RING: A Strategy for Minimizing the Cost of Online Data Placement Reorganization for Btree Indexed Database over Shared-nothing Parallel Machines

Hisham Feelifl Masaru Kitsuregawa
Institute of Industrial Science, The University of Tokyo
7-22-1, Roppongi, Minato-ku, Tokyo 106-8558, Japan
{hisham, kitsure}@tkl.iis.u-tokyo.ac.jp
Tel: +81-3-3402-6231,Fax: +81-3-3423-2834

Abstract

In this paper, we propose a new strategy called RING that utilizes a ring configuration of the range partition strategy to achieve efficient online data placement reorganization in shared-nothing environments. In the event of reorganization, it has been observed that the range partition strategy in its well-known configuration, the linear configuration, can lead to two main drawbacks: high reorganization costs and performance dependency on hotspot locations in a system. Thus we modify the configuration into a ring seeking to minimize the effect of these drawbacks. We also introduce a new standalone heat-balancing algorithm to support the ring configuration, where its migration decisions are deduced by taking into account reorganization costs while balancing a system. RING is based on simple but effective approaches that incur only little overhead so that performance is almost optimized for free. Our simulation results indicate that under a wide range of requirements, performance can be considerably improved by modifying the underlying structure to support the ring configuration.

1. Introduction

High data volume in various fields like the Internet, Web and data warehouse has dramatically increased the need for efficient and flexible mechanisms to provide high performance in parallel database systems. Added to its scalability and cost, shared-nothing architecture is one of the attractive approaches for high-performance database systems [8]. However unlike other architectures, e.g. shared-disk, it suffers from the load (heat) balancing problems, where load balancing is difficult to achieve because it relies on the effectiveness of database partitioning for the query workloads [8]. Thus for efficient processing, data are typically declustered and indexed across the system processing elements (PEs), however, access patterns are inherently skew that can lead to performance bottleneck as some PEs become hotspot while many other PEs are cold (idle). Therefore, data reorganization among the PEs is essential and should be done online for optimum system performance. While data are being reorganized, the corresponding indices have to be modified as well; thus online data reorganization should also satisfactorily deal with the index modification [1,2,3,6]. We follow this direction and we base our strategy on minimizing costs of modifying indices.

We ideally seek an online reorganization strategy that not only does good heat balancing, but also minimizes the reorganization costs in the event of reorganization. Here, reorganization cost is defined as the amount of data (and the related objects, e.g. indices) that must be reorganized. Minimization of reorganization costs is an important goal especially when the reorganization has to be performed online, since the amount of data to be reorganized generally determines the duration of the reorganization, which in turn determines the time that concurrent operations experience degraded performance. A very effective way of reducing the duration of reorganization is to minimize the amount of data to be reorganized [1].

We assume that data are initially range partitioned across the PEs so that a parallel access method can be easily implemented as well as a heat-balancing facility can be employed in a system. However, it imposes restrictive rules at the edge PEs (the rightmost PE and the leftmost PE), where the rightmost PE can only migrate data with its left neighbour and the leftmost PE can only migrate data with its right neighbour. Such restrictions can lead to high reorganization costs that degrade performance for concurrent operations. And it can lead to performance dependency on hotspot locations in a system [2]. By relaxing such restrictions as supporting migration between the edge PEs, it increases migration freedom in a system and it creates a good chance to minimize reorganization costs. Because migration can be wrapped around at the edge PEs we refer to this configuration of migration under

0-7695-0996-7/01 $10.00 © 2001 IEEE

the range partition strategy by the ring configuration. Allowing the ring configuration in a system and preserving the partitioning strategy in the same time leads to modify the underlying structure, which in turn imposes additional goal to minimize such modification cost so that the ring would be more feasible for online reorganization.
In this paper, we propose a new strategy called RING that utilizes the ring configuration to achieve more efficient online data placement reorganization in shared-nothing environments. The strategy basically extends the existing ones, e.g. [2,3], in several points. First it modifies the underlying structure to support the ring configuration with almost negligible cost, so that if the ring will lead to better performance, the performance is almost improved for free. Second it introduces a new heat-balancing algorithm, called 2-bag, that can distribute a given heat among the PEs, as evenly as possible and as fast as possible, if so is required. The main mechanism of the algorithm is motivated to minimize reorganization costs under a given performance requirement. The strategy is further supported by parameters that can be tuned to cover a wide range of performance requirements in terms of distributing accesses among the PEs, and, partitioning reorganization jobs which indicates the speed for a system to adapt itself to an access pattern. Third as a result of minimizing reorganization costs and increasing migration freedom in a system, RING can remove the dependency of reorganization costs on hotspot locations in a system. So that performance during reorganization is mainly correlated to an access pattern's skew rather than to its skew and its favourite locations in a system. Through simulation studies, the conduct results prove that performance can be considerably improved by modifying the underlying structure to support the ring configuration
The rest of the paper is as follows. Section 2 discusses the related work. Section 3 describes the underlying index. Section 4 presents the ring configuration in terms of its costs and benefits for online reorganization. Section 5 introduces the 2-bag algorithm. Section 6 reports our experimental study. Section 7 considers our remarks for future work, and finally, we conclude the paper.

2. Related Work

Recently, researchers have noted the advantage of online reorganization that can provide high performance database systems. In [5], they present online efficient methods for the dynamic redistribution of data, however, they do not consider index modification while reorganization. In [4], 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 of [4,7] are proposed for centralized DBMS and require the use of locks, where using locks during reorganization can degrade performance significantly [1]. In [1], they present two elegant alternatives for performing the necessary index modifications, called OAT page movement and BULK page movement. However, the underlying structure is B+-tree that can lead to a considerable cost while modifying indices [2,3,6].
By modifying structure and supporting bulk migration, in [6] they use the Fat-Btree structure that can minimize index-modification costs. However, their 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 in a system [3]. The techniques of [2,3,6] are based on the linear configuration that can lead to high migration workloads and performance dependency on hotspot locations. In contrast, we propose the ring configuration by which migration workloads are minimized and performance can be improved.

