A Fast Convergence Technique for Online Heat-balancing of Btree Indexed Database over Shared-nothing Parallel Systems

Hisham Feelifl[1] Masaru Kitsuregawa[1] Beng-Chin Ooi[2]

[1] Institute of Industrial Science, The University of Tokyo
3rd Dept., 7-22-1 Roppongi, Tokyo, 106-8558, Japan
{hisham, kitsure}@tkl.iis.u-tokyo.ac.jp
[2] Department of Computer Science, National University of Singapore
Kent Ridge, 119260, Singapore
ooibc@comp.nus.edu.sg

Abstract. In shared-nothing environments, data is typically declustered and indexed across the system processing elements (PEs) to achieve efficient processing. However access patterns are inherently dynamic and skewed, thus, data reorganization based on the data access history (heat) is essential and should be done online. While the data is being reorganized, indexes need to be modified too, therefore, reorganization should additionally deal with the index modification. Based on minimization of index modification, we propose a data reorganization technique over a shared-nothing parallel system. By finding the exact work that should be done, the technique can smoothly balance a given heat across the PEs as fast as possible, if it is required. By tuning its parameters, it can cover a wide range of balancing requirements. We evaluate its performance through simulation studies. Its effectiveness is clarified quantitatively.

1 Introduction

The explosive growth of data volume in various fields such as the Internet, Web, and, data warehouse increases the need for fast response. A shared-nothing parallel architecture is one of the typical examples to achieve such response [4, 13]. As demonstrated by the existing machines such as Bubba [2] and NEDO 100 Node PC Cluster [10], shared-nothing architectures can provide fast response at low cost, high extensibility and availability. However, shared-nothing architectures suffer from load balancing problems. Load (heat) balancing is difficult to achieve, compared with shared-disk architectures, because it relies on the effectiveness of database partitioning for the query workload [13]. To achieve efficient query (and transaction) execution, data is typically declustered across the PEs, and, indexed at each of the PEs. However, access patterns are inherently dynamic and skewed which can lead to performance degradation as some PEs become hotspots (frequently accessed) while many other PEs are cold (infrequently accessed). Thus heat-balancing is essential.

Heat balancing is particularly challenging for evolving workloads, where 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 [8]. As the data is moving from hotspot PEs to cold PEs, the

corresponding indexes have to be modified too. Thus, data reorganization should also deal with the index modification [1].

In this paper, we propose a new technique to facilitate more efficient data reorganization. It is based on index-modification minimization, where the amount of data to be migrated (migration unit) corresponds to the entirety of one or more index branches at a source PE. In this case, it would be easy to prune the entirety of index branches from a source PE tree as well as attaching these branches into a destination PE tree using bulk-migration technique [5]. To distribute a given heat across the PEs, we introduce a new heat-balancing algorithm that is distinguished from the existing algorithms in several respects. It provides the exact solution to balance a system without any heuristic mechanisms. Thus it can distribute a given heat as evenly as possible across the PEs and as fast as possible, if it is required. Because of its exact solution, its convergence is guaranteed so that it can be employed in applications in which fast responses and fast balancing (adaptation) are relevant requirements. Furthermore, we support the technique by parameters that can be tuned to fit a wide range of balancing requirements in terms of heat distribution and balancing speed over a wide range of access pattern skew.

The next section details the related work. Section 3 describes the underlying index and the migration unit. Section 4 introduces a new heat-balancing algorithm. Sec. 5 deals with the experimental study and Sec. 6 concludes the paper.

2 Related Work

Recently, there has been much work in the area of online reorganization. [8] presents an efficient online method for the dynamic redistribution of data, however it does not cover index modification during reorganization. [7] outlines the issues involved in changing all references to a record when its primary identifier is changed due to a record move. The techniques of [7, 12] are proposed for centralized DBMS and require the use of locks, where using locks during reorganization can degrade performance significantly [1]. [1] presents two alternatives for performing the necessary index modifications, called one-at-a-time OAT page movement and BULK page movement. However, both techniques depend on the conventional Btree algorithms that can lead to considerable index-modification cost [5]. [11] suggests using the Fat-Btree structure to speedup migration issues so that index can be modified with minimum cost. However, the objective is to balance the number of pages across the PEs (space balancing) rather than heat balancing. Access pattern skew can lead to performance bottleneck even though there is a space balance [5, 13]. [5] assumes heat balancing but without consideration for the balancing speed (the speed of the system to adapt itself to an access pattern), where fast balancing (adaptation) is the main requirement for the most advanced applications, e.g. WWW servers. The balancing algorithm of both [5] and [11] is simply the disk-cooling algorithm [8] that can lead to a long convergence time and in some cases unstable situations as a result of its local view while balancing a system (see Sec. 4). In contrast, we avoid long convergence and unstable cases through an algorithm that utilizes the range-partitioning strategy and can find the exact solution in one step

without any heuristic mechanism so that the system is heat-balanced. By introducing balancing parameters into its solution, it can also cover a wide range of requirements.

