データベースシステム 119−67
（１９９９．７．２３）

Feelifl Hisham、喜連川 優

Shared nothing system の並列インデックスデータベースに於いて、個々の PE に負荷の 偏りができた場合、パフォーマンス は低下してしまう。本論文では、最小の修正コストで、データ を動的に 動かし負荷分散させることにより、このパフォーマンスの低下を解消する手法を提案し、その有効性を示す。

Online Heat Balancing for Parallel Indexed Database On Shared Nothing System

Hisham Feelifl and Masaru Kitsuregawa
Institute of Industrial Science, The University of Tokyo
{hisham, kitsure} @tkl.iis.u-tokyo.ac.jp

Shared nothing systems offer a tremendous processing capacity. In such highly parallel systems the data are typically declustered across the system processing elements (PEs) to exploit the I/O bandwidth of the PEs. However, the access pattern is inherently dynamic, which in turn can lead to performance degradation as some PEs become hot spot (bottleneck). Therefore, reorganization for heat (load) balancing is essential and should be online. Our objective is to migrate the data from the hot PEs to the cold PEs with minimal cost of modifying the index structure of the system, so that the system is heat balancing and consequently higher performance could be achieved with minimal reorganization cost. In this paper, we propose an online heat balancing strategy for parallel indexed database on shared nothing system, in which the data migration process itself is based on the heat statistics generated by the access pattern that may be directed to the distributed index structure of the system. The proposed strategy captures the intuitive goal of distributing the given heat across the system PEs as evenly as possible and the result demonstrates that it is efficient in correcting any degradation in the system performance.

1. Introduction

Nowadays, the shared-nothing parallel architectures have become increasingly popular and attractive for their cost effectiveness, scalability, and availability. In such architectures, there are many processing elements (PEs) connected by an interconnection network where each PE has its processor, exclusive memory modules and one or more disk units. The data are typically declustered across disk drives attached to each PE's and the execution of a transaction or a query is distributed over the network, the only shared resource. However, the access pattern is inherently dynamic, which in turn can lead to performance degradation as some PEs become "hot spot" (frequently accessed). Therefore, reorganization for heat (load) balancing is essential. The basic motivation to investigate and realize heat-balancing facilities comes from simple experience that several applications in shared nothing systems usually do not exploit the system very good. Heat balancing is particularly challenging for evolving workloads, where the hot and cold data change over time. Data reorganization can only counteract such situations, and such reorganizations should be performed online without requiring the system to be quiescent [WZS91 & SWZ93]. Additionally, to achieve efficient query and transaction evaluation, data at each PE are indexed. Therefore, data reorganization should satisfactorily deal with the index modification as moving data from "hot spot" PEs to cold PEs (infrequently accessed) [AON96]. In this paper, we propose an online heat balancing strategy for parallel indexed database on shared nothing system, in which the data migration process itself is based on the heat statistics generated by the access pattern that may be directed to the distributed index structure of the system. The organization of the paper is as follows. In the next section, we briefly discuss the related work for online data reorganization and index modification. Section 3 is the basement for the system distributed search structure. Section 4 clarifies our considerations to the system workload and migration strategy as well. In Section 5, we present our heat balancing strategy. Sec. 6 deals with our experimental work and finally, we summarize and present our conclusions.

2. Related work

There is always some ideal data placement for the workload, which is executing at a particular instant in time. Usually, this ideal placement changes continuously as the workload changes in time. Whenever the actual placement is not ideal, there is a benefit in reorganization. Unfortunately, there is a cost involved in reorganizing. Obviously, reorganization should take place only when the benefit outweighs the cost [CABK88]. Though there has been much work in the area of online reorganization in the recent years In [WZS91 & SWZ93], the authors present an online method for the dynamic redistribution of data, which is based on reallocation of file, fragments. A limitation of their study is that they do not consider index modification. Perhaps [SD92] is the first paper that discusses a solution for online index reorganization. They outline the issues involved in changing of all references to a record when its primary identifier is changed due to a record move. The techniques in the [SD92, ZS96] are limited to centralized DBMS and require the use of locks, where using locks during reorganization can degrade performance significantly [AON96].

