DEIM Forum 2011 E2-5

An Evaluation on Dynamic Access-Skew Balancing Performance of Compound Parallel Btree for Chained Declustering Parallel Systems

Min LUO † Haruo YOKOTA ‡

† Department of Computer Science, Tokyo Institute of Technology 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8552, Japan

E-mail: † luomin@de.cs.titech.ac.jp, ‡ yokota@cs.titech.ac.jp

Abstract Access skew is the most important challenge for scalable parallel systems, especially when data are in range partitioned schema. To realize scalability, many dynamic accesses skew balancing methods with data reorganization and parallel index structures on shared-nothing parallel infrastructure have been proposed. Data migration with range-partitioned placement using a parallel Btree is one solution. The combination of range partitioning and chained declustered replicas provides the dynamic skew balancing abilities for high scalability. However, no previous study has provided any practical implementation of this capability. In addition, independent treatment of the primary and backup data in each node results inefficient skew balancing. We propose a novel compound parallel index, termed Fat-Btree, to provide access paths to both primary and backup data across a chained declustering system for efficient dynamic skew balancing with low maintain cost. Experiments using PostgreSQL on a 160-node PC cluster demonstrate the effects.

Keyword Fat-Btree, Skew Balancing, Scalability, Parallel Database

1. Introduction

The explosive growth of digital information together with the demand for its 24H on-line availability have generated interest in research on databases with high scalability (HS) and high availability (HA), both of which are improved by the replication on a shared-nothing parallel infrastructure. Efficient accessing methods with skew-balancing ability and data replication strategies are very important for achieving HS and HA, respectively [1].

It has been argued that system with strong consistency have unstable behavior when scaled up [3], therefore, research interest has become focused on consistency and availability, leaving the scalability issue seldom addressed [4]. Recently, well-known massive data centers for cloud-based applications, such as PNUTS, Dynamo and BigTable, have adopted the strategy of sacrificing strong consistency for availability and scalability. This strategy benefits scalability, but the advantage of replication in gaining higher throughput is lost. Because the replicas are not ready to be queried most of the time, client queries requiring high data consistency cannot be balanced with replicas, and system availability may also be lost in the long run if strong consistency cannot be ensured [4]. Therefore, sacrificing consistency for scalability does not have a dependable effect, and this has long been an obstacle to developing efficient replication techniques.

To address this problem, efficient data access methods with strong consistency guarantees play an important role. Basically, there are two main methods of efficient data accessing in distributed systems: distributed-hash-table (DHT)-based [5] and B-tree-based [8] methods. DHT based methods efficiently support point queries, but destroy the semantics and locality of data and incur a high cost in range queries. Btree-based methods for value-range partitioning schema efficiently support point and range queries, but suffer limited access-skew balancing ability.

To provide fast and scalable data accessing for range partitioning, many B-tree-based parallel index structures have been introduced to shared-nothing environment [3]. However, they do not provide the dynamical access-skew balancing ability required for high scalability. To balance the skew, high-cost data migration or index reconstruction processes are always required in these systems to prepare parallel data access paths to the replicas on other nodes for load balancing. In addition, current parallel B-tree indexes do not consider index management on replica data for higher system scalability and availability.

In this paper, we propose a database infrastructure for indexing range-partitioned data on chained declustering parallel systems by a novel compound parallel B-tree structure with dynamical access-skew-balancing ability. In this infrastructure, data consistency is guaranteed by eagerly synchronization with low cost due to the minimum replication degree in chained declustering. Our proposed compound parallel B-tree structure on these consistent data copies supports immediate load balancing, by transferring the load between primary and replica copies without data migration cost or index reconstruction cost. Thus, the skew problem in range partition schema is solved by this dynamic skew balancing ability, and it makes range-partition schema superior to existing hash-based methods because the efficient range query ability it preserves. We evaluate the dynamic access-skew balancing ability in this infrastructure by using a prototype system we developed. Experimental results on a 160-nodes machine will demonstrate the efficiency of our proposed compound parallel B-tree index.