3. The Underlying Index

Data are initially range partitioned across the PEs so that the access method can associatively access data for strict match queries, range queries and cluster data with similar values together. Such partition allows reductions in redundant I/O operations, and introduces inter-operation parallelism as well as intra-operation parallelism. One solution to associative access is to have a global index mechanism. Conceptually the global index is a two-tier index; the first-tier index directs the search to the PE wherein the data are stored, and the second-tier index directs the search to the data pages wherein the records are stored. The non-overlapping data partitioning gives also non-overlapping indices so that the first-tier index can be easily implemented as a partitioning vector, with entries number equals the PEs number. To ensure that there is no central PE, the first-tier index is further replicated in all the PEs [3].
Using a B-tree based index enables more efficient processing of range queries than a hashed index. Thus the second-tier index is a collection of aBtrees [3], one at each PE; each aBtree independently indexes its data. An aBtree is a variant of Btree, where its root node can be a *fat* node, i.e., for a Btree of order d, and maximum of $2d$ entries, the root node can contain more than $2d$ entries. This property of aBtrees gives the ability to design a tree with a predefined height while inserting keys in the tree.
We design the tree height to be the same across the PEs 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 instead of updating the whole trees (or some of their nodes) by inserting/deleting the keys of the migrated data one at time, e.g. the OAT technique [1]. This minimizes the required I/O cost to modify the structures, which in turn speeds up migration issues in a system. Such bulk migration does not change the tree

structure, but it causes an update in root nodes of Btrees at a source and a destination PE, which requires the first-tier index copies to be updated. This is can be done in a lazy manner by piggybacking update messages onto messages used for other purposes [3].

4. The Ring Configuration

Since data is range partitioned across the PEs, we can only move data from one PE to its neighbors PEs that hold the preceding or succeeding ranges. The edge PEs violate this rule, where the rightmost PE that can only migrate data with its left neighbour and the leftmost PE that can only migrate data with its right neighbour. We always refer to this data partition with its possible migrations as the *linear configuration.*

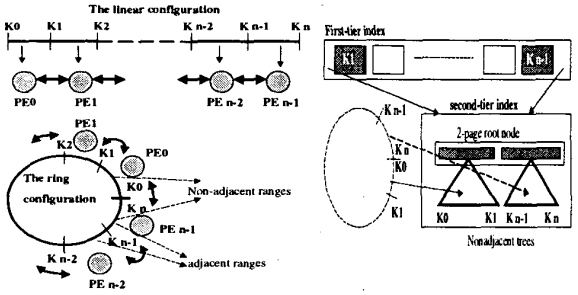

Figure 1: (a) Configurations (b) Index structure at a PE holding 2-key ranges (2 trees).

In the event of migration, it has been observed that the PEs can be logically configured into a ring rather than a linear set, and meanwhile, it is possible to preserve the assumed partition strategy. So that migration can be wrapped around at the edge PEs and data can be migrated without restrictions at these PEs. It establishes a fair rule at each of the PEs, each has exactly two neighbors, and it gives a good chance to minimize migration workloads and improve performance. However, the edge PEs, which are ring neighbors, hold non-adjacent trees that may disorganize the structure during migration. Allowing these PEs to hold two trees instead of only one can protect the structure at these PEs from any disturbance that may occur during migration. Permitting only non-adjacent trees at these PEs is not sufficient since as migration is taking place, those non-adjacent trees can be occurred at any other PE. We consequently support this modification of holding "two trees" at each of the PEs. The location of the non-adjacent trees is dependent on migration decisions that have been done so far. But, if there are non-adjacent trees at some PE, then the structure at any other PE should be abstracted by only one tree. Therefore, this modification, in its worst case, will lead to N+1 trees in a system instead of N trees as in the linear configuration, where N is the PEs number. The corresponding cost is equivalent to that of adding one tree to a system. Since the underlying index is based on the aBtree structure, which basically supports a root node of multiple pages, *"fat root"*, thus the implementation of holding two trees at a PE can be easily achieved without additional overheads, see Fig. 1. In Figure 1 the encountered root is a 2-page root each page corresponds to one tree. The structure does not change too much but this modification actually exploits its support for data reorganization. We always refer to this modification with its possible migrations as the *ring configuration.*

To demonstrate the effect of holding two trees at a PE, 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 PE0 is hotspot, then we have the freedom to migrate heat (data) with some key range, say 10-25, to PE1 or to migrate heat of other key range, say 1-15, to PE3. If for some reason we select the key range 1-15 for migration to PE3, then PE3 will hold two key ranges, one adjacent range to PE2, 76- 100, and the other is a non adjacent range, 1-15. Migration of key range 1-15 to PE3 should be reflected on the first-tier copies. Inserting a new entry into the first-tier copies can reflect such change so that the future search for that range will be directed to PE3. The cost of inserting a new entry into the first tier copies is equivalent to that of updating them, which is normally done after acknowledging every migration in a system. Thus the effect of having two trees at a PE has zero cost on updating the first-tier copies. Meanwhile, if after some interval of time PE3 becomes hotspot and PE0 becomes cold, then PE3 can migrate data of its key ranges to PE0. It is possible to reunion the whole key range (1-15) with that of PE0, or some part of it or all of it plus other key range that originally belongs to PE3, say 90-100. The range itself is dependent on the heat to be migrated from PE3 to PE0, but the example demonstrates the dynamic processes that can be occurred in the ring configuration during migration. The location of the nonadjacent trees is dynamically changed, and the number of entries of the first-tier index (at any copy) is dynamically alternated between N and N+1. The maximum of this alternation will never exceed N+1 since there is always two nonadjacent ranges as configuring a linear set of key ranges into a ring as shown in Figure 1.