In [AON96] they examined the problem of online index reorganization. They present two alternatives for

performing the necessary index modifications, called one-at-a-time OAT page movement and BULK page movement. The OAT moves one data page at a time, and modify the indexes for records in that data page. While the BULK makes a copy of the entire chunk of data (typically 16 or 32 pages) that is to be moves at the destination node, and then modify the indexes at the source and the destination node. While these algorithms are extremes on the spectrum of the granularity of data movement, they both use the conventional B+-tree algorithms for insertion and deletion. This dependency on the conventional B+-tree algorithms slowdown the migration process and consequently, affect the system performance. To minimize the index modification cost, in [LKOT99] they suggest the Fat-Btree as a powerful search structure that supports the data reorganization and speeds up the migration process. While range partitioning can efficiently support range and exact match queries, it can lead to data skew where certain values of the key-range attribute occur more frequently than other values. To solve this problem, they base their strategy on the "Disk Cooling" algorithm [SWZ93]. Although, the strategy is simple and has better control when multiple PEs are overloaded, but there is no guarantee for distributing the given load as evenly as possible. By taking the advantages of the Fat-Btrees and a new proposed heat balancing strategy, we propose an efficient strategy to reorganize the data online with minimal cost of modifying the indexes.

3. The System Distributed Search structure

We assume that data are initially range partitioned across all the system PEs so that the access method can associatively access data for strict match queries, range queries and cluster data with similar values together. Using a B-tree based index enables more efficient processing of range queries than a hashed index, where only the nodes containing data in the specified range are accessed. One solution to associative access is to have a global index mechanism replicated on each PE [OV91]. The global index indicates the placement of a record onto a set of PEs. Conceptually, the global index is a two-level index with a major clustering on the PE range and a minor clustering on some attribute of the relation (see Fig. 1.A.). The first level directs the search to the PE wherein the data is stored. This indexing layer is essentially a partitioning vector with n-1 and n "pointers" for a system of n PEs. This layer can be cached in main memory for fast access and it is duplicated across the system PEs to ensure that there is no central PE through which all accesses must pass. Since this indexing level is often read but rarely updated, the maintenance of the copies of this layer is hardly be required. The second level of the index is a collection of Fat-Btrees, one at each PE, each Fat-Btree independently indexes the data at its PE. The Fat-Btree is basically B+ tree with additional properties:

1. The root node can be a fat node, i.e., for a B+-tree of order d (and maximum of 2d entries), the root can contain more than 2d entries.
2. From (1), the height of a Fat-Btree can be designed to a prescribed value.
3. The tree is able to exploit the bulk-loading mechanism.

If the height of the Fat-Btree at the migration source and destination are the same, then the amount of data to be migrated correspond to the entirety of one or more branches of the Fat-Btree at the source PE [LKOT99]. So that, it would be easy to prune the entirety of the branches from the Fat-Btree at the source PE as well as attaching these branches into the Fat-Btree at the destination PE using bulk-loading technique without excess overhead. Thereby, it is required to design all the Fat-Btrees at the same height, which is essentially determined by the PE with the fewest number of records. We adopt our reorganization strategy on the advantages of the Fat-Btree in speeding up the migration process and consequently minimizing the reorganization cost. We also treat an index branch as a unit of migration.

4. The Heat Migration Strategy

Online heat balancing is done in four basic steps: monitoring PE workload, exchanging this information between PEs if it is necessary, calculating new distribution and making the work migrating decision, and the actual data migration.

4.1 The workload

The system workload is reflected by a metric, called heat [CAB88]. The heat of an object is the access frequency of the object over some period of time. The following heat statistics may be collected continuously (or periodically) while the system is running:-

- **H (Pg)**: the heat of the data page; the access frequency to the page **Pg.**
- **H (Br)**: the heat of a branch in a Fat-Btree, i.e., the sum of the access frequencies of the data pages that belong to the branch **Br**.
- **H (Ft)**: The heat of a Fat-Btree, i.e., the accumulated heat of the branches of the tree.

Since every PE has only one Fat-Btree, thereby, the heat of a PE, H (PE), equals the heat of its Fat-Btree. Nevertheless, we can generally define the heat of a range R = {Rmin .. Rmax} as the access frequency of R over some period of time. A range R as a logical or abstract quantity could be achieved at any physical quantity such as a database page, an index branch, an index tree, and a PE, so that: *Heat (***R***) = Heat (***O***),*
where **O** = *{ data page, index branch, index tree, PE }* that holds the range **R** = *{Rmin.. Rmax}*.