3 The Global Index

We assume that data is initially range partitioned across all the system PEs so that the access method can associatively access data for strict match queries, range queries and can cluster data with similar values together. Using a B-tree based index enables more efficient processing of range queries compared to a hashed index because in the former only the nodes containing data in the specified range are accessed. One solution to associative access is to have a global index mechanism [6], conceptually, the global index is a two-level index with a major clustering on the PE's key range and a minor clustering on sub-ranges of the key range. This non-overlapping data partitioning gives also non-overlapping indexes [9], so that the first–level index can be implemented as a partitioning vector with entries number equals the PEs number. To ensure that there is no central PE, the first-level index is further replicated in all the PEs [5]. While the first-level index directs the search to the PE wherein the data is stored, the second-level index is basically a collection of Btrees, one at each PE; each Btree independently indexes the data at its PE [5, 9, 11].

We assume the tree height at each of the PEs is the same, so that the amount of data to be migrated corresponds to the entirety of one or more branches of the Btree at a source PE. The attachment of branches at a destination tree and detachment these branches at a source tree are essentially pointer updates so that index is modified with minimum cost. This branch-migration technique does not change the tree structure, but it changes the record distribution at a source and a destination PE. It causes an update in the root node of the Btrees at these PEs, which in turn requires the first-level index copies to be updated. This is can be done in a lazy manner by piggybacking update messages onto messages used for other purposes [5].

Since it is common to cache a part of the index in memory to accelerate access and cost per bit is declined, then it is possible to use large memory while data is migrated from a PE to other PE. Using of large memory for access method accelerates access, for example, [9] assumes that all the nodes of a Btree are placed in memory. Thus, if a root at a PE will overflow due to insertion of some branches (as a result of migration) then instead of having a physical fat structure the branches (and the corresponding data pages) that lead to fatness can be temporarily stored in memory. This saves considerable cost of I/O operations if the amount of migrated data is large, which can also lead to speed up migration issues among the PEs. To demonstrate this point, assume we have 4 PEs with the following key ranges: PE0 is assigned to hold 1-25, PE1 26-50, PE2 51-75, and PE3 76-100. If there is a migration decision that migrate some branches of key range say 15-25 from PE0 to PE1 and if the root node at PE1 is full, then it is possible to store these branches (and the corresponding data pages) into the memory of PE1. Then if it is required to migrate some branches from PE1 to PE2 of key range say 40-50, thus there will be rooms to accommodate the branches of key range 15-25 into the root of PE1 without need for being physical fat. While the branches of key range 15-25 are being stored in the memory of PE1, there is a

possibility to migrate these branches (or some of them) from PE1 to PE0, if it is required. Since there is no disk access, the corresponding migration cost is dominated by the communication cost between PE1 and PE0 and thus it speeds up migration issues between PE1 and PE0. Such buffering effect at PE1 can be generalized across the PEs so that it can provide fast access, and, migration at each of the PEs. Therefore using the PE memory if the fat condition is occurred at its root can reduce the cost of having a physical fat structure at a PE. The fatness property of the underlying index can be viewed as a temporary status that could be occurred at some PEs while data is being reorganized. Without loosing the generality we assume that the underlying index is basically Btree that can support bulk-migration without essence of being physically fat. However, to evaluate the performance of the proposed technique without any benefits due to buffering effect at each of the PEs, in our simulation we select the physical Fat-Btree structure [11] as the underlying index. So that the structure with its physical fatness represents the worst case of our reorganization.