Modifying the structure implies also modifying all procedures that deal with search operations and migration issues. It has been found that the upgrade code for that is simple and straightforward. Figure 2 gives an example for an upgrade code to extracts a required heat from a source PE in the ring configuration using that of the linear one.

Although its effect may be imagined as adding a new PE to a system but modifying the structure to support the ring configuration incurs less cost and it is almost negligible.

To show the gain we achieve from this modification, assume first that it is required to have a smooth heat distribution among linearly configured PEs. So that migration can be cascaded from the hotspot PE to the

coldest PE which can be several PEs away (in ripple ways). This ripple migration can lead to high volumes of data movement which in turn degrades performance as many PEs are involved in migration issues. For example, assume a system of 8 PEs with; PE0 is the hotspot and PE7 is the coldest PE. To satisfy the given smooth requirement, PE0 migrates data to PE1, which in turn migrate data to PE2, this expensive process could be repeated until some hot data being reached at PE7 as shown in Fig. 3. However, with ring configured PEs it would be easy to migrate directly hot data from PE0 to PE7, and thus saving a lot of migration jobs at the mid-PEs, which certainly will be reflected on the performance. This will be more beneficial in systems of large size of PEs working under highly skew environments, where migration workloads are proportionally related to the amount of data (heat) to be migrated, and, the number of the PEs that should be involved in migration issues.

```
Linear_GetHeatFromTree(PEs, TreeNo, Heat)
  return all index branches that correspond to Heat
int FindAdjacentTree(PEs, DestRange)
  // for every tree at PEs find the minimal adjacency with the
  // DestRange and retrun the corresponding tree no.
Ring_GetHeatFromPE(PEs, DestRange,Heat)
  HeatType H=0;int TreeNo;
  TreeNo=FindAdjacentTree(PEs,DestRange);
  while (H<Heat) {
    H+=Linear_GetHeatFromTree(PEs,TreeNo,Heat);
    TreeNo=1-TreeNo;} // get the other tree, if there is.
  return All index branches to be migrated.
```

Figure. 2: Upgrade code, an example.

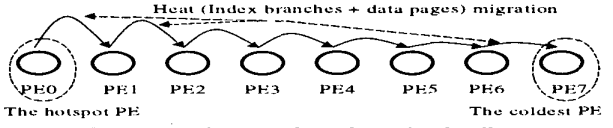

Figure 3: Ripple migrations in the linear configuration, an example

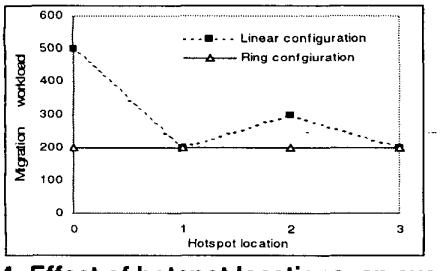

Figure 4: Effect of hotspot locations, an example.

It has been observed that because of its inflexibility at the edge PEs, the linear configuration can lead to migration workloads that are dependent on hotspot locations in a system. If the hotspot occurs at the edge PEs, it would be more costly than that if it occurs at some other locations, e.g. at the middle. This dependency dominates the system performance during reorganization. Ideally, performance during reorganization should be correlated to skew of access patterns rather than to their skew as well as to their favourite locations in a system. We generally cannot predict where will hotspots occur? But we can support a system by an inexpensive mechanism to remove such unfavourable effect. To demonstrate this point, assume a system of 4 PEs and a heat distribution (100,200,300,400) that can be assigned to the PEs with the following combinations; [(400,300,200,100), (100,400,300,200), (200,100,400,300), (300,200,100,400)]. The first distribution represents the hotspot at PE0, while the second represents the hotspot at PE1, and so on. Assume further, it is required to balance the system so that each of the PEs has 250 (average heat). Approximating the migration workload by *migration workload ≈ heat to be migrated* can give some insight to the problem. Fig. 4 shows the resulting migration workload against the variation of the hotspot location in the given distributions. It shows that there are hotspot locations that give the minimal workload, e.g. hotspot at PE1 and at PE3. It also gives some information about the percentage of the maximal to the minimal workload, which is 250 %. This high percentage enhances our objectives to remove or reduce such effect that mainly occurs because of the migration restrictions at the edge PEs in the linear configuration.

The ring configuration relaxes such restrictive migration by supporting migration between the edge PEs. It can be imagined that under each of the given distributions we can track the change in the hotspot location so that the given distribution can be mapped (arranged) into the linear one that gives the minimal workload.

This mapping can be viewed as virtually cutting the ring at some proper point so that it converts the given ring into the linear set that gives the minimal workload. For example, consider the distribution of (PE0=400, PE1=300, PE2=200, PE3=100) with the ring configuration. By preventing migration (or virtually cutting the link) between PE0 and PE1, it converts the ring into the linear one of (PE1=300, PE3=200, PE3=100 PE0=400). Its migration behavior is equivalent to that of (300,200,100,400) in the linear configuration, which gives the minimal workload as shown in Figure 4. By applying this cutting technique on each of the given distributions, the figure shows the ring configuration can remove the workload dependency on the hotspot location in the given distributions, and thus it can save a lot of migration workloads that may be done if the configuration were linear. Apart of its complexity (as exhaustively search for the proper cut), the cutting technique generally does not give the minimal workload because it is possible to obtain such minimal without need to prevent migration between two PEs in a system. But, we use it for demonstration and to give some insight to the balancing problem in the ring configuration, which is covered in the next section.

5. Heat Balancing: The Two-bag Algorithm

In this section, we present a new standalone heat-balancing algorithm to extract migration decisions that minimize reorganization costs for a ring-configured system. First we present the problem then we discuss the details of the proposed solution.