2. Background

We briefly review two existing technologies for constructing scalable and available shared-nothing parallel databases: data placement strategies and parallel indexing structures.

2.1 Data Placement Strategies

The shared-nothing configuration is generally used to achieve high performance in parallel database systems. This is because it consists of a set of independent PEs that does not share memory or disks so that the computational and I/O resources can be maximized. It is simple to speed and scale the system up to hundreds of PEs [9].

Chained declustering [6] is a technique that offers high availability and good load balancing on shared-nothing parallel systems, which has higher reliability than other declustering schemes, such as mirrored/interleaved declusterings [13]. In chained declustering schema, PEs in one relation-cluster maintains two physical copies of the relation, a primary copy and a backup copy. These two copies are declustered across the cluster by the same partitioning strategy. Because the corresponding fragments of primary and backup copies are stored on different PEs, no data is lost after a failure. In addition, chained declustering is suitable for range partitioned data placement on both primary and backup data copies, which is able to provide efficient range query performance than all the other declustering schemes.

As mentioned in Section 1, the selection of the data-partitioning strategy determines the availability and throughput of chained declustering schema. Although each partitioning strategy has at least one significant limitation, overcoming the current shortages in one of them will make it outperform the other. In this paper, we will provide the solution to improve the inefficient skew balancing performance of range partitioning scheme based on chained declutering systems.

2.2 Parallel Indexing Structures

As mentioned in Section 1, there are two main methods of distributed data accessing. DHT-based methods uniformly map nodes and data objects into a single ID space, and each node is responsible for a specific range of the ID space. Considerable effort has been devoted to supporting range-query applications in DHT-based systems [10], [11]. However, their efficiency is still limited as all systems require additional structures, which introduce an extra overhead. On the other hand, B-tree-based parallel indexing with efficient range-query capability has been proposed for high throughput and efficient data skew handling via index node migration. However, it suffers from a high index update cost and limited access-skew-balancing ability. Because both strategies have their advantages and disadvantages, the disadvantage of the B-tree-based parallel index can be reduced if its shortcomings can be overcome.

To reduce the update cost in parallel B-tree indexing, an update-conscious parallel B-tree structure, a Fat-Btree, has been proposed [2]. An example of a four-PE Fat-Btree is given in Fig. 1, where multiple copies of index nodes close to the root node with relatively low update frequency are replicated on several PEs, while leaf nodes with relatively high update frequency are distributed across the PEs. Thus, the maintenance cost of the Fat-Btree is much lower than that of other parallel Btree structures. In addition, Fat-Btree has a higher cache hit rate [2] and more efficient concurrency control protocols than other methods [7].

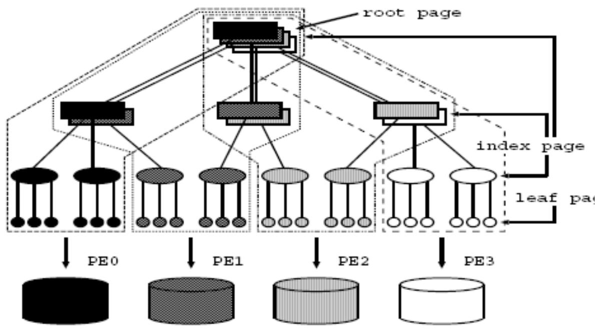


Fig.1 Fat-Btree

3. Compound Treatment Infrastructure

Here, we propose our method for high scalability in parallel database by supporting dynamic load balancing.