4 Online Heat Balancing

Online heat balancing is done in four basic steps: monitoring PE workload, exchanging information between PEs, calculating new distribution and making the work migrating decision, and the actual data migration. In this section, we first clarify our consideration to the workload. Then we present a new algorithm to calculate a new heat distribution from the current one

The workload is reflected by a metric, called *heat* [3]. We define the heat of a range R = {Rmin .. Rmax} as the access frequency of R during a certain period of time. A range R as a logical quantity can be determined by any physical object in the system such as a data page, an index branch, and an index (sub)tree. The cost of maintaining heat statistics on R is dependent on its physical object. The highest cost can occur, if we maintain statistics for each of the data pages. This roughly requires maintaining statistics for every possible point in a given range. The minimal cost could be achieved if we maintain statistics for each tree (PE) in the system, and it requires information proportional to the number of PEs. Although it is simple, but it gives inaccurate estimation in the workload. There are mid-cost approaches, e.g., maintaining heat statistics for each index branch or for each sub-tree at a root node. These approaches give a compromise solution in terms of cost and accuracy. In our simulation, we use one such mid-cost approach in which heat statistics information is maintained for each sub-tree of a root node at a PE. To minimize the required information, uniform heat distribution is assumed in the deeper levels. In principle, we assume the workload estimation is a “design parameter” that depends on the applications and their requirements.

4.1 The Full-window Algorithm: Algorithm Basic

Since data is range partitioned across the PEs, we can only move data from one PE to its neighboring PEs which hold the preceding or succeeding ranges. This migration

rule has two exceptions, the first deals with the *rightmost* PE, RMPE, which can only migrate data with its left neighbor, while the second deals with the *leftmost* PE, LMPE, which can only migrate data with its right neighbor. Two main observations formulate the proposed algorithm; the first one is related to using the disk-cooling algorithm (DCA) [8] with the assumed range partitioning strategy, while the second is related to the migration exceptions at the RMPE and the LMPE.

Observation (1): Instability of the DCA.

To balance a given heat across the PEs, the existing techniques, such as [5, 11], use the DCA as the balancing algorithm. The DCA was introduced as an efficient general-purpose algorithm for balancing disk arrays. In the DCA, the hotspot disk is first selected as the migration source, and, the coldest disk is selected as the migration destination. By making this decision, it generates a new hotspot disk, if there is, at which the process can be repeated until all disks are heat-balanced. If we apply the DCA to the PEs with the assumed range-partition strategy, then, the procedure for selecting the coldest PE will be shorten as checking the right and the left neighbors of the hotspot PE. The colder neighbor will be selected as the migration destination.

The algorithm is simple but it has some unsatisfactory cases that can be occurred during balancing. For example, assume a system of 4 PEs with a heat distribution of (PE0: 200,PE1: 50, PE2: 115, PE3: 35) and a threshold heat of 110 (10 % above the average heat). The first scan gives PE0 as the hotspot, and heat of 90 is migrated to PE1 at which the heat will become 140. The second scan gives PE1 as the hotspot and its colder neighbor is PE0. Heat of 30 is migrated from PE1 to PE0. Thus some heat is returned back to PE0 by which the balancing process will be entered into endless loop and most of the work is consumed in useless migrations between PE0 and PE1 without distributing the given heat to other cold PEs. Such instability case occurs because the heat of PE2 is higher than the threshold heat which in turn blocks heat migration to other cold PEs, e.g. PE3. With the assumed range-partitioning strategy the DCA does not have a mechanism that can stop such useless migrations or a special mechanism that can deal with heat distributions that contain some blocking cases as in the given example. Such mechanism is important in shared nothing environments, where migrations of large data without any beneficial effect dramatically degrade their performance. Therefore, the DCA can not provide a fast balancing with full guarantee in its convergence and in general its local view to a system can lead to long convergence which slowdowns a system to adapt itself to a given access pattern. The observation gives the motivation to develop a new algorithm so that fast balancing can be achieved without doubt in its convergence and its convergence can further be tuned to fit a range of requirements.

Observation (2): The Rightmost and the Leftmost PEs.

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

1. If the heat at the RMPE is larger than the average heat, then it is required to migrate its excess heat to its left neighbor PE (LNPE).

2. If the heat at the RMPE is less than the average heat, then it is required to achieve its missing heat from its LNPE. .

In the first case, we refer to migration as a *direct migration*, since we can directly initiate heat migration from the RMPE to its LNPE, while in the second case, it completely depends on the heat at its LNPE as follows:

2.1 If its LNPE does not have such missing heat then it is required to achieve heat from other neighbors to the LNPE, which in turn suggests a recursive approach. We refer to this migration as *stacked migration*, because we can not initiate migration unless its LNPE has such heat. We push this information into a stack called the *migration stack* by storing its components: *source*, *destination*, and, *missing heat*.

2.2 If its LNPE has this missing heat; then migration can be directly initiated from the LNPE to the RMPE, and therefore we have a *direct migration* case.

The above cases (from 1 to 2) find exactly the decision to balance the RMPE, and, they can be used also to find exactly the decision to balance the LMPE.