5.1 The Problem and The Solution Strategy

We are given N ring-configured objects, e.g. PEs or disks, their heat distribution, and a performance requirement is represented by parameters ξ and α. The parameter ξ represents a requirement for heat distribution, so that the heat at each of the PEs is no more than a *threshold heat*, where the *threshold heat* = $(1+\xi)$ x *average heat*. The parameter α, $0 \leq \alpha \leq 1$, represents a requirement for the required speed (steps) to adapt a system to an access pattern, i.e. how many steps $(=1/\alpha)$ are required to balance a system [3]. The problem is to find migration decisions that are required to satisfy the requirement (ξ, α) on the given ring with the minimal migration workload. Clearly, the problem belongs to the optimization problems and it can be solved using a greedy mechanism. A solution can be found if we succeed first to answer the following simple questions:

(1) At which object we should start balancing?
(2) If we succeed to select some objects as a starting point, then, how can we progress this selection to cover more objects, so that the system is heat-balanced by the minimal migration workload.

The initial objective for any heat balancing should intuitively focus first on the hotspot object, where the performance degradation comes from. Since it is required to include a threshold requirement, then it would be better to ensure that the hotspot object is being covered by any possible solution. We therefore select the hotspot object as a starting point. Then there will be a local requirement to migrate its excess heat (*its heat – threshold heat*) to one/both of its ring neighbors. Hence the second question comes, which neighbor we should select, the clockwise or the anti-clockwise neighbor or both of them? We select neighbors that lead to minimize migration workloads. Accordingly, we use migration workloads as an objective function that to be minimized while balancing a system. Since the migration workload is proportionally related to the heat to be migrated, we therefore evaluate it by; *migration workload ≈ heat to be migrated*, so that it can be simply evaluated from the given distribution. Given this simple strategy, we must translate it into code. To simplify matters, in the next subsections we first consider the solution for (ξ, 1) requirements, then we will generalize it for (ξ, α) requirements.

5.2 Definitions and Mechanisms

I- Marked-arc: is the arc that connects all the selected objects (PEs) so far. The Initial marked-arc = {the hotspot PE}. As selection is taking place, the arc is dynamically expanded, by including more PEs, in the directions that minimize migration workloads. During the course of the algorithm, the PEs may be thought of as divided into three categories as follows (see Fig. 6). (a) *Marked PEs*: PEs that belongs to the marked arc constructed so far. (b) *Fringe PEs*: not in the marked arc, but adjacent to some PEs in the marked arc. (c) *Unseen PEs*: All others. If a PE belongs to fringe or unseen PEs, we refer it as *unmarked*

II- Heat Requirement (*HR*): it represents the amount of the excess/missing heat that required for balance a current marked-arc. Since a marked-arc has two ends, then there will be generally two heat requirements that have to be satisfied. Its two ends (*End1* and *End2*) and its heat requirements (*HR1* and *HR2*) can represent a marked-arc by a structure of components (*End1*, *End2*, *HR1*, *HR2*). Each *HR* has the corresponding cost to satisfy it. If a marked-arc has one (or both) of its end has (have) zero heat requirement(s), it means the given arc is already balanced at such end (ends) and thus the corresponding cost(s) will set to zero.

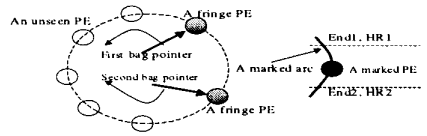

Figure 5: Notations of the 2-bag Mechanism

III – 2-bag Mechanism: A Heat-collection Mechanism

While constructing a marked-arc, we have to satisfy its heat requirements (*HR1* and *HR2*) and evaluate their cost. The procedure to satisfy one HR with minimal workloads can be abstractly formulated as collecting this HR from the nearest unmarked PEs that minimize workloads, which is equivalent to attaching some unmarked PEs to the encountered marked-arc. Since a ring gives the chance to simultaneously collect heat from both ends of a marked-arc, we introduce a mechanism, called 2-bag, that utilizes this property of a ring to satisfy a HR. The collection is done through two bags, one for each end of a marked-arc, each bag has the following structure:

```
typedef      struct
{ PeType     Pointer; // a pointer to the current unmarked PE.
  DirType    Direction; // clockwise or anti-clockwise
  HeatType   Collected; // collected heat so far
  HeatType   Excess; // the accumulated excess/missing heat.
  CostType   Workload; // migration workload evaluation.
  DecsType   *Decisions; // migration decisions
  FlagType   Activity; // 1 active, 0 reset.
} HeatBagType;
```

The "*Pointer*" component points to the current unmarked PE during heat collection, while the "*Direction*" gives its advance direction (clockwise or anti-clockwise). The component "*Collected*" indicates the collected heat by a bag to satisfy a *HR*. During traversing the unmarked PEs, the accumulated excess-heat at the current PE is

calculated by, *Excess* +=(Heat(PE)- threshold heat). The function of this component is as follows, if we accept the collected heat of a bag, we can simply generate a new heat requirement for the corresponding end of a marked-arc by, *HR*= - *Excess* of that bag. The "*Workload*" component evaluates the migration workload while collecting heat using the mentioned workload function, it also indicates the less expensive (lighter) bag. The "*Decisions*" component traces all the steps to collect the heat denoted by "*Collected*". The "*Activity*" indicates that if the corresponding bag is active or not.