Because replicas in existing parallel databases cannot be directly accessed in parallel, system performance can be improved if the backup are managed by one parallel index together with the primary. Because chained declustering scheme places continuous fragments in range partition way, data are coupled and indexed in the compound Fat-Btree structure without any intersection. Thus, there are two subFat-Btrees to manage the primary

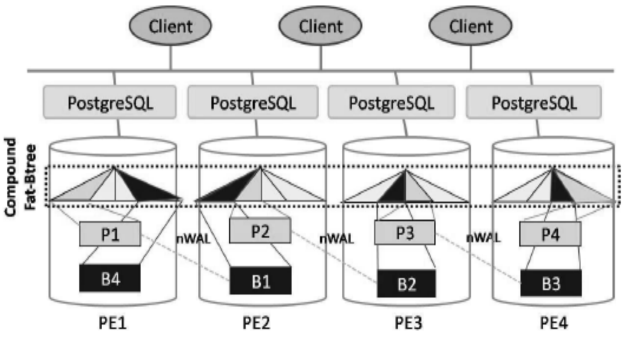


Fig.2 Infrastructure of CompIndexCDR

and backup data on each PE. Because the compound subFat-Btree on one PE also has the overlapping intermediate paths to its neighboring subFat-Btrees, similar to the original Fat-Btree, it provides an access path from the root node to all primary and backup data located in any PEs. We name this infrastructure as compound index chained declustered replication or CompIndexCDR.

Figure 3 shows an example of this infrastructure. The upper part shows a global view of the intermediate nodes in the B-tree index for all the data over the range 1–60, which are evenly stored by four PEs. Using the original Fat-Btree, some of the intermediate nodes are replicated in several PEs because they are overlapped. Note that intermediate nodes may have pointers to copies of their leaf nodes located in other PEs. For example, the copy of node “1, 10” in PE2 has a pointer to the leaf nodes “1, 7” and “10, 16” in PE1 and the leaf node “10, 16” in PE2. These overlapped intermediate paths allow any data to be traced in the system from the root index.

As shown in the lower part of Fig. 3, each PE has two subFat-Btrees, for its backup and primary data. Due to the same copy of data, they have a similar index structure and intermediate nodes in the primary and replica. Each parent index node has pointers to its child index nodes that are multiply replicated in near PEs. We mark these paths with different flags value (‘P/B’) to identify what kind of child index nodes they lead to. These paths are kept available during SMOs [7]. Transactions are carried out only following the paths that are marked with flag ‘P’ to access the primary data. Flag of the paths to primary and backup data will be exchanged when skew happens. The skewed access will follow the changed paths to replicas, thus balance the skew without data migration or index reconstruction cost.

3.1 Index Maintenance in Compound Fat-Btree

In CompIndexCDR, index consistency between primary

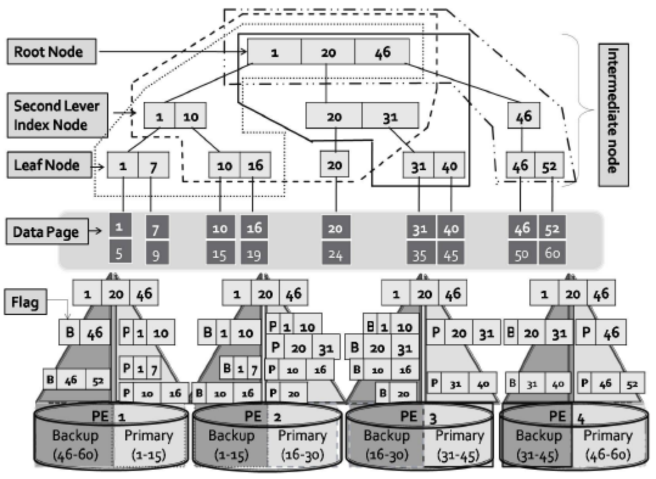

Root Node
Second Lever Index Node
Leaf Node
Data Page
Flag
Intermediate node
1 20 46
1 10
20 31
46
1 7
10 16
20
31 40
46 52
PE 1 Backup (46-60) Primary (1-15)
PE 2 Backup (1-15) Primary (16-30)
PE 3 Backup (16-30) Primary (31-45)
PE 4 Backup (31-45) Primary (46-60)

Fig.3 A Compound Fat-Btree Model

and backup is important. Compared to original Fat-Btree, there are more overlapped intermediate index nodes and paths requiring the consistency maintenance between neighboring PEs. This task can been divided into two phases. Phase-a: potentially conflicting transactions on primary and backup modification of the same replicas of intermediate index nodes are avoided. Phase-b: modifications on overlapped index nodes are propagated and synchronized in all related PEs.

