DEWS00
Paper Number: **66**

Heat Balancing for Btree Indexed Database over Ring Configured Shared Nothing Machines

Hisham Feelifl and Masaru Kitsuregawa
Institute of Industrial Science, The University of Tokyo
Institute of Industrial Science, University of Tokyo
7-22-1, Roppongi, Minato-ku, Tokyo 106, Japan.
E-mail: {hisham, kitsure} @tkl.iis.u-tokyo.ac.jp

Abstract

In shared nothing machines, data are typically declustered and indexed across the system processing elements (PEs) to achieve efficient query and transaction execution. Since the access pattern is inherently dynamic, therefore, there is no one placement of relations that is optimal for the lifetime of the database system. Whenever data is moved across PEs, the indexes need to be modified too. Consequently reorganization based on the access history (heat) of the data or its corresponding index is essential and should be online, in addition, it should satisfactorily deal with the index modification, as data is moving across the PEs. In this paper, we study the heat balancing strategies that have the capability of minimizing the index modification. Meanwhile, it has been observed that the system performance during reorganization is greatly affect by the cost of the balancing decisions (migration cost) and the location of the hot spot in the system. In this paper, we propose a new migration configuration, namely the ring configuration, it has the capability to reduce the migration cost and to absorb the dependency on hot spot locations. It has been shown there is a considerable reduction in the system migration cost, which mainly improves the system performance during the reorganization process. We also propose a new online heat balancing algorithm, namely two bags of heat, that utilizes the ring configuration with additional advantages of distributing the given heat as evenly as possible across the system PEs and the capability to minimize the balancing convergent time. We evaluate the performance of the proposed strategy in comparison with the linear configuration; the conducted simulation shows its efficiency in correcting the system-performance degradation.

1. Introduction

As demonstrated by the existing machines such as Bubba [B90], Gamma [D90] and Kitsuregawa's 100 Node PC Cluster [TOK97], shared nothing machines have three main virtues: cost, extensibility, and availability. By implementing a distributed database design that favors the smooth incremental growth of the system by the addition of inexpensive processing elements (PEs), extensibility can be better (in the thousands of PEs). With careful partitioning of the data on multiple disks, linear speedup and linear scaleup could be achieved for simple workloads. By replicating data on multiple PEs, high availability can be also achieved. Shared nothing architectures suffer from the load balancing problems. Load (heat) balancing is difficult to achieve because it relies on the effectiveness of database partitioning for the query workloads. Unlike other architectures, load balancing is decided based on data location and not the actual load of the system [V93]. The basic motivation to investigate and realize heat-balancing facilities comes from simple experience that several applications in shared nothing machines usually do not exploit the system very good. Further more, the addition of new PEs in the system presumably requires reorganizing the database to deal with the heat balancing issues.

In shared nothing machines, each PE has exclusive access to its memory modules and disk unit(s). The PEs communicate with each other by sending messages via the communication network, the only shared resource. To achieve efficient query and transaction execution, data are typically declustered and indexed across the system PEs, where the execution of a transaction or a query is distributed over the network. However, the access pattern is inherently dynamic, which in turn can lead to performance degradation as some PEs become "hot spot" (frequently accessed), therefore, reorganization for heat balancing is essential. Heat balancing is particularly challenging for evolving workloads, where the hot and cold data (infrequently accessed) change over time. Data reorganization can only counteract such situations, and such reorganizations should be performed online without requiring the system to be quiescent [WZS91 & SWZ93]. Additionally, as the data moving from hot spot PEs to cold PEs, the corresponding indexes have to be modified too. Therefore, data reorganization should also satisfactorily deal with the index modification [AON96].

In [FK99-1 & FK99-2], they propose online heat balancing strategies for parallel indexed database, in which the data migration process is based on distributing the given heat as evenly as possible across the system PEs. However, it has been observed that the migration cost produced by their migration decisions may be considered high. The minimization of the migration cost is an important factor that improves the system response during the reorganization process. In addition, it is has been observed that their migration cost is dependent on the hot spot location in the system. Such dependency on the hot spot location in the system may lead to long responses during the reorganization process, if the hot spot occur in a costly position. In this paper, we extend their work by increasing the

alternatives during the heat balancing process and tuning their strategies performance in term of migration cost and insensitivity to the hot spot location in the system. By introducing a new migration configuration, namely the ring configuration, has the capability to reduce the migration cost and to absorb the dependency on the hot spot location in the system. The organization of the paper is as follows. In the next section, we briefly discuss the related work. Section 3 establishes the system search structure, the considered migration strategy for index modification, and the system workload. Section 4 clarifies the considered migration configurations and Section 5 discusses the proposed heat balancing strategy for ring configured shared nothing machines. Sec. 6 deals with the experimental work and finally, we conclude the paper.