Assume a marked-arc, a heat requirement *HR*, and two bags, one for each of its ends. One bag collects heat from the fringe PE of *End1* towards *End2* of the marked-arc, while the other collects from the fringe PE of *End2* towards *End1*, see Figure 5. Note that the bags' pointers are synchronized in their advance to discover the nearest unmarked PEs that minimize workloads. As heat is being collected, we can accept one of two bags or both, depending on the sign of the "*Collected*" component at each bag and that of the current *HR* by the following acceptance rule:

If sign (*This.Collect*) == sign (*HR*)
  If sign (*Other.Collect*) == sign (*HR*)
    If $|HR| >= |This.Collected + Other.Collected|$ **accept** both
    Else **accept** the bag that has the minimal workload.
  Else **accept** *This*.
Else **reset** *This*.

This rule ensures that the current marked-arc is expanded as selecting its nearest unmarked PEs that can balance it with minimal workloads. Once a bag's collection has been accepted, the current *HR* is updated accordingly and the corresponding migration decisions referenced by "*Decisions*" are stored as well. The collection procedure will terminate once the current *HR* becomes zero or there is no benefit to collect more heat.

While heat is being collected and as a result of the opposite movements of the bags' pointers, it is possible that the pointers collide at some PE, which can lead to invalid collection. However, we solve this problem by providing a locking mechanism, and, an activity-resetting mechanism. The locking mechanism ensures that any amount of excess/missing heat at an unmarked PE is being considered by only one bag and thus there is no multiple consideration for any amount of heat. The activity-resetting mechanism gives the possibility to minimize workloads as collecting heat by only one bag instead of two. The mechanism detects whenever bags' pointers collide, then it checks for a bag that have to be reset, by applying the above acceptance rule that also gives the bag to be reset, if there is. If so it resets the proper bag to its last accepted state and unlock the way for the accepted bag to continue its traverse/collection. This also gives the possibility to minimize workloads without essence to synchronize the pointers' advances. Both mechanisms ensure that heat is collected with minimal workloads and without contradictions that may be occurred as a result of two opposite moving pointers.

5-3 The Algorithm

The algorithm starts by picking up the hotspot PE from the given ring. If one or both its neighbors are hotspot too, then they virtually combine into one hotspot PE. This hotspot PE has only one heat requirement, *HR0= Heat (the hotspot PE) – threshold heat,* that can be satisfied through the 2-bag mechanism. The 2-bag mechanism generates a new marked-arc with two heat requirements (*HR1, HR2*). Each requirement can be satisfied using the 2-bag mechanism again, thus the algorithm constructs two marked-arcs, one corresponds for each requirement. To further minimize migration workloads, it accepts the marked-arc that has the minimal workloads. If a marked-arc has only one *HR* = 0, then it only considers the other *HR*. As accepting a marked-arc it generates new heat requirements by which the process can be repeated again until there is no more requirements. The algorithm terminates whenever the set of the unmarked PEs is empty, or, all of the unmarked PEs are not hotspot.

While constructing a marked-arc, it is possible to obtain one that has zero *HR* at its both ends and the algorithm termination is not satisfied. It means the encountered marked-arc is heat-balanced, we therefore store it and construct a new one from the current unmarked PEs by picking up their local hotspot PE and repeating the whole procedure until the algorithm termination is reached. Therefore, the algorithm output is generally a set of marked-arcs, each marked-arc has the required decisions to balance it with minimal workloads.

If all decisions generated so far are issued then they balance the system according to the parameter ξ and in a one step (α=1). This one-step reorganization implies that it does the whole work in a short time, which may be relevant for some applications, e.g. WWW servers, or irrelevant (or harmful or expensive) for the others. Representing a migration decision by *Source*, *Destination*, and *RequiredHeat* components, and normalizing every "*RequiredHeat*" component by "α* *RequiredHeat*", we can additionally introduce the speed effect while reorganizing a system [2]. So that the overall reorganization is partitioned into $1/\alpha$ parts (steps), each part can be separately issued to the system at some appropriate intervals or even in periodic ways. This Partitioning gives incremental reorganizations, where their steady state conditions are identical to that of one-step reorganization. Both parameters ξ and α give the ability to cover a range of performance requirements in terms of distributing jobs among the PEs and controlling the system speed to adapt itself to access patterns. Figure 6 shows the general structure of the 2-bag algorithm with its two main parameters ξ and α.

The algorithm could be generally used to balance N ring-configured objects with minimal costs regardless of the assumed migration unit. In the next section, we adapt it for the ring configuration with an index branch as a migration unit over shared-nothing parallel machines.

Procedure FitSpeedRequirement (α)
For *E* in migration decisions **do** *E.RequiredHeat* *= α.
Procedure NewMarkedArc(ξ)
Pick up the local hotspot PE and initiate a marked-arc;
HR0 = Heat (local hotspot PE) – Threshold heat (ξ)
Algorithm: TwoBags(α, ξ)
HR0=**NewMarkedArc**(ξ)
while "*it does not terminate*" do
Apply **2-BagMechanism** (*HR0*) and check termination
For *HRi* in {*HR1, HR2*} **do 2-BagMechanism** (*HRi*)
Accept the marked-arc that gives the minimal workloads.
If (*marked-arc has zero requirements*) **NewMarkedArc**(ξ)
FitSpeedRequirement(α)

Figure 6: The general structure of the algorithm.

6. Simulation Result

In this section, we describe our experiments to study and validate the performance of our strategy for online data placement reorganization. We evaluate the system performance in the linear and the ring configurations, where the metric used is the impact on the system migration workload and response time during reorganization. Table 1 shows the major parameters and their used values.

We first create an initial aBtree with the tuple key values generated using a uniform distribution (space balance). Then we generate range queries using Zipf distribution, the queries are generated with skew defined by the decay factor (τ) of Zipf distribution. Thus, there are more range queries are issued at one PE than the other PEs depending on τ. This heat skew initiates the migration of branches among the PEs, depending on the given requirement. We select one-step reorganization, α=1, as default requirements with a relatively high skew of τ=0.3 to show in addition the worst cases (and the effects) of fast reorganizations. We model each of the PEs as a resource and the queries as entities. We assume that heat balancing is done in centralized scheme and it is checked/initiated every 100 x N queries, where N is the PEs number. If there is a need for data reorganization we extract the required migration decisions using the algorithm of [2] in the linear configuration while the 2-bag algorithm in the ring one. We express the system migration workload as the accumulation of the individual migration workload at each of the PEs. We express the PE migration workload as the accumulation of time intervals in which the given PE is involved as a source or as a destination.