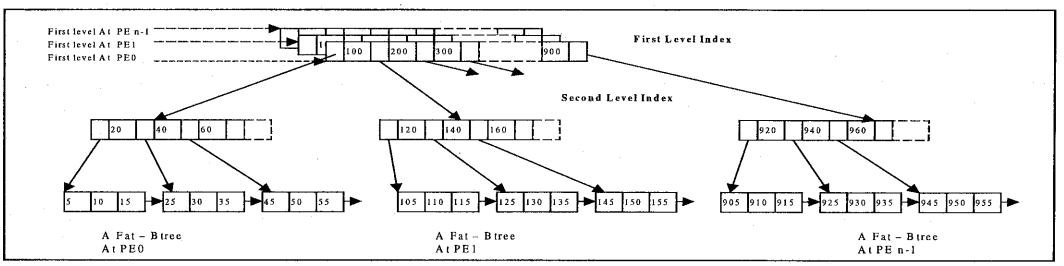


Figure 1.A: A sample global index structure for illustration.

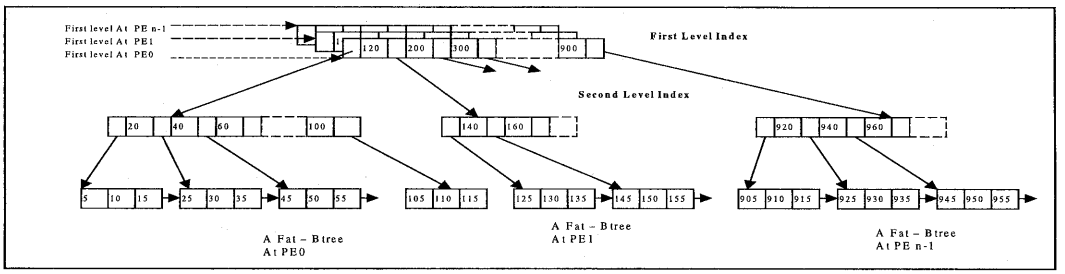


Figure 1.B: The consequent index structure after migration of data from PE1 to PE0.

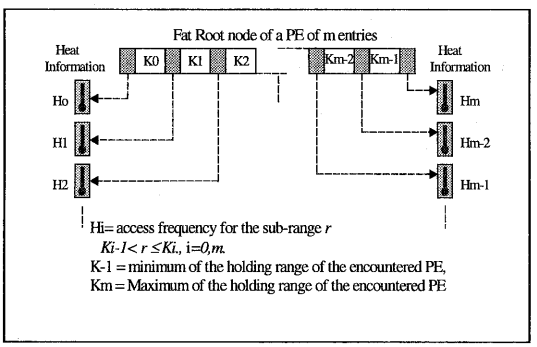


Figure 2: Heat statistics at the fat root node of a PE.

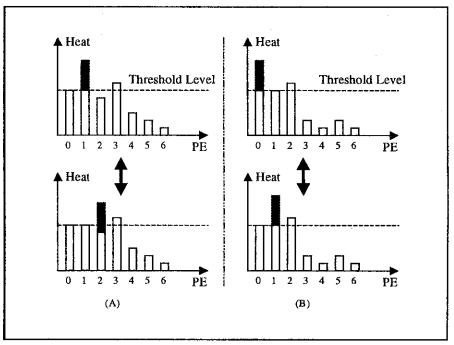


Figure 3: Examples of useless migrations

The complexity of maintaining statistics on a range R varies from maximal cost cases to minimal cost cases depending upon the physical quantity **O** that holds the range R. The maximal cost cases in their extreme fashion could be high if we maintain the heat statistics for every data page in the system, **O** = {data page}, which roughly requires maintaining statistics for every possible point in the whole range. Although this approach for the workload is very costly, but it has its own advantage in estimating an accurate figure of the workload. On the other extreme, the minimal cost cases in their extreme way could be achieved if we maintain the heat statistics for every tree (or PE) in the system, **O** = {PE}, which requires only information proportional to the number of the system PEs. Although this approach is not costly, but it has its own disadvantage in inaccurate estimation of the workload. On the middle of the spectrum, there are mid-cost cases, e.g., maintaining the heat statistics for every index branch in the system or even for every sub-tree at every root node in the system, O = {Sub-tree}. The main advantage of these mid-cost alternatives is; they may provide some compromise solution in term of cost and accuracy for measuring the system workload and migration cost. For sake of simplicity, we will focus on the minimal cost and mid-cost alternatives.

4.1.1. The Minimal Cost Workload.

One straightforward way for quantifying the workload is to consider the following scenario; if N queries are issued to the system, then, a PE*i* will receive Ni queries, from these N queries. The number Ni varies from a PE to another, depending on the PE holding range. One can

assume this Ni represents the heat of this PE*i* (and its Fat-Btree) during this interval of time. Moreover, if we assume the tree heat is uniformly distributed across its branches, then we can simply - with minimal statistics information - estimate the heat of every branch in the distributed Fat-Btree by using a top-down strategy.. By this way, we approximate the actual workload to the index pages workload with the advantage of minimal heat statistics and simple algorithms are needed.