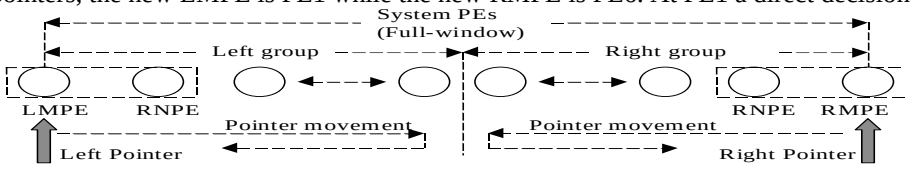
Based on this observation, assume we have a system of N PEs and we divide the PEs into two groups: left group and right group, see Fig. 1. We formulate the idea by first considering the system's RMPE (LMPE) with its LNPE (RNPE) and recording the proper decision to balance it, and, if it is necessary push the generated decision into the *migration stack*. Then, by dropping virtually these PEs from the system, there will be new RMPE and LMPE at which we can find the proper decisions to balance them with their neighbors as before, and therefore the process can be repeated until the system virtually consists of two PEs. At this point, it is easy to know exactly the migration decision that should be taken between the left and the right groups. Using the *migration stack*, we pop decisions by sequentially traversing the right group from left to right direction, and, the left group from right to left direction. Note that an empty stack indicates the definite algorithm termination. During traversing the PEs we store a generated decision into a structure called the *migration table* so that we can additionally know from the table what is the decision sequence to balance a system. Obtaining such sequence will help global balancing schemes in their scheduling to migration jobs while balancing a system.

Fig. 1 gives the algorithm notation, mechanism and high-level description. It shows two pointers called *right pointer* and *left pointer*, one for each group. The *right pointer* is initially pointed to the RMPE and it traverses its group from the right to the left direction. While the "left pointer" is initially pointed to the LMPE and it traverses its group from the left to the right. During these traverses, the current excess/missing heat at each pointer (PE) is recorded as well.

Case example

Assume a system of 8 PEs with the following heat distribution PE0: 200, PE1: 50, PE2: 115, PE3: 25, PE4: 260, PE5 20, PE6: 50 PE7 30. Assume further it is required to balance the system so that the heat at each of the PEs = 100 (average heat). The algorithm starts with its left pointer is pointed to PE0 while its right pointer is pointed to PE7. At PE0 a direct decision is generated as (source=PE0, destinnation=PE1, required heat =100) and the excess heat at PE1 becomes 50. At PE7 a decision of (PE6, PE7, 70) is pushed in the migration stack because PE6 does not have the

missing heat of PE7 (70). The excess heat at PE6 becomes –120. By advancing both pointers, the new LMPE is PE1 while the new RMPE is PE6. At PE1 a direct decision




```
HeatType GetDecision(PE, Neighbor,ExcessHeat)
 // Balance the given PE with its neighbor. Identify
 // the generated decision by source, destination,
 // excess heat & type of decision (direct or stacked)
 if(Decision.Type=="Direct") Store(Decision);
 else PushDecision(MigrationStack,Decision);
 Calculate the new excess heat and return it.

void StackedDecisions(MigrationStack, MigrationTable)
  while(! EmptyStack(MigrationStack))
    Decision = PopDecision(MigrationStack);
    Store(Desicion); // in the migration table

Algorithm Full_Window (PeMin,PeMax)
// PEmin and PEmax covers the system PEs..Full window
   RP=PeMax;// Right Pointer pointed to the RMPE
   LP=PeMin;// Left Pointer pointed to the LMPE
   RightExcess=LeftExcess=0 //reset excess heats
   MigrationStack=CreateStack();Done=0;
   while(!Done)
    LeftExcess=GetDecision(LP,LP+1,LeftExcess);
    if(RP<=LP)Done=1;
    else{RightExcess=GetDecision(RP,RP-1,RightExcess);
         LP++;RP--;}
   StackedDecisions(MigrationStack,MigrationTable);
```

Fig. 1. The full-window algorithm: its notation, mechanism, and high level description.