In the first set of experiments, we study the configuration effect on migration workloads. To visualize migrations among the PEs in both configurations, we record the PE migration workload in each configuration as shown in Figure 7. It shows the ripple migration effect on both configurations that gives bell-like curves with local maximum at some PEs, e.g.. PE4 in the linear, and, PE4 and PE15 in the ring one. It shows also the added capability at PE0, the hotspot, to migrate its excess heat to other PEs through PE15. Therefore, workloads at the mid-PEs are greatly reduced, compared to that of the linear, and the system migration workload is reduced accordingly. As shown because of the given ξ, some PEs have zero workloads, e.g. from PE12 to PE15 in the linear and from PE7 and PE12 in the ring one, where their resulting heats are less than the threshold heat.

Table 1: Major parameters and their used values.

Parameter	Default values / variation
System Parameters	
Number of PEs in the cluster	16 /32.
Network bandwidth	120 Mbits/s
Time to read/write a data page	8 ms.
Database Parameters	
Number of records	1 million,
Index node size	.4KB, key = 4 byte.
Data page size	4KB.
Query Parameters	
Zipf distribution decay factor	0.3/ 0.1 → 0.9.
Mean arrival rate	20.0 / 8.0 per second
Mean service time	500 ms.
Others	
The hot-spot location	0/ 0 → 15 for 16 PEs.
Speed parameter (α)	1/ {0, 1/6, 1/8}
Heat distribution parameter (ζ)	0.1/ 0 → 0.5

With the ring one, there are workload increments at some PEs, e.g. at PE0, PE13, PE14 and, PE15. The increments at PE13, PE14 and PE15 do not have the considerable effect on the performance because all of these PEs are almost idle (cold) before reorganization, see their workloads in the linear configuration. But, the workload increment at PE0, the hotspot, can not be ignored and it certainly degrades its performance during reorganization. Such increment can not be avoided because otherwise the migration workload will not be minimized. It occurs because the excess heat at PE1 is migrated to other PEs, e.g. PE13 and PE14, through PE0 and PE15 so that the given ξ requirement is satisfied across the PEs.

Studying the migration behavior of the hotspot PE in the ring configuration shows that it is involved in the ripple migrations of (PE1→**PE0**→PE15 → ..) by which it is involved as a source and as destination in the same time (in the same sequence of decisions [2]). A similar situation occurs in the linear but at another PE, PE1, which is involved in the ripple migrations of (PE0→**PE1**→PE2→ ..). To some extent, these similar situations give nearly similar migration workloads at PE1 in the linear, and, at PE0 in the ring, see Figure 7. Therefore, the workload of PE0 in the ring configuration

could be occurred in the system regardless of the used configuration. Furthermore, it is also possible to have a heat distribution that leads to migration decisions in which the hotspot of the given distribution is involved as a source and as a destination in the same time, regardless of the configuration. For example assume 4 PEs are linearly configured with the following heat distribution (PE0=50, PE1=400, PE2=300, PE4=250), so that each PE has 250 unit of heat after balancing. The corresponding migration decisions gives the hotspot PE, PE1, as a source for the decision PE1 →PE0 of heat =200, and in the same time as a destination, for the decision PE2 →PE1 of heat = 50. Thus the ring configuration does not introduce a negative effect, as it increases the hotspot workload. Regardless of the used configuration in section 7 we present one solution to improve performance of a hotspot PE that has conditions of migration workload and arrival rate similar to that of PE0 in the ring configuration.

Figure 8 compares the sensitivity of the system migration-workload to ξ parameter in both configurations. It shows the migration workloads saving in the ring configuration to fit ξ requirements, e.g. it saves about 350% for $\xi \rightarrow 0$ requirements. As ξ increases the saving decreases until the configuration (reorganization itself) does not have the great effect on the system performance, where the performance degrades as ξ increases.

To evaluate the influence of the hotspot location on migration workloads, the query set is designed so that the hotspot location can be changed from *0* to *N-1*, so that a query set (*i*), *i= 0,1, .. ,N-1*, represents the hotspot at the PE*i*. Figure 9 shows the system migration workload for each of the query sets in both configurations. As expected, migration workloads in the linear configuration are dependent on hotspot locations. It shows that the approach of minimizing migration workloads, which is employed in the ring configuration and the 2-bag algorithm, is efficient in removing such dependency. Consequently migration workloads are mainly correlated to access patterns' skew rather than to their skew as well as to their favorite locations in a system. The figure also shows that the percentage of the maximal workload in the linear to that of the ring is about 320%, and, the average saving in migration workloads is about 163 %.

To study the average saving in migration workloads under different environments of access patterns and hotspot locations, we repeat the pervious experiment under a range of the skew factor (τ). However, to cover a range of τ the experiment is conducted with; a low arrival rate of 8 queries/sec, and, migration is initiated to fit the ζ requirement regardless if there is a steep degradation in the system response or not. Figure 10 compares the system's average migration workload in both configurations. It shows the ring capability to save a lot of migration workloads, and this capability is proportionally related to τ. To demonstrate scalability in the same time, we repeated this experiment over a cluster of 32PEs. As the number of PEs employed is increased, the reorganization costs is increased proportionally, and more saving in such costs can be achieved by the ring configuration as shown in figure. The experiment proves that the ring is more reasonable in its reorganization under a wide range of working environments.