For phase-a: IX/X-latch-requiring process only in primary subFat-Btree will avoid conflicts transactions for both primary and backup data. Because the backup’s intermediate index nodes contain the same data and structures as the primary. Any conflicting transactions on the primary index also conflict on the backup when synchronizing backup. Therefore, the original concurrency control method [7] is still effective in CompIndexCDR.

For phase-b: when a transaction successfully obtains all the X latches required at a *host* PE, all the participating PEs split their specified node and send back the pointer of the newly created node to the *host* PE. The *host* PE manages the sets of pointers received and propagates new pointer information to update them in the participating PEs. Because the high synchronization cost may occurs if all levels of intermediate paths are maintained between the primary and backup indexes, we maintain two levels of them with the index fanout number equal to 16. In this case, the smallest transformable load unit of each index *Flag* modification is up to 1/4 of the average load, even when *N* is as large as 64, thus provides a precise skew balancing ability. Experimental results in Section 4 will show the efficiency of our index maintenance method in dynamic skew balancing with high system throughput.

3.2 Dynamic Skew Balancing Algorithms

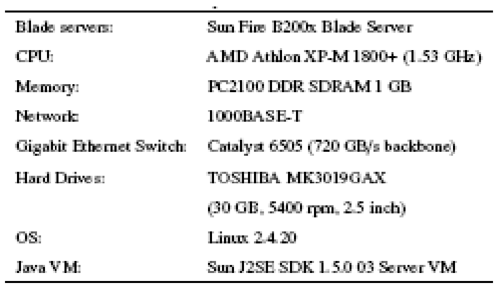
Because of the limitation to the length of this paper, we provide the pseudo-code for the load-balancing here.

A load-balancing algorithm for CompIndexCDR is shown in Fig. 4. A primitive method is used to find the skews in the system first. This method continuously forwards a list that contains the volume of recent queries in every PE. Each PE places its recent query volume into its corresponding entry in the list and forwards the list to the next PE. If a PE founds its volume exceeds some threshold above the average amount of queries in the list, it will start a load balancing process automatically.

```
Dynamic Load Balancing
begin
  Let op, $lp$ and $rp$ be the target, left and right PE, respectively;
  Let $pdm[i]$ be the amount of primary data on node $i$;
  Receive a list of loads $ll$ from $lp$;
  Update $ll[op]$ to reflect the current workload of the PE;
  Calculate the average load $al$ from $ll$;
  if $ll[op] > al + Threshold$ then
     // if this skew can be balanced without data migration
    Calculate the new primary placement $pp[i] += (ll[op] - al) / 4$;
    for (i = 0; i < n; i++)        // n is the number of PEs
    {
      broadcast the modification msg to $node[i]$ to change primary;
      if $pdm[i] > pp[i]$ then
      {
          Demote_Primary(the leftmost data page in $pdm[i]$, $pdm[i] - pp[i]$);
          set the 'Flag' for the new 'backup' at $pdm[i] - pp[i]$;
          broadcast the modification msg to $node[i+1]${or $node[0]$ if $i = n$}to change backup there;
          Promote_Backup(the leftmost data page in $pdm[i+1]$, $pdm[i] - pp[i]$);
          set the 'Flag' for the new 'primary' at $pdm[i] - pp[i]$;
      }
      else
      {
          Promote_Backup(the rightmost data page in $pdm[i]$, $pp[i] - pdm[i]$);
          set the 'Flag' for the new 'primary' at $pp[i] - pdm[i]$;
          broadcast the modification msg to $node[i-1]$ {or node[n] if i = 0}to change primary there;
          Demote_Primary(the rightmost data page in $pdm[i]$, $pp[i] - pdm[i]$);
          set the 'Flag' for the new 'backup' at $pp[i] - pdm[i]$;
      }
     }
     Reset list $ll$ ;
  if $op$ is the rightmost PE then
    Send $ll$ to the leftmost PE;
  else
    Send $ll$ to $rp$;
end
```