Table 1. The generated migration table for the given example

Decisions Sequence	Source	Destination	Required Heat	Comment
1	PE0	PE1	100	Direct
2	PE2	PE3	50	Direct
3	PE2	PE3	65	Direct
4	PE4	PE3	10	Direct
5	PE4	PE5	200	Direct
6	PE5	PE6	120	Stacked
7	PE6	PE7	70	Stacked

of (PE1, PE2, 50) is generated and the excess heat at PE2 becomes 65. At PE6 a stacked decision is generated as (PE5, PE6, 120) and the excess heat at PE5 becomes –200. By advancing both pointers, a direct decision of (PE2, PE3, 65) is generated

and the excess heat at PE3 becomes -10. A direct decision of (PE4, PE5, 200) is generated and the excess heat at PE4 becomes 10. The next step gives a direct decision of (PE4, PE3, 10). Since each pointer completes its traverse to its group, then it uses the *migration stack* to extract the pushed decisions so far. Sequential pop operation completes the sequence to balance the system as given in Table 1. These pop operations are equivalent to traversing the PEs in the opposite direction to that of the first traverse, see Fig. 1. If this migration table is issued to the system, then in one step it distributes the given heat as evenly as possible across the PEs.

The full-window algorithm is basically distinguished from the DCA in two respects; its utilization to the assumed range-partitioning strategy, and, its global view to a system. Both give the exact solution to balance a system without any possibility of unstable cases while balancing. Consequently it gives the chance to do the required work as fast as possible (or as required) with full guarantee in its convergence. The above algorithm can be also extended in many ways, for example, instead of considering the whole PEs (full-window), it possible to derive other derived algorithms that based on some PEs (partial-window) which supports local balancing schemes rather than the global ones. We will consider such partial-window algorithms with their features and advantages in a future work.

4.2 The Migration-workload Parameters: Speed and Distribution

If a *migration table* generated by the above algorithm is issued to the given system then in one step it balances the PEs as evenly as possible. This one-step reorganization does the whole work in a short time that may be accepted by some requirements and rejected by the others, depending on their acceptance to its effect. Although the one-step reorganization gives the capability for the fastest adaptation to access patterns, but it can lead to slow responses (during reorganization) at some PEs at which there are large amount of data movement and high arrival rates of users' queries. During reorganization, users of such PEs are considered as the victims of the fastest reorganization. The number of victims increases as the skew of access patterns increases. This can be considered as the main disadvantage of the one–step reorganziation. Many requirements usually give some threshold in the required heat distribution and they also allow balancing in incremental ways rather than that of the one step. Thus, we view the requirement space as a 2-dimentional space of two parameters; one represents the speed requirement and the other represents the heat-distribution requirement. The current objective is to introduce such parameters in the balancing process so that migration decisions are correlated to a requirement.
Assume a migration entry of the *migration table* can be represented by:

```
typedef    struct {PeType   Source;
  PeType   Destination;
  HeatType RequiredHeat;
  void*    Others;} MigrationTableEntryType;
```

Since a system migration workload is proportionally related to the summation of the component "*RequiredHeat*" across the *migration table*, thus, controlling this component by some parameters (requirements) gives a direct correlation between requirements and migration decisions.

First we consider the speed parameter, α, we introduce this parameter into the component "*RequiredHeat*" by the following normalization (transformation): "*RequiredHeat*"= α * "*RequiredHeat*", $0 \leq \alpha \leq 1.0$. So that for $\alpha=0$, it implies dropping (or postponing) the encountered entry, while for $\alpha=1.0$ it implies a full acceptance/speed for such entry. For $0<\alpha<1.0$ it implies we divide the current entry (job) into M small sub-jobs, where $M=1/\alpha$. If these M sub-jobs are issued one by one in periodic ways or whenever it is possible, then we achieve balancing in incremental ways, which certainly is reflected on a system speed to adapt itself to an access pattern. Although incremental balancing can lead to slow adaptation but it partitions the whole work into small jobs so that we can avoid the disadvantage of the one-step reorganization, especially under highly skew environments. Thus in general, any migration job can be postponed or incrementally issued or completely issued to a system depending upon its assigned α. For the sake of simplicity we select to unify α across the *migration table* so that one value of α gives one effect on the balancing speed (and the system response). Since the steady state of the M-step reorganization should be equal to that of the one-step as a result of its integration effect, then we can satisfy a heat distribution requirement (steady state) over a range of α.