To reflect the ring effect on the system performance, we observe the system average response without migration, and, with migration in both configurations. Figure 11 traces the system average response during reorganization. It basically affirms the effectiveness of data reorganization, in general, and heat balancing, in specific, in correcting the performance degradation that can be occurred as a result of access skew. And, it shows the effectiveness of the ring configuration in improving the system performance during reorganization. Since both configurations do the same heat balancing under the given requirements, however, the amount of data to be reorganized is different which in turn determines the time that concurrent operations experience degraded performance. As shown, it gives better performance in terms of the fastest response for users' queries and the fastest speed for the system to adapt itself to the given access pattern. This better performance can be viewed as achieving a new upper-limit of the system adaptation speed, which is not easily achievable with linearly configured systems because of their high reorganization costs. This performance can be also achieved regardless of the hotspot locations of the access pattern.

7. Discussion and Future Work

In this section, we discuss/extend the experimental results of section 6 as well as we cover our future goals.

Under fast reorganization requirements, the potential possibility of high migration workload at a hotspot PE in both configurations (e.g. consider Figure 7 and PE0 in the ring) leads to consider some local optimization techniques at this PE to hide (absorb) its high workload from its performance. One of these techniques is based on scheduling migration decisions in which this hotspot PE is involved. For example, issuing first the decisions in which the hotspot PE is a source (e.g. PE0→PE15 in the ring) and postponing the ones in which this PE is a destination (e.g. PE1→PE0 in the ring). Then, once its response becomes within some accepted level, the postponed decisions can be considered later. Applying this simple scheduling technique on PE0 in the ring, Figure 12 shows that the hotspot response in both configurations can be brought to similar levels of performance and therefore its workload increment has been hidden from its performance. Figure 11 also reflects this effect on the system response time, which is not equivalent to that effect at the hotspot PE because the system response is

already improved by the ring effect (as minimizing migration workloads in a global respect). Although the above technique is simple and sufficient but more research is required to generalize it in cases of multiple hotspot PEs having similar conditions to that of PE0 in the ring. In a future work we intend to consider that as well as to exploit more local optimization techniques to optimize performance.

Studying of figures 7, 11, and 12 shows that the effectiveness of minimizing reorganziation costs in improving not only the average system performance but also the performance at each of the PEs during reorganization which is quit fair for all of the system's users during reorganization. This is not easily achievable with the linear configuration because of high migration workloads at many PEs. Fast reorganizations with the linear configuration improve the hotspot performance as fast as required but in the same time they degrade the performance at many PEs as well. Users during steady state conditions (after reorganization) are the luckiest ones but most of users during reorganization form a dark picture of victims for fast reorganizations, e.g. users of PE0 ~ PE8 during reorganization. This picture is distributed across many PEs in the system, which in turn limits the possibility of using some local optimization techniques to improve performance during reorganization. However with the ring only PE0's users during reorganization represent the real victims in correcting the system degradation as fast as possible. Since there is only one PE, or generally a few of PEs, at which performance degrades, thus performance can be more improved by local optimization techniques as shown in Figure 11.

Tuning the parameter α in its global respect can control a system reaction to an access pattern at a desired level. Figure 13 shows this ability, where $\alpha=0$ represents performance without reorganization, and, $0 < \alpha \leq 1$ represents performance with incremental reorganization of steps = $1/\alpha$. We plan automatically tuning this parameter according to dynamics of access patterns in both local and global respects. So that performance can be optimized over a wide range of access dynamics.

Due to its high cost, it may be even more desirable to operate the system in skew conditions than to perform the required data reorganization. For example, in Bubba [8] the system estimates the data reorganization cost and decides whether to tolerate the skew condition or to perform the required data reorganization. To demonstrate such tradeoff in terms of the ξ parameter, we measure the system average response in the ring configuration under various values of ξ. Figure 14 shows the obtained results, which affirms that as ξ increases, reorganization cost decreases and the performance degrades accordingly. Under the given condition of skew and arrival rate, performance/reorganization tradeoffs of $\xi \geq 0.5$, are inefficient in correcting the system degradation because the resulting reorganizations are not sufficient for the given condition. The system should be more reorganized by decreasing the value of ξ up to the most inexpensive level, which means there is a critical value of ξ at which the amount of data to be reorganized is quite necessary and sufficient to keep performance within some accepted levels. More investigation is required to explore a general methodology by which such critical ξ can be determined under a given condition of skew and arrival rate.

8. Conclusion

Data placement reorganization in parallel database systems is a critical factor in determining performance. Given the dynamic nature of access patterns, the optimal placement of relations will change over time and this will necessitate some reorganization to maintain the system performance at acceptable levels. In this paper, we focused on one particular aspect –namely, how does the choice of a migration configuration affect the subsequent reorganization costs in parallel databases? This is an important practical issue since the amount of data to be reorganized directly affects the duration of the online reorganization, which in turn determines the duration over which concurrent operations experience degraded performance due to contention with the reorganizer. We have developed an online data reorganization strategy over shared-nothing machines. It demonstrates that modifying the underlying structure to support the proper configuration is efficient in optimizing the system performance. Apart from complexity, simulation is chosen at first to gain some quantitative insight into the performance of our strategy. Developing strategies that automate data reorganization and minimize its subsequent costs is a crucially important problem. We believe that this paper is a promising approach to solve this problem.

References

1. Achyutuni, K.J., Omiecinski, E., and Navathe, S.B. Two Techniques for On-line Index Modification in Shared Nothing Parallel Databases. Procs ACM SIGMOD 1996, 125-136.

2. Feelifl, H., Kitsuregawa, M., and Ooi, B.C. A Fast Convergence Technique for Online Heat-balancing of Btree Indexed Database over Shared-Nothing Parallel System. LNCS, DEXA00, pp. 846-858, 2000.