Fig. 4 Dynamic Load Balancing

By using the above algorithm, each PE is able to transfer up to its primary's entire access load to its replica data at a neighbor PE. The replica PE is able to further transfer a proper amount of its own primary's access load to its replica PE. This process will be carried across the PE clusters to achieve an evenly balanced result that each PE has the same load. However, this evenly skew balancing ability depends on the skew degree in the chained declustering system.



Blade servers:	Sun Fire B200x Blade Server
CPU:	AMD Athlon XP-M 1800+ (1.53 GHz)
Memory:	PC2100 DDR SDRAM 1 GB
Network:	1000BASE-T
Gigabit Ethernet Switch:	Catalyst 6505 (720 GB/s backbone)
Hard Drives:	TOSHIBA MK3019GAX
	(30 GB, 5400 rpm, 2.5 inch)
OS:	Linux 2.4.20
Java VM:	Sun J2SE SDK 1.5.0 03 Server VM

Fig.5 Experimental Enviroment

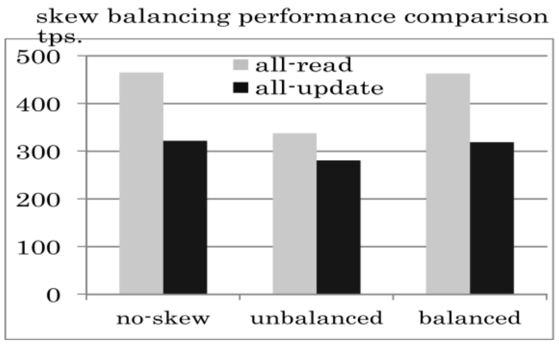


Fig. 6 Skew Balancing Performance Comparison

In particular, assume a skewed PE has load of β and all the other PEs have a same load of δ. The skew β that could be evenly balanced is up to 2(n-1) /(n-2) times of δ in this CompIndexCDR infrustructure**.** This is because the skewed $PE_{\_S}$ may have half of β at the maximum to be transferred to its replica $PE_{\_R}$ (otherwise, $PE_{\_R}$ will have more load than $PE_{\_S}$ even if $PE_{\_R}$'s own primary workload is transferred to other PEs). Thus, the evenly balanced results of ***$(\beta + (n-1)*\delta)/n$*** should be larger than the minimum of ***$\beta/2$*** load that a skewed PE could be remained, otherwise, skewed PE will have more load than the average load of other PEs.

4. Experiments

To evaluate the proposed compound treatment index structure, we implement a Compound Fat-Btree with open source PostgreSQL DBMS. Details of this implementation are introduced in [1]. Other configurations of the cluster system used in our experiments are shown in Fig. 5.

In this paper, we focus on the skew balancing performance evaluation in the Compound Fat-Btree based system. For simplicity we assumed the following items:

• each request occurs individually, according to the probability density function, and is not affected by the state of any other request.

• both primary copies and backup copies are balanced on all disks as the initial state.

• every disk has a sufficiently large queue for storing requests.

Firstly, we examine a constant skew balancing efficiency in the system. In this experiment, we use a four-PE cluster. Each PE has 10,000 tuples and serves 32 users. 40% of the requests are accessing the data in one PE and the other three PEs share the remaining 60%. Thus workload on the hot node is two times of that on the other nodes, which generates a two times skew in the system.