4.1.2 A Mid-Cost Workload.

Instead of assuming uniform heat distribution at the Fat-Btree root node, we can additionally maintain the heat of every sub-tree at the root node of every PE in the system (see Fig. 2). Then, in order to minimize the heat statistics information, we assume a uniform heat distribution in the deeper levels (than the root). This assumption is based on; as we traverse a tree from the root to the leaves, as the range become narrower and the possibility of uniform heat distribution become higher. This approach has its own advantage in getting more accuracy in the workload estimation with a reasonable heat statistics information. As well, we can extent the strategy by maintaining the heat statistics information beyond the root node at deeper levels in the encountered tree. In our simulation, we consider this mid-cost approach. In principle, we consider the selection of the workload is a "design parameter" which depends upon the applications and their requirements.

4.2 Data Migration Strategy: Branches Migration

A source PE (PE*r*) can be defined as the PE from which the data pages (through the corresponding index branches) have to be moved to other PEs. Similarly, a destination PE (PE*d*) can be defined as the PE at which the data pages (and index branches) have to be stored. Recall that data is range partitioned, thereby we can only move data from one PE to its neighboring PEs (left or right or both), which hold the preceding or succeeding ranges. However, there are only two exceptions. The first exception deals with the "rightmost" PE, which has the capability of data migration only in the left direction, while the second exception deals with the "leftmost" PE, which has the capability of data migration only in the right direction. These migration directions represent the heat balance horizon.

Assume that we initiate the migration process between a PE*r* and a PE*d*, where each of these PEs has its own search structure. Traditionally, the migrated data are inserted one at time (one record at time, one page at time), which can be very costly especially if the number of records (or pages) is very large. The idea behind our migration strategy is as follows:

1. Since every branch in the search structure represents a unique sub-range form the whole range of the key attribute, so migration of branches (physical point of view) means migration of sub-ranges (logical point of view) from the PE*r* to the PE*d*. Every sub-range has its own access history (heat), therefore, if we balance the hot sub-ranges as even as possible across the system PEs, then, the system performance is higher.

2. The hot sub-ranges migration may be very costly as it includes index modification at the PE*r* and PE*d*. However, this critical cost could be kept minimal if both the PE*r* and PE*d* have flexible search structures that can support this kind of migration with minimal cost, so that both insertion at the PE*d* and deletion at the PE*r* can be carried out in simple and inexpensive way. Our objective with the search structures is just fulfilled with the distributed Fat-Btree as a powerful index structure that could support the sub-ranges migration with minimal cost.

The attachment of branches at the destination tree and detachment these branches at the source tree are essentially pointer updates (see Fig. 1.B.). The migration itself can be undertaken locally among adjacent PEs because of the features of the Fat-Btree. Therefore, the cost of controlling concurrency and synchronization is kept lower in the Fat-Btree. The amount of data to be migrated is obtained from the index branches at the source PE. By this way, the tree structure itself does not change by the branch migration. However, branches migration itself changes the data pages distribution at both the PE*r* and PE*d*. In the same time, it causes an update in the root node of the Fat-Btrees at both the PE*r* and PE*d*. Thereby, the migration of branches in the Fat-Btrees requires that the index entries in the first level node copies to be updated. During migration process the first level entries at the source and destination PEs are already updated. While the other copies at the other PEs are updated in a lazy manner by piggybacking update messages onto messages used for other purposes. In the same time, the 2-level index structure remains usable even some copies of the first level are not completely updated. Since if there is search, based on the old copy of the first level, then the search will direct to the source or destination PE, wherein the updated version of the first level.

5. The Heat Balancing Strategy

The heat balancing is an important factor that can determine the response times, speedups, and throughput of the system. The data reorganization should satisfactorily deal with this issue. Generally, the heat balance problem is closely related to scheduling and resource allocation, and can be static or online. A static strategy relates to decisions made a-apriori. On the other hand, it is important that heat balancing acts dynamically

by measuring the current system heat. To come up with such balancing, the balancing has to be aware of the periodic workload changes of the system PEs and needs to take into account the heat distribution during the data availability period. In this section, we present our online heat balancing strategy, which is based on the following two observations:

Observation (1): The useless migrations

In order to balance the given heat to the system through the access pattern, one may use the well-known "Disk Cooling" Algorithm (DCA) [WZS91 & SWZ93]. The DCA was introduced for balancing disk arrays, as an efficient general greedy algorithm, in which the hottest disk is first selected as the migration source, then, the coldest disk is selected as the migration destination. The process is repeated until the system is heat-balanced. If we apply this strategy to our system with its special characteristic in the migration directions. Then, the DCA procedure for selecting the coldest PE will be shorten as checking the right and the left neighbors of the hottest PE. The neighbor that has minimum heat will be selected as the migration destination. We refer to this strategy as the "maximum heat" strategy.