Second we consider the heat-distribution parameter, ζ, we assume requirements on heat distribution are given as; balance a system so that heat at each of the PEs does not exceed some threshold value, $(1+ \zeta\%)$ average heat. Note that the *migration table* has been constructed so far with the assumption of $\zeta=0$. Because a ζ requirement gives the maximum allowable heat at hotspot PEs, we process the *migration table* by focusing on hotspot PEs. By picking up all the PEs that have heat higher than the threshold heat and modify (adjust) the component "*RequiredHeat*" so that the resultant heat at each of these PEs does not exceed the threshold heat. Then simulating the migration effect on the modified entries will generate new hotspot PEs by which the process can be repeated until a ζ requirement is satisfied across the *migration table*. This heat modification procedure can be sequentially done by; picking up the current hotspot PE and extracting a sub-table from the current table so that the sub-table contains all entries in which this hotspot PE is a source or a destination. We modify the "*RequiredHeat*" components in this sub-table so that the resultant heat of the current hotspot PE does not exceed the threshold heat. Simulating the migration effect in the current sub-table generates a new hotspot PE at which the process can be repeated until all entries in the migration table are modified according to the given ζ. The modification of the *"RequiredHeat"* component across the migration table implies also dropping some entries at which their sources are balanced in term of the given ζ requirement. The general structure of such procedure is;

```
FitZetaRequirement(MigrationTable,ζ)
  while((HotSpot=PickUpHotSpotPE(ζ))!=empty) do
     ST=ExtractSubTable(MigrationTable,HotSpot);
     while(E(ST)!=empty) do ModifyHeat(E.RequiredHeat);
     SimulateEffect(ST);
```

The introduced parameters α and ζ give the capability to cover a wide range of balancing requirements in terms of heat distribution and balancing speed. The

requirements space includes the most restrictive ones, e.g. α=1, and, ζ=0. These basic parameters and the basic algorithm fulfill our objective of this paper. In the next section we report our simulation results.

5 Simulation Results

In this section, we describe our experiments to study the performance of online data reorganization using the full-window algorithm. We evaluate the system performance where the metric used is the impact on the response time of queries and the system migration workload. Table 2 shows the major parameters and their used values. We first create an initial Fat-Btree with the tuple key values generated using a uniform distribution (space-balanced). Then we generate range queries using Zipf-distribution skew defined by the skew factor (τ) of the Zipf-distribution. Thus, there are more range queries are issued at one PE than the other PEs, depending on the skew factor τ. A query range is selected to be equal that of an index branch so that the workload is based on the assumed migration unit (an index branch and its corresponding data pages). The heat skew initiates the migration of branches between the PEs, depending on the given ζ requirement. We model each of the PEs as a resource and the queries as entities. We assume heat balancing is done in centralized scheme and it is initiated every 100*N queries, where N is the PEs number.

Table 2. The major parameters and their values

Parameter	Default Value	Variation
System Parameters: Number of PEs in the cluster Network bandwidth Time to read or write a page	 16 120 Mbits/s 8 ms	 32, 64.
Database Parameters Number of records Index node size Data page	2.1 millions (2MB) 4KB, key=4 Bytes. 4KB	
Query Parameters Zipf distribution of decay factor (τ) Hot spot location Mean arrival rate Mean service rate	 0.3 at PE0 20 500 ms	 0.1 $\rightarrow$ 0.9 PE0 $\rightarrow$ PE15 10 for skew variation experiment
Requirement Parameters. Speed parameter (α) Heat distribution parameter (ζ)	 1/4 10 %	 0, 1/64, 1/32, 1/16,1/8, 1. 0, 5, 15, 20, 30%.

It has been observed that balancing using the DCA leads to unstable balancing cases especially under skew of τ > 0.1 which in turn limits our consideration to access pattern skew and balancing speed as well. Thus we exclude the DCA results from our discussion relying on the fact of the full-window algorithm always gives the exact solution without any unstable cases. In the first set of experiments, we study the effect of the full-window parameters (α, ζ) on the system performance. Figure 2 shows the

full-window capability to fit a wide range of requirements including the most restrictive one (α=1.0, ζ=0.0). Figure 2.a traces the hotspot's response time, which indicates also the system capability to adapt itself to access patterns under various requirements on the balancing (adaptation) speed. Note that α=0 represents the hotspot response without heat balancing (space-balanced), while α=1 represents that response with one-step reorganization. As shown this response can be controlled to a desired level by tuning the parameter α. We express the migration workload at a PE as the total time during which the encountered PE has been involved in migration issues as source or destination. Figure 2.b shows the migration workload at each of the PEs. The bell-like curves are obtained as a result of distributing heat by migrating index branches (and the corresponding data pages) from the hot PEs (e.g. PE0, PE1) to other cold PEs (e.g. PE10, PE11) through some PEs (e.g. PE3, PE4), in a ripple mechanism. As shown this ripple mechanism is mainly dependent on the ζ requirement, where ζ=0 represents distributing the given heat as evenly as possible (the highest migration workload). As requirements relax this restrictive heat distribution, the ripple-migration effect can be reduced accordingly.