Fig. 6 shows that this skew decreases the throughput in “all-read” and “all-update” by 25% and 15%, respectively. After using the balancing methods in Sec. 3.2, the throughput in “all-read” is almost the same as that in “no-skew”, and the “all-update” is also much improved, which verifies the effectiveness of load-balancing as argued in Sec. 3.

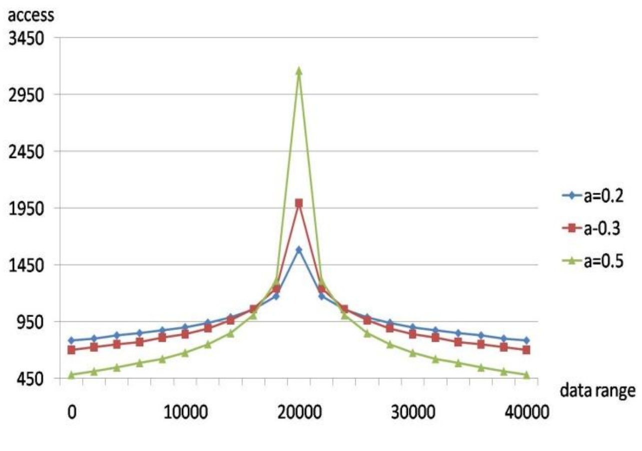


Fig.7 Access patterns

Secondly, we generate access frequency (heat) based on a Zipf() distribution function [12]. For example, the data stored in above four-PE cluster will be queried by randomly selected key values of [$20000(1\pm\times x^{\frac{1}{1-\alpha}})$]. x $\in(0,1)$ and is chosen randomly when generating each query; α is the parameter in Zipf() which determines the skew degree in the created access pattern. In Fig. 7, the horizontal axis stands for the data ‘x’ ranged from (0, 40,000) that are stored in our cluster system, the vertical axis stands for the number of queries that will be generated to query a data ‘x’ based on the Zipf() function. As it shows, when α = 0.2, there is about 1.5 times skew; when α = 0.3, there is about 2 times skew; and when α = 0.5, the skew will be over three times.

We first observe the system throughput which changes during the whole skewed period and the corresponding dynamic skew balancing process.

The threshold is set to 0.05 in this experiment. Experimental results are shown in Fig. 8.

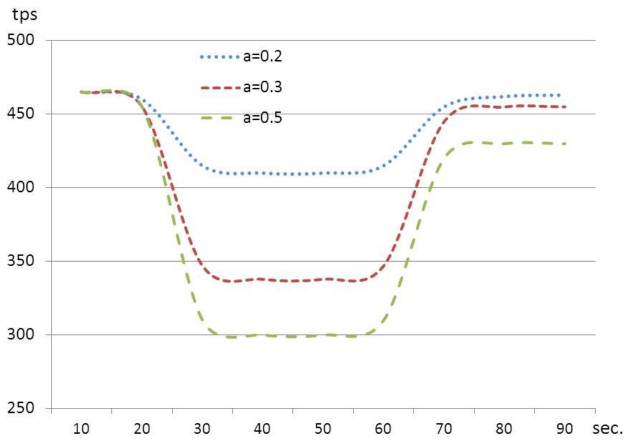


Fig. 8 Zipf distribution skews balancing (threshold=0.05)

As it shows, the skew of each access pattern results throughput decreasing at about 20%, 27% and 33%, respectively. About 40 seconds later, skew detection algorithm (Fig. 4) found the skew and starts load balancing. This is because in our algorithm, the *ll[n]* list stores the recent 10 workload records of each node and is transferred around the cluster one by one every second. The load balancing process is able to recovery system throughput with only 0.5%, 2.2%, and 7.5% decrease.

However, if the threshold is set to 0.03, the load balancing performance of α = 0.5 will be better. Because the first balanced results in Fig.8 generates a new skewed node (1.16 times) for α = 0.5, when transfer the first hot node’s load to the other nodes. When setting threshold = 0.05, this new skewed node will not be discovered. As shown in Fig. 9, the skew will be further balanced when a small threshold is chosen.