Obviously, the strategy is simple but it has some useless migration cases that could arise while the heat balancing process is running. Fig. 3 gives two examples for such unsatisfactory cases. For example, in Fig. 3.A, the maximum heat is at PE1, in order to balance its heat, its excess heat over the threshold level have is migrated to one its coolest neighbor, in this example PE2 is the coolest neighbor. However, after migrating the heat from PE2 to PE1, the maximum heat becomes at PE2 and its coolest neighbor becomes PE1 which in turn will repeat the whole process with inefficiency in distributing the given heat across the system PEs, even though there are many cold PEs in the system. Briefly, in the maximum heat strategy there is a missing mechanism to stop further migrations from arriving to the hottest PE again. Such mechanism becomes a need in a system like shared nothing system, in which the useless migration of large data dramatically degrades its performance. This observation leads us to consider a new strategy for balancing the given heat.

Observation (2): The Rightmost and the Leftmost PEs

Since the data are initially range partitioned across all the system PEs, thus the rightmost PE (RMPE) must be heat-balanced by one of the following two cases:

(1) If the heat at the RMPE is larger than the average (or threshold) heat value, then in order to balance this PE, it is essential to migrate its excess heat to its left neighbor PE (LNPE). The amount of migrated heat should be corresponding to the excess heat at this RMPE as far as we can.

(2) If the heat at the RMPE is less than the average value, then in order to balance this RMPE, it is essential to migrate a missing heat from its left neighbor PE (LNPE) to it. The amount of migrated heat should be also corresponding to the missing heat at the RMPE as far as we can.

In the first case, we called the migration as "direct migration", since we can directly initiate the migration process from the RMPE to the LNPE, so that the RMPE is heat-balanced. However in the second case, it depends completely upon the heat at its LNPE as:

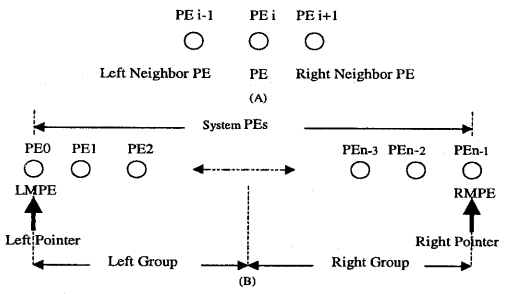
- If this LNPE has the missing heat at the RMPE; then we can directly initiate the heat migration from LNPE to RMPE, so that the RMPE is heat-balanced.
- If the LNPE has not the missing heat at the RMPE; then it is essential to get this missing heat – recursively- from some PEs in the system. We called the migration in this case a "stacked" migration, because at this moment, we can not initiate the migration from the LNPE to RMPE. However, whenever the LNPE gets this missing heat, we can initiate the migration. Therefore, we push the action of this type into a stack called the migration stack, by recording the action components: source, destination, and, amount of the missing heat.

With the same point of view, we can balance the leftmost PE (LMPE) and record its balancing cases.

From the above observation, assume we have a system of N PEs and we divide its PEs into two groups: left group and right group (see Fig. 4.B). We formulate the principal idea of our algorithm on balancing first (or to know the needed migrations) the system's RMPE and LMPE, and, if it is necessary, push the corresponding actions into the migration stack. Then, by dropping virtually these PEs from the system, there will be a new RMPE and LMPE for the system. Consequently, we can consider these new RMPE and LMPE as before and repeat the same procedure until the system virtually consists of only two PEs. At which, It will be so easy to migrate the heat between the right and left groups. Using the migration stack, we can pop the actions to balance the system PEs by sequentially traversing the right group from left to right and sequentially traversing the left group from right to left. The algorithm will be terminated in these traversing whenever the migration stack is empty.