2. Related work

Data reorganization should take place only when the benefit outweighs the cost [CABK88]. Though there has been much work in the area of online reorganization in the recent years. In [WZS91 & SWZ93], the authors present an online method for the dynamic redistribution of data, which is based on reallocation of file, fragments. A limitation of their study is that they do not consider index modification. [SC91 & SC92] present a simulation based performance study of online index construction algorithms, they present ten algorithms which typically as follows; a reorganizer scans the data, copying out information for index entries, concurrently with updaters that modify the same data. The proposed algorithms differ in the data structures used for concurrent updates, their strategies for combining these updates with the newly created entries, and finally, in the degree of concurrency supported following the scan phase. Although these algorithms are limited to centralized DBMS, they may consider as the basis of recent work [e.g. AON96]. Perhaps [SD92] is the first paper that discusses a solution for online index reorganization. They outline the issues involved in changing of all references to a record when its primary identifier is changed due to a record move. The techniques in the [SD92, ZS96] are limited to centralized DBMS and require the use of locks, where using locks during reorganization can degrade performance significantly [AON96]. In [AON96] they examined the problem of online index reorganization. They present two alternatives for performing the necessary index modifications, called one-at-a-time OAT page movement and BULK page movement. While these alternatives are extremes on the spectrum of the granularity of data movement, they both depend on the conventional B+-tree algorithms in insertion and deletion, which increases the cost of, index modification.

To minimize the index modification cost, in [YKM99] they suggest the Fat-Btree as a powerful search structure that supports the data reorganization and speeds up the migration process. Basically, they have noted that whenever trees height at a source and a destination are at the same value, the amount of data to be migrated correspond to the entirety of one or more index branches at a source PE. So that, 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 without excess overhead in index modification. However, their objective is to balance the number of pages across the system PEs (space balancing) rather than balancing the search structure workload that produced by the system access pattern (heat balancing) across the system PEs. Where, access pattern skew may lead to performance bottleneck even though the system is already space balanced. In addition, they base their space-balancing algorithm on the disk-cooling algorithm [SWZ93], which has relatively long convergent time and instability cases [FK99-1]. By using the Fat-Btree and considering the access pattern skew, in [FK99-1] they propose an online heat balancing strategy to reorganize the data with minimal cost of modifying the indexes. The heat balancing is based on distributing the given heat as evenly as possible across the system PEs, which in general improves the system performance, with the advantage of minimizing the convergent time of the balancing process. However, the migration decisions that required for balancing the given heat are carried out through one step, which may be harmful for the system performance. In order to reduce the effect of the one step migration, in [FK99-2] they provide two algorithms. In the first algorithm, they drop some of the system PEs from the balancing process, which reduces the migration cost. However, the mechanism of dropping some PEs is completely depends on the given heat distribution rather than on the system requirements. While in the other algorithm the migration decisions are carried out in incremental way (many steps) instead of one step. This is has the advantage of reducing the migration cost, and thus better performance during the reorganization but it increases the convergent time of the balancing process, which may be an important requirement in many environments. In addition, it increases the cost required to update the first level of the global index as results of long migration decisions. In contrast, this paper extends their algorithms by providing a new migration configuration that can decrease the migration cost and preserve the main advantage of minimizing the balancing convergent time and minimizing the first level updates. We also base our strategy on the Fat-Btree as a powerful search structure has the capability of minimizing the index modification cost and on the principle of distributing the given heat as evenly as possible, which has a great effect on the system performance.

3. The System Global Index

We assume that data are initially range partitioned across all the system PEs so that the access method can associatively access data for strict match queries, range queries and cluster data with similar values together. Using a B-tree based index enables more efficient processing of range queries than a hashed index, where only the nodes containing data in the specified range are accessed. One solution to associative access is to have a global index mechanism replicated on each PE [OV91]. Conceptually, the global index is a two-level index with a major clustering on the PE range and a minor clustering on some attribute of the relation. The first level directs the search to the PE wherein the data is stored. The second level of the index is a collection of Fat-Btrees, one at each PE; each Fat-Btree independently indexes the data at its PE [YKM99].

A source PE (PE*r*) can be defined as the PE from which the data pages (through the corresponding index branches) have to be moved to other PEs. Similarly, a destination PE (PE*d*) can be defined as the PE at which the data pages (and index branches) have to be stored. If the height of the Fat-Btree at the migration source and destination are the same, then the amount of data to be migrated correspond to the entirety of one or more branches of the Fat-Btree at the source PE. So that, it would be easy to prune the entirety of the branches from the Fat-Btree at the source PE as well as attaching these branches into the Fat-Btree at the destination PE using bulk-migration technique without excess overhead. The attachment of branches at the destination tree and detachment these branches at the source tree are essentially pointer updates. The amount of data to be migrated is obtained from the index branches at the source PE. The branches to be migrated are obtained from the heat statistics of the system access pattern. The branch migration does not change the tree structure itself, but it redistributes the data pages distribution at both the PE*r* and PE*d*. In addition, it causes an update in the root node of the Fat-Btrees at both the PE*r* and PE*d*, 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. We base our reorganization strategy on the advantages of the Fat-Btree in speeding up the migration process and minimizing the index modification cost.