To demonstrate the disadvantages of the one-step reorganization (α=1.0), we first trace the average response time at each of the PEs during reorganization and we record the PE responses that almost dominate the system response during reorganization. As shown in Figure 3 (left), PEs other than the hotspot like PE2 and PE4 have slow responses because at these PEs there are high migration workloads and high arrival rates of users' queries. Such effect under the considered skew (τ=0.3) can not be avoided with the given requirement of the fastest balancing. Users of PE2, PE4, and PE6 during reorganization are the victims of the fastest balancing. However, if requirements allow incremental balancing under high skew environments, then by lowering α, e.g., α=0.25, such effect can be avoided as shown in Figure 3.(left). It shows the system response is dominated by the main hotspot PE, as a result of partitioning the whole work into 4 steps. The experiment indicates also the advantages of the incremental balancing under high skewed environments.

With the assumed range partitioning strategy, it has been observed that the system migration workload (and response) is dependent on hot-spot locations in the system, where the RMPE and the LMPE do not have much freedom in their migration direction like the other PEs. Figure 4 affirms such dependency on hotspot locations, where we express the system migration-workload as the summation of the migration workload at each of the PEs. It shows also that the ratio of the maximum migration workload to its minimum is about 2.5. This high ratio indicates the main problem of the given range-partition configuration which is mainly selected to simplify the implementation of the first-level index, and consequently, the search operations. However it gives us the motivation to consider another configuration in a future work so that migration decisions are mainly correlated to access patterns skew rather than to their favorite locations in a system. It shows also that we select the hotspot at PE0 (the worst case of the migration workload) to evaluate the proposed technique.

To demonstrate the scalability of the proposed strategy, we study the system migration-workload under different environments of skewed access patterns, and, number of PEs. Figure 5 shows that as the access pattern skew increases the system's migration workload increases in a nearly linear relationship. It demonstrates also the

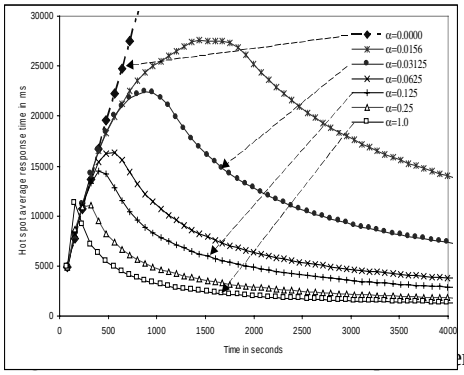
superiority of the full-window algorithm in dealing with access patterns over a wide range of skew. We repeated the experiment for different numbers of PEs for clusters of 32 and 64 PEs. It shows that as the number of PEs increases, the migration workload increases which in turn emphasis the need for heat balancing.

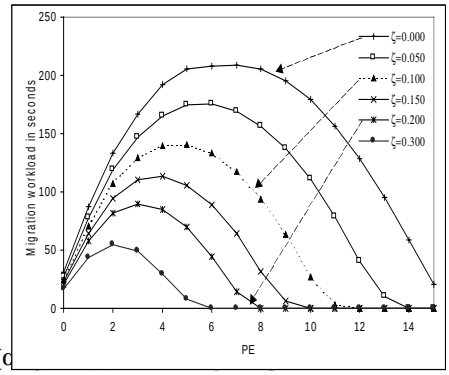
6 Conclusion

In this paper, we have developed a heat-balancing technique for Btree indexed database over shared-nothing parallel systems. It demonstrates that finding the exact solution for heat balancing avoids unstable cases while balancing, and, long convergence time. This gain gives a system the capability to adapt itself to access patterns as fast as required. Through its parameters, the technique can cover a wide range of balancing requirements in terms of heat distribution and balancing speed over a wide range of access pattern skew. It can be used for data placement with the goals of optimal system performance and it is useful for dealing with both advanced DBMS such as office document management or WWW servers, and relational database systems. In these application data declustering can be exploited by an appropriate mapping of documents/records into index keys. Apart from complexity, the simulation results are a first step toward gaining quantitative insight into the performance of our technique. Developing techniques that automate data placement in shared-nothing systems is a crucially important problem. We believe that the proposed technique is a promising approach toward solving this problem.