The algorithm starts by sequentially traversing the right group from the right to the left and in the same time, sequentially traversing the left group from left to right. This suggests a variable pointer called "right pointer" initially pointed to the RMPE and decreases by "one" as it traverses from right to left. Similarly, sequential traversing the left group from left to right, suggests a pointer variable called "left pointer" is initially pointed to

the LMPE and increases by "one" as it traverses from left to right. During these sequential traverses, two variables called the "right excess heat" and the "left excess heat" are used to record the excess heat at any subsequent traversing in the right and left groups, respectively. Fig.4.C gives a high level description of this algorithm.




```
HeatType BalanceProcessor( PE, neighbor, ExcessHeat )
// Balance the given PE with its neighbor
// identify the source, destination, migrated heat, and type of migration
// (direct or stacked).
CurrentAction={Source,Destination,MigratedHeat,Type }
if(CurrentAction.Type=="Direct Migration")
    StoreMigartion ( CurrentAction);
else PushMigration ( CurrentAction );
// then Calculate the new excess heat
void     StackedMigration()
{ While (! EmptyStack()) {
    CurrentAction = PopMigrationAction();
     StoreMigration( CurrentAction ); }
}
void    HeatBalancingStrategy(PEMin, PEMax)
{ ClearMigrationStack(); // clear migration stack
  RMPE=PEMax; // Right pointer = Rightmost PE
  LMPE=PEMin;  // Left Pointer = Leftmost PE
  REH=0; // Right excess heat
  LEH=0; // Left excess heat.
  Done=0;
  while(!Done){
   LEH=BalanceProcessor(LMPE,1,LEH);
  if(RMPE<=LMPE) Done=1;
  else{ REH=BalanceProcessor(RMPE,-1,REH)
     RMPE--;
     LMPE++; }
StackedMigration();}
```

(C)

Figure4: (A), (B) Algorithm notation, (C) A high level description of the proposed algorithm.

By applying our heat balancing strategy, it can build a migration "directory" in which the sequence and all required migrations to be done so that the system is heat balancing are stored. From this migration directory, we can additionally tune the migration decisions as follows:

- If the total amount of heat in the migration directory to be moved is below a threshold, then this directory can be dropped, since this may indicate that the system is almost balanced.
- It can apply a profitability analysis, a trade-off between the benefits of moving work to balance heat and the cost of work movement. So that some migrations in the directory may be prioritized or postponed or dropped.

6 Simulation Result

In this section, we describe our experiments to study the performance of the online data reorganization with two heat balancing strategies: "the maximum heat" and our strategy. The metric used is the impact on the response time of queries. Table 1.0 shows the major system, database and query configuration parameters with their default values and variation settings. We use a functional shared nothing parallel database system where each PE has its own disk(s) and memory. The system PEs communicate with each other by exchanging messages across the interconnection network, set at 120 Mbit per second, the communication bandwidth is hardly a bottleneck during reorganization, therefore, the communication delay is set at 2 msec. We first create an initial Fat-Btree with the tuple key values generated using a uniform distribution. Then we generate 2,000,000 range queries using Zipf-distribution, which concentrates the queries in a narrow key range. These queries are generated with skew that defined by the skew factor (τ) of Zipf-distribution. Therefore, there are more range queries issued at one PE than the other PEs, depending on the skew factor τ. The heat skew will initiate the migration of branches between the PEs, depending on the running heat balancing strategy. We model each of the PEs as a resource and the queries as entities. We assume the heat balancing is done in centralized scheme and it is initiated after every 1000 queries.

Table 1: the parameters and their used values.

Parameter	Default values / variation
System Parameters:	
Number of PEs in the cluster	16 /32
Index node size	4K page
Network bandwidth	120 Mbits/s
Time to read or write a data page	8 ms
Database Parameters:	
Number of records	1 million
Key size	4 bytes
Query Parameters:	
Number of queries	2,000,000
Zipf distribution decay factor	0.2 / 0.5/ 0.1 → 0.9
Mean arrival rate	20 per second / 5.0 → 60.0
Mean service time	500 ms

We study the effect of heat balancing on the response time. Figure 5 compares the average response time without and with heat balancing. It shows the effectiveness of heat balancing in reducing the average response time and increasing the system bandwidth. We observe that the response time of a query in the "hot spot" PE differs greatly from the system response time. Figure 6 shows the effect of migration on the response time of the "hot spot" PE. These figures affirm the effectiveness of distributing the given heat as even as possible, on the

system response time and that of the "hot spot" PE, by migrating some index branches and hence data pages from the hot PEs to the cold PEs. In addition, Figure 7 shows the scalability on the average response time with a cluster of 32 PEs.