In addition, the hottest point in the access pattern may change periodically. This dynamic changing skew can be simulated by modifying value ε in the randomly selected key values function Zipf(α)= {[$20000(1\pm\times x^{\frac{1}{1-\alpha}})$+ ε] mod

40000}, ε ∈(0,10000,20000,30000). Thus, the hottest PE in the cluster may also change dynamically. Based on the skew detection method in Fig. 4, this dynamic changing skewed node will also be discovered when its access pattern is found exceed the threshold.

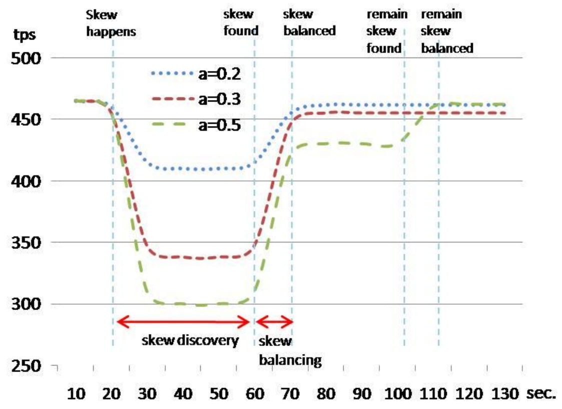


Fig. 9 Zipf distribution skews balancing (threshold=0.03)

Note that skew is balanced by changing the primary/backup roles of some portion of data in each node; therefore, these data should be turned back as their original roles before resign new roles for them to balance a secondary skew. For example, a secondary skew happens on a fake primary of node(i), it should not be balanced by demoting current primary and promoting the backup data on node(i+1), as described in Fig. 4, because the backup data of the fake primary is on node(i-1).

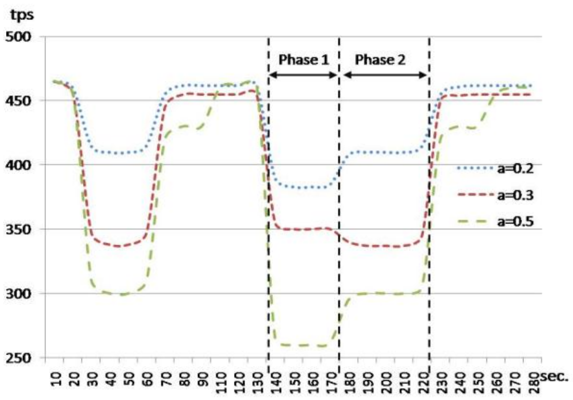


Fig. 10 Dynamic skew balancing

Fig. 10 shows the dynamic load balancing performance. Each secondary skew in the access pattern is able to be balanced within two phases. Phase 1 shows the throughput after the secondary skew happens in system. Phase 2 shows the throughput after the roles of data is turned as usual. Because the role of primary/backup data is changed at first skew balancing process (before Phase 1), access to the fake primary/backup under new access patterns results the difference between Phase 2. However, after Phase 2, system skew is still able to be balanced in our system.

We provide a skew balancing comparison result between our proposed system and original chained declustering system to illustrate the efficiency of the proposed CompIndexCDR more clearly. In this experiment, the original chained declustering system has a Fat-Btree index on the primary data for efficient parallel processing of user requests. However, the backup data are indexed by independent local B-tree index on each PE, thus the backup data on one PE are not able to be accessed from other PEs in the system. To provide load balancing ability in the original chained declustering, we have to construct parallel index for the backup data whose primary copy is having skewed workload. In this experiment, the parallel index construction process is done by physically dumping the backup data into the primary parts on each node, thus it will take much longer time than the 'Flag' modification process in CompIndexCDR, and system throughput during this process will also be greatly reduced, which is shown by the red dot lines in Fig. 11. On the other hand, CompIndexCDR immediately balanced the system skew once the skew happens, and system throughput are almost as same as original in most of time.

