sci. 319 (2014) 171–188.
[7] L. Wang, H. Geng, P. Liu, K. Lu, J. Kolodziej, R. Ranjan, A.Y. Zomaya, Particle swarm optimization based dictionary learning for remote sensing big data, Knowl.-Based Syst. 79 (2015) 43–50.
[8] K. He, A. Fisher, L. Wang, A. Gember, A. Akella, T. Ristenpart, Next stop, the cloud: Understanding modern web service deployment in EC2 and AZURE, in: Proceedings of the 2013 Conference on Internet Measurement Conference, IMC'13, ACM, New York, NY, USA, 2013, pp. 177–190.
[9] B. Nemade, S. Moorthy, O. Kadam, Cloud computing: Windows AZURE platform, in: Proceedings of the International Conference & Workshop on Emerging Trends in Technology, ICWET'11, ACM, New York, NY, USA, 2011, pp. 1361–1362.
[10] Overview of oracle berkeley db, 2010. http://www.oracle.com/us/products/database/berkeley-db/overview/index.html.
[11] A.A. Farrag, M.T. Özsu, Using semantic knowledge of transactions to increase concurrency, ACM Trans. Database Syst. (TODS) 14 (4) (1989) 503–525.
[12] J. Krueger, C. Kim, M. Grund, N. Satish, D. Schwalb, J. Chhugani, H. Plattner, P. Dubey, A. Zeier, Fast updates on read-optimized databases using multi-core CPUS, Proc. VLDB Endow. 5 (1) (2011) 61–72.
[13] W. Xue, J. Shu, Y. Liu, M. Xue, Corslet: A shared storage system keeping your data private, Sci. China Inf. Sci. 54 (6) (2011) 1119–1128.
[14] J. Shu, Z. Shen, W. Xue, Shield: A stackable secure storage system for file sharing in public storage, J. Parallel Distrib. Comput. 74 (9) (2014) 2872–2883.
[15] L. Millet, M. Lorrillere, L. Arantes, S. Gançarski, H. Naacke, J. Sopena, Facing peak loads in a P2P transaction system, in: Proceedings of the First Workshop on P2P and Dependability, ACM, 2012, pp. 1–7.