In order to evaluate the performance of the considered strategies; we first focus on the "hot spot" PE and record a measure, CD, which is the total migration cost (time), whenever the "hot spot" PE is a migration destination. We define a term called "average useless migration per second" which is CD divide by the time required for executing the considered queries (experiment time). Intuitively, this measure should be equal zero since the "hot spot" PE should not be a migration destination. Any migration that involves this PE as destination may considered as a useless migration. Figure 8 shows the "maximum heat" strategy, as a result of the missing mechanism to stop further migrations from arriving to the hot spot" PE again, has a high figure of useless migration as the skew increases and the number of PEs decreases. Our strategy does not record any cases of useless migrations. Figure 9 shows the migration cost per second against the skew factor of our strategy for clusters of 8,16 and 32 PEs. The result affirms that as the number of PEs increases and as the skew increases the migration cost increases, as a result of distributing the given heat as even as possible between the system PEs. In order to cover the variation of the skew factor, all the experiments of the skew factor variation are carried out at a low arrival rate of 5 per second.

7. Conclusion

In this paper, we have proposed a heat balance model including data migration with minimal cost of index modification on a shared nothing system that can be employed in many practical database applications. Data reorganization is necessary because the optimal data placement changes with time and usage of the parallel database system, and significant performance improvements can be obtained by reorganization. Data reorganization has been studied before, bot no work addresses the critical issue of balancing the given heat as even as possible across the system PEs with minimal index modification. Thus decreasing the response time and increasing the throughput of the system. This paper fills an essential void.

References

[AON96] K.J. Achyutuni, E. Omiecinski, and S. B. Navathe. Two techniques for On-line Index Modification in Shared Nothing Parallel Databases. Proceedings ACM SIGMOD 1996.

[CABK88] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data Placement in Bubba. *In Proceedings of ACM SIGMOD Conference, pages 99-108,1988.*

[LKOT99] M.L. Lee, M. Kitsuregawa, B.C. Ooi and K.L. Tan. Online Reorganization in Shared Nothing Systems. 1999.

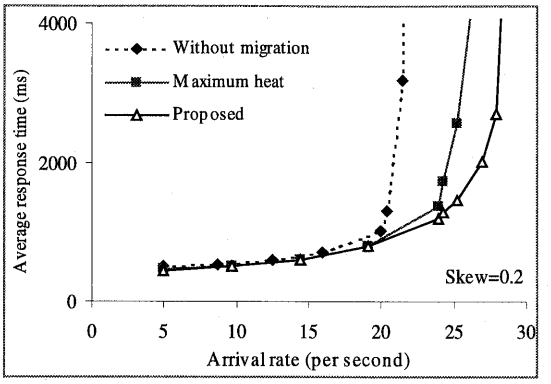
[OV91] M.T. Ozsu and P. Valduriez. Principles of Distributed Database Systems, *Prentice Hall, 1991.*

[SD92] B. Salzberg and A. Dimock. Principles of transaction-based on-line reorganization. *Proceedings of the 18th International Conference on Very Large Databases, pages 511-520, 1992.*

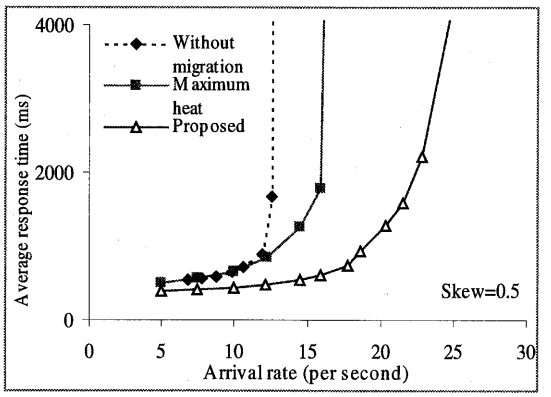
[SWZ93] P. Scheuermann, G. Weikum, P. Zabback. Adaptive Load Balancing in Disk Arrays. *In Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms (FODO), 1993.*

[WZS91] G. Weikum, P. Zabbak, and P. Scheuermann. Dynamic file allocation in disk arrays. *In Proceeding of the ACM SIGMOD Conference, 1991.*

[ZS96] C. Zou and B. Salzberg. On-Line Reorganization of Sparsely-Populated B+ Trees, *Proceedings ACM, pages 115-124,1996.*

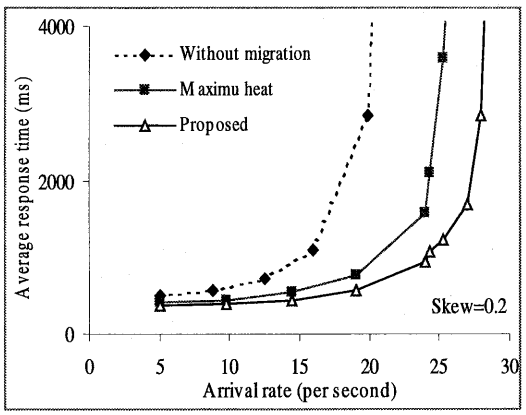


(5-a)