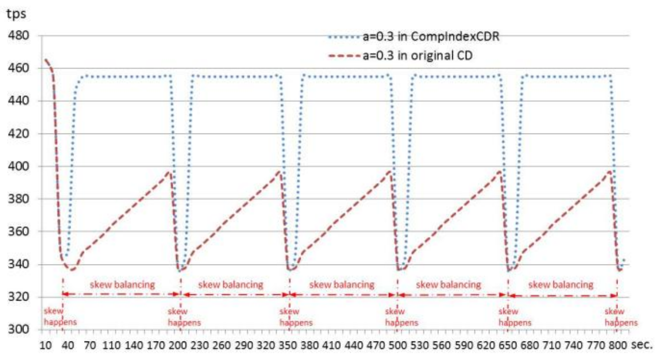


Fig. 11 Comparison of CompIndexCDR and original CD

5. Conclusions

In this paper, we provide the evaluation results for the dynamic skew balancing ability in a compound parallel B-tree index we proposed on chained declustering scheme CompIndexCDR. As far as we know, this is the first treatment to support dynamic skew handling by the compound management of primary and backup data in the replication system. System throughput is able to be recovered within our system. We will provide more efficient skew detecting and balancing algorithms for the proposed system in our future work.

6. Acknowledgements

This research was supported in part by MEXT Grant-in-Aid for Scientific Research on Priority Areas (#201013017) and JSPS Grant-in-Aid for Scientific Research (A) (#22240005).

Referencs

[1] M. Luo, A. Watanabe, and H. Yokota, "Compound treatment of chained declustered replicas using a parallel Btree for high scalability and availability," in DEXA'10, LNCS, Vol. 6262/2010, pp.49-63, 2010.

[2] H. Yokota, Y. Kanemasa, and J. Miyazaki, "Fat-Btree: an update conscious parallel directory structure," in ICDE'99., p.448.

[3] J. Gray, P. Helland, and P. O'Neil, "The dangers of replication and a solution," Proc. ACM SIGMOD, pp.172-182, 1996.

[4] H. Yu and A. Vahdat, "The costs and limits of availability for replicated services," ACM Trans. Compt. Syst, vol.24, no.1, pp.70-113, 2006.

[5] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT." Proc. the annual Conf. on USENIX Annual Tech. Conf. (ATEC '04). USENIX Association, Berkeley, CA, USA, 2004.

[6] H. Hsiao and D. J. DeWitt, "Chained declustering: a new availability strategy for multiprocessor database machines," Proc. ICDE'90, pp.456-465.

[7] T. Yoshihara, D. Kobayashi, and H. Yokota, "Mark-opt: A concurrency control protocol for parallel B-tree structures to reduce the cost of SMOs," IEICE Trans. Inf. Syst., vol.90, no.8, pp.1213-1224, 2007.

[8] R. Bayer, E. McCreight, "Organization and Maintenance of Large Ordered Indices," Mathematical and Information Sciences Report No. 20, Boeing Scientific Research Laboratories, 1970.

[9] D. Dewitt and J. Gray, "Parallel database systems: the future of high performance database systems," Commun. ACM, vol.35, no.6, pp.85-98, 1992.

[10] A. Gupta, D. Agrawal, and A. El Abbadi, "Approximate range selection queries in peer-to-peer," Proc. Conf. Innovative Data Systems Research (CIDR), 2002.

[11] J. Gao and P. Steenkiste, "An adaptive protocol for efficient support of range queries in DHT-based systems," in ICNP 04: Proc. 12th IEEE Int. Conf. Network Protocols. Washington, DC, USA: IEEE Comput. Soc., pp.239-250, 2004.

[12] D. E. Knuth. Sorting and Searching. Addison-Wesley Publishing Company, 1973.

[13] Bruce Jacob, Spencer Ng, David Wang, Memory systems: cache, DRAM, disk, Morgan Kaufmann Publishing Company, 2007.