3. Lee, M.L., Kitsuregawa, M., Ooi, B.C., Tan, K, and Mondal, A. Towards Self-Tuning Data Placement in Parallel Database Systems, Procs. ACM SIGMOD, pp. 225-236, 2000.

4. Salzberg, B. and Dimock, A.. Principles of Transaction-based On-line reorganization. VLDB, pp. 511-520,1992.

5. Scheuermann, P. Weikum, G., and Zabback, P. Adaptive Load Balancing in Disk Arrays. FODO, 1993

6.Yokota, H., Kanemasa, Y. and Miyaazaki, J. Fat-Btree: An Update-Conscious Directory Structure. IEEE Data Engineering,1999, 448-457.

7. Zou, C. and Salzberg, B. "On-Line Reorganization of Sparsely-Populated B+ Trees". Procs. ACM, SIGMOD 1996, 115-124.

8. Valduriez, P. Parallel Database Systems: Open Problems and New Issues. Distributed and Parallel Databases 1(2), April 1993, 137-165, Kluwer Academic Publishers, Boston, MA.

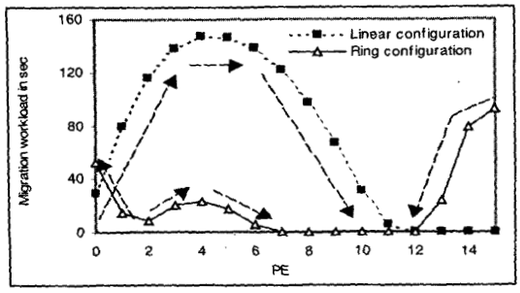

Figure 7: Configuration effect on the PEs migration workload

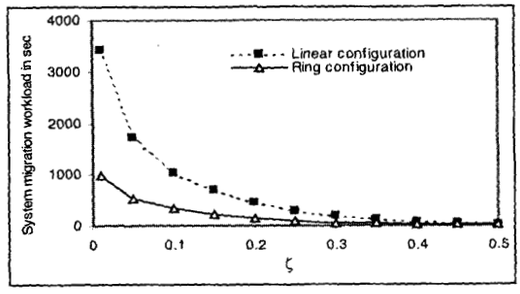

Figure 8: Effect of ζ parameter on the system migration-workload in each configuration.

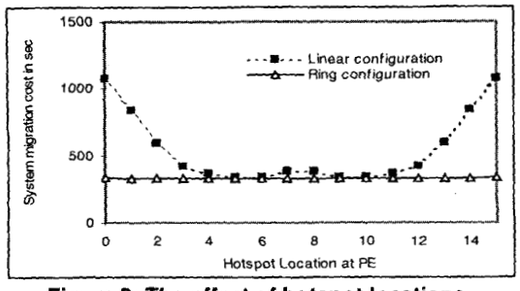

Figure 9: The effect of hotspot locations.

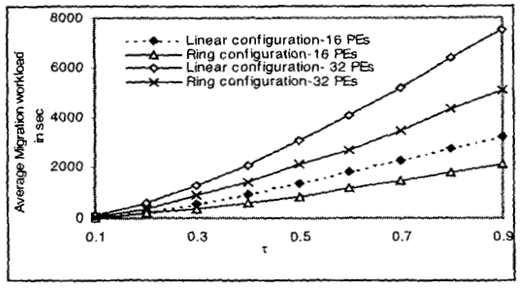

Figure 10: Average migration workloads under a range of τ for clusters of 16 and 32 PEs.

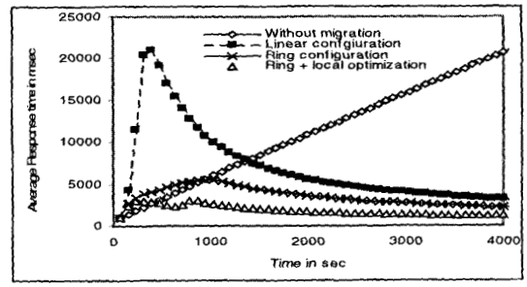

Figure 11: Effect of heat balancing.

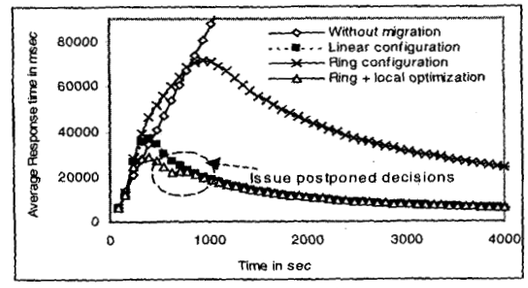

Figure 12: The hotspot PE's response.

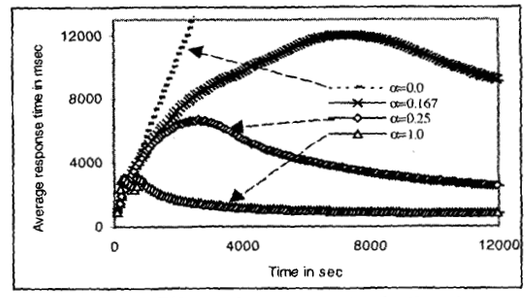

Figure 13: Effect of α parameter in the ring configuration.

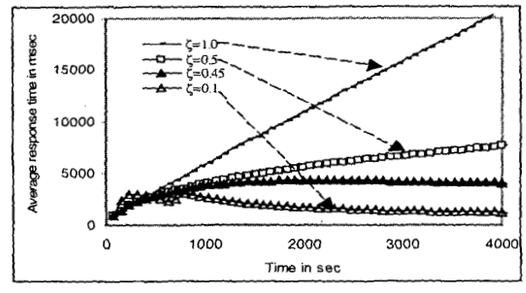

Figure 14: Effect of ζ parameter on the system average response in the ring configuration.