[16] A. Pavlo, E.P. Jones, S. Zdonik, On predictive modeling for optimizing transaction execution in parallel OLTP systems, Proc. VLDB Endow. 5 (2) (2011) 85–96.
[17] I. Sarr, H. Naacke, S. Gançarski, Transpeer: Adaptive distributed transaction monitoring for web2. 0 applications, in: Proceedings of the 2010 ACM Symposium on Applied Computing, ACM, 2010, pp. 423–430.
[18] W. Emmerich, M. Aoyama, J. Sventek, The impact of research on middleware technology, ACM SIGOPS Oper. Syst. Rev. 41 (1) (2007) 89–112.
[19] S. Gilbert, N. Lynch, Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, ACM SIGACT News 33 (2) (2002) 51–59.
[20] E. Brewer, Cap twelve years later: How the "rules" have changed, Computer 45 (2) (2012) 23–29.
[21] 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 Trans. Comput. Syst. 26 (2) (2008) 4:1–4:26.
[22] M. Stonebraker, A. Weisberg, The voltDB main memory DBMS, IEEE Data Eng. Bull. 36 (2) (2013) 21–27.
[23] J. Shu, Z. Shen, W. Xue, Y. Fu, Secure storage system and key technologies, in: Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific, IEEE, 2013, pp. 376–383.
[24] Z. Shen, J. Shu, W. Xue, Keyword search with access control over encrypted data in cloud computing, in: 2014 IEEE 22nd International Symposium of Quality of Service (IWQoS), IEEE, 2014, pp. 87–92.
[25] Z. Shen, J. Shu, W. Xue, Preferred keyword search over encrypted data in cloud computing, in: 2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS), IEEE, 2013, pp. 1–6.
[26] A.-B. Arntsen, M. Mortensen, R. Karlsen, A. Andersen, A. Munch-Ellingsen, Flexible transaction processing in the argos middleware, in: Proceedings of the 2008 EDBT Workshop on Software Engineering for Tailor-Made Data Management, ACM, 2008, pp. 12–17.
[27] L. Wang, Y. Ma, A. Zomaya, R. Ranjan, D. Chen, A parallel file system with application-aware data layout policies in digital earth, IEEE Trans. Parallel Distrib. Syst. 26 (6) (2014) 1497–1508.
[28] L. Camargos, F. Pedone, M. Wieloch, Sprint: a middleware for high-performance transaction processing, ACM SIGOPS Oper. Syst. Rev. 41 (3) (2007) 385–398.
[29] Y. Shatsky, E. Gudes, E. Gudes, Tops: a new design for transactions in publish/subscribe middleware, in: Proceedings of the Second International Conference on Distributed Event-Based Systems, ACM, 2008, pp. 201–210.
[30] Y. Lin, B. Kemme, M. Patiño-Martínez, R. Jiménez-Peris, Middleware based data replication providing snapshot isolation, in: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, ACM, 2005, pp. 419–430.
[31] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E.P. Jones, S. Madden, M. Stonebraker, Y. Zhang, et al., H-store: a high-performance, distributed main memory transaction processing system, Proc. VLDB Endow. 1 (2) (2008) 1496–1499.
[32] Y. Ma, L. Wang, A.Y. Zomaya, D. Chen, R. Ranjan, Task-tree based large-scale mosaicking for massive remote sensed imageries with dynamic dag scheduling, IEEE Trans. Parallel Distrib. Syst. 25 (8) (2014) 2126–2137.
[33] Y. Zou, W. Xue, S. Liu, A case study of large-scale parallel i/o analysis and optimization for numerical weather prediction system, Future Gener. Comput. Syst. 37 (2014) 378–389.
[34] Z. Deng, X. Wu, L. Wang, X. Chen, R. Ranjan, A. Zomaya, D. Chen, Parallel processing of dynamic continuous queries over streaming data flows, IEEE Trans. Parallel Distrib. Syst. 26 (3) (2015) 834–846.

Jianjiang Li is currently an associate professor at University of Science and Technology Beijing, China. He received his Ph.D. degree in computer science from Tsinghua University in 2005. He was a visiting scholar at Temple University from Jan. 2014 to Jan. 2015. His current research interests include parallel computing, cloud computing and parallel compilation.

Qian Ge is currently a student in University of Science and Technology Beijing for her master degree. Her research interests include distributed system technology and fault tolerance for database. She received her bachelor's degree in 2012 from China Women's University.

Jie Wu is the chair and a Laura H. Carnell Professor in the Department of Computer and Information Sciences at Temple University. Prior to joining Temple University, USA, he was a program director at the National Science Foundation and a distinguished professor at Florida Atlantic University. He received his Ph.D. degree from Florida Atlantic University in 1989. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly published in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Computers, IEEE Transactions on Service Computing, and Journal of Parallel and Distributed Computing. Dr. Wu was general co-chair/chair for IEEE MASS 2006 and IEEE IPDPS 2008 and program co-chair for IEEE INFOCOM 2011. Currently, he is serving as general chair for IEEE ICDCS 2013 and ACM MobiHoc 2014, and program chair for CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Yue Li has graduated from University of Science and Technology Beijing with his master degree in 2014. He received his bachelor's degree in 2011 from Tian jin College, University of Science and Technology Beijing. His research interests include cloud computing and distributed system technology.

Xiaolei Yang is currently a student in University of Science and Technology Beijing for his master degree. He received his bachelor's degree in 2013 from University of Science and Technology Beijing. His research interests include cloud computing and recommend systems.

Zhanning Ma is currently a student in University of Science and Technology Beijing for his master degree. His research interests include distributed storage technology and data duplicate removal. He received his bachelor's degree in 2010 from Taishan Medical University.