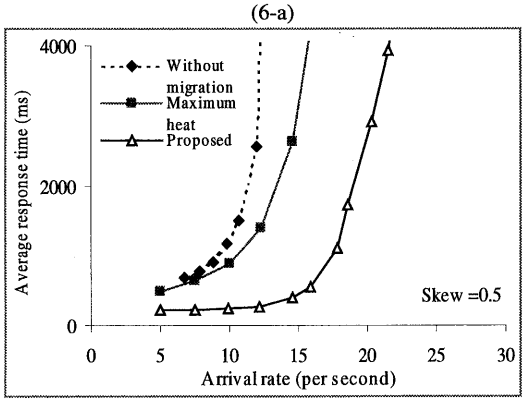


(5-b)

Figure 5: Effect of heat balancing on the average response time of 16 PEs cluster. (a) Skew=0.2, (b) Skew=0.5

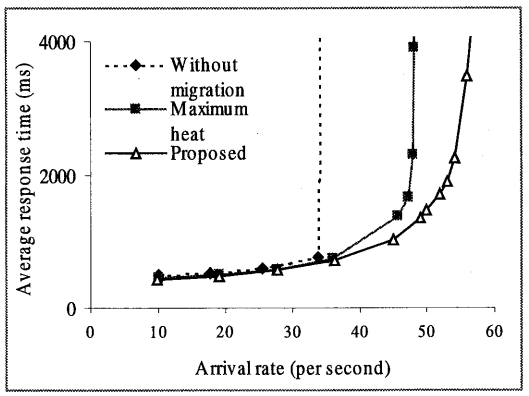


(6-a)



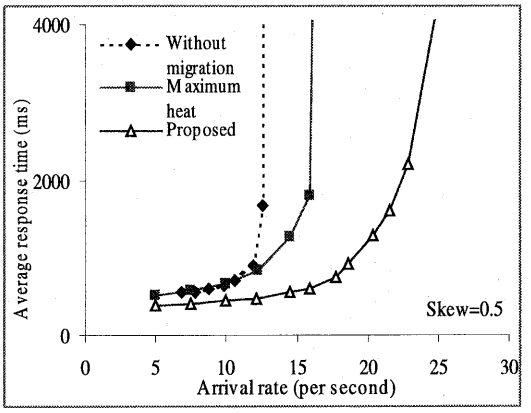
(6-b)

Figure 6:Effect of heat balancing on the average response time of the "hot spot" PE in a 16-PEs cluster.



(7-a)

Figure 7: Effect of heat balancing on the average response time of a 32-PEs cluster, (a) Skew=0.2, (b) Skew=0.5.



(7-b)

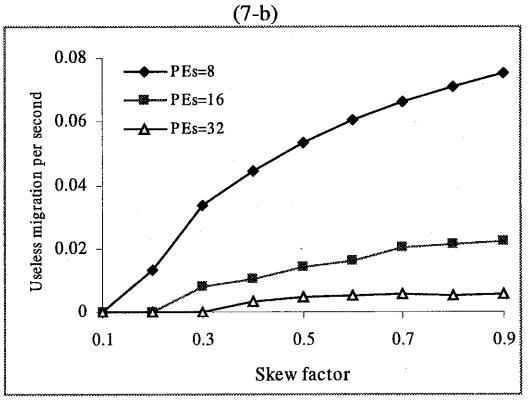


Figure 8: The average useless migration cost per second of the maximum heat strategy with the skew variation.

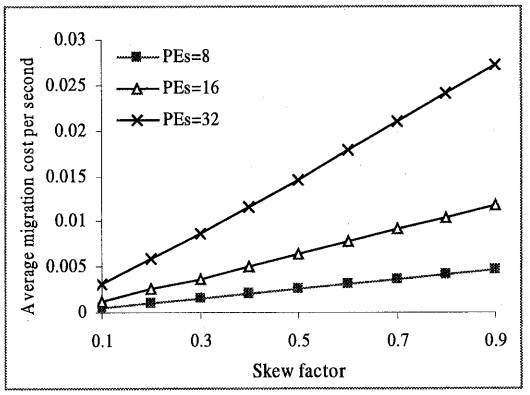


Figure 9: The average migration cost per second of our heat balancing strategy with the skew variation.