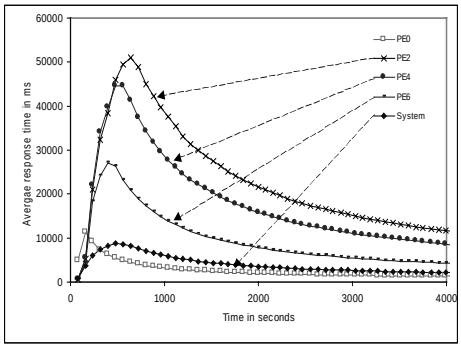
References

1. Achyutuni, K. J., Omiecinski, E., Navathe, S. B.: Two techniques for On-line Index Modification in Shared Nothing Parallel Databases. Procs ACM SIGMOD (1996)
2. Boral, H., et al: Prototyping Bubba, a Highly Parallel Databse System, IEEE Trans. On Knowledge and Data Eng., Vol. 2, No. 1, March (1990)
3. Copeland, G., Alexander, W., Boughter, E., Keller, T.: Data Placement in Bubba. Proc. of ACM SIGMOD Conference, pages 99-108, (1988)
4. DeWitt, D.J. and Gray, J.: The Future of High Performance Database Systems. Communication of ACM, 35(6), 85-98, (1992)
5. Lee, M. L.,Kitsuregawa, M., Ooi, B.C., Tan, K, Mondal, A.: Towards Self-Tuning Data Placement in Parallel Database Systems, Proc.. ACM SIGMOD pages 225-236 (2000).
6. Ozsu M., Valduriez., P.: Principles of Distributed Database Systems, Prentice Hall, (1991)
7. Salzberg, B., A. Dimock. Principles of transaction-based on-line reorganization. Procs. of the 18th Inter. Conf. on VLDB, pages 511-520, (1992)
8. Scheuermann, P., Weikum, G., Zabback, P., Adaptive Load Balancing in Disk Arrays. Proceedings of the 4th Inter. Conf. FODO, (1993)
9. Seeger B. and Larson P. Multi-Disk B-trees. ACM SIGMOD Conf.1991, 436-445
10. Tamura, T., Oguchi, M., Kitsuregawa, M.: Parallel Database Processing on a 100 Node PC Cluster: Case for Decision Support Query Processing and Data Mining. Proc. Of SC97: High Performance Networking and Computing, (1997)
11. Yokota, H., Kanemasa, Y., Miyaazaki, J.. Fat-Btree: An Update-Conscious Directory Structure. Procs. of IEEE the 15th IEEE Conf. on Data Engineering, pp. 448-457, (1999)
12. Zou C., Salzberg, B.: .On-Line Reorganization of Sparsely-Populated B+ Trees. Procs. ACM, pages 115-124, (1996)
13. Valduriez, P.,: Parallel Database Systems: Open Problems and New Issues, Distributed and Parallel Databases 1, No. 2, 137-165, Kluwer Academic Publishers, Boston, MA (1993).





ers (**a**) the hot spot PE (α effect) **(b)** the PE migration workload (ζ effect).



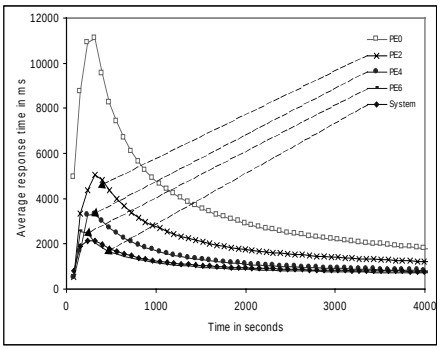


Fig. 3. The effect of the one-step (left) and M-steps (right with α=0.25) reorganizations on the PEs that dominate the system response.

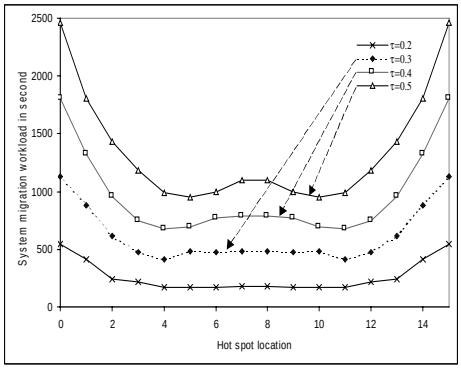


Fig. 4. The effect of the hotspot location on the system migration workload.

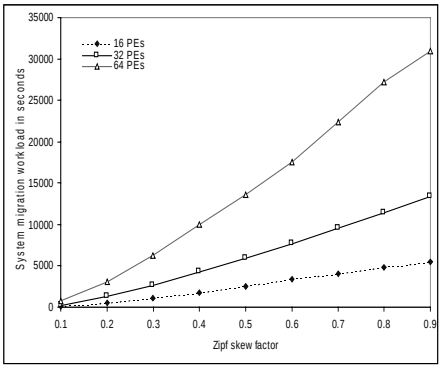


Fig. 5. The scalability of the proposed technique.