3.1. The system workload

The system workload is reflected by a metric, called *heat* [CAB88]. We define the heat of a range R = *{Rmin .. Rmax}* as the access frequency of R over some period of time. A range R as a logical or abstract quantity could be achieved by any physical quantity in the system such as a data page, an index branch, an index tree, and a PE. Maintaining heat statistics on a range varies from a maximal to a minimal cost case, depending upon the physical quantity **O** that holds the range R. The maximal cost case may appear, if we maintain heat statistics for every data page in the system, so that **O** = {data page}. This roughly requires maintaining statistics for every possible point in the whole range. The minimal cost case could be achieved if we maintain heat statistics for every tree (or PE) in the system, **O** = {PE}, which requires information proportional to the PEs number. Although the approach is simple and inexpensive, but it has the disadvantage of the inaccurate estimation in the workload. On the middle of the spectrum, there are mid-cost cases, e.g., maintaining heat statistics for every index branch in the system or even for every sub-tree at every root node in the system, O = {Sub-tree}. The main advantage of such approaches is; they provide some compromise solution in term of cost and accuracy that may be required to measure the system workload. In our simulation, we use one of such mid-cost approaches that suggested by [FK99-1], in which we maintain heat statistics information for every sub-tree at a root node of a PE. Then, in order to minimize the required information, they assume a uniform heat distribution in the deeper levels of every sub-tree. Thereby, it would be easy to estimate the heat of every index branch in the distributed search structure. However, we consider, in principle, the workload estimation (approach) is a "design parameter" that mainly depends on the applications and their requirements.

4. Migration Configurations

In this section, we discuss the migration configurations; the linear and the ring configurations. Where, both are based on the range partitioning strategy.

4.1 The Linear Configuration

Since data is range partitioned across the PEs, we can only move data from one PE to its neighboring PEs (left or right or both), which hold the preceding or succeeding ranges, see Fig. 1. In Fig. 1, a link is a virtual link between PEs indicating that there is a permitted migration that can be carried out between such PEs. Clearly, the above rule of the migration directions in the system has only two exceptions. The first exception deals with the "rightmost" PE, which has the capability of migrating data only with its left neighbor. While the second deals with the "leftmost" PE, which has the capability of migrating data only with its right neighbor. This non-overlapping data partitioning gives also non-overlapping indexes. With the advantage of simplifying the system global index, especially in the implementation of its first level. So that, the first level, which directs the search to the proper wherein the data is stored, can be basically implemented as a partitioning vector with entries number equals the PEs number in the system. We refer this type of data partition with its migration directions as the *linear configuration.*

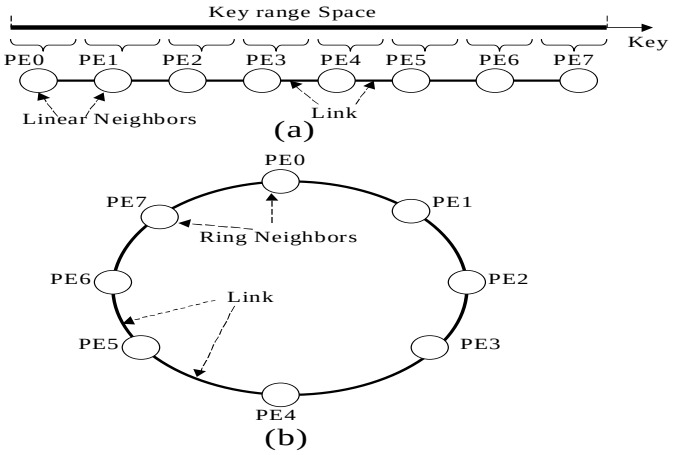


Fig.1 .0 Migration configurations of 8 PEs. (a) The linear configuration. (b) The ring configuration.

4.2. The Ring Configuration

It has been observed that the system PEs can be configured into a ring rather than a linear set, and in the same time, the range partitioning strategy with the 2-levels search structure can be preserved. So that, migration can be wrapped around by allowing the rightmost and the leftmost PEs to hold two key ranges (or two trees) instead of only one (as in the linear configuration). Since with range partitioning strategy, the rightmost and the left PEs are initially hold non-adjacent ranges, we support the ring configuration by permitting of non-adjacent trees at these PEs. However, when data reorganization

is taking place, it is possible that those non-adjacent trees occur at any PE in the system rather than the rightmost or the leftmost PE. Therefore, we support this property at any PE in the system. With the ring configuration, there will be in general non-adjacent trees that could occur at any PE in the system and its location is mainly dependent on the migration decisions that have been done so far. While there is non-adjacent trees at some PE, the search structure at every other PE in the system is only one tree. Noting that, we have more flexibility in the migration directions at any PE in the system; i.e. heat can be propagated through the system without any restriction at the rightmost or the leftmost PE that imposed by the range partition strategy. It also gives the chance to reduce the system migration cost and its effect on the system performance during reorganization.

Before considering the migration issues that may be achieved by the ring configuration, let us demonstrate its effect on the search structure. Assume we have 4 PEs with the following key ranges: PE0 is assigned 1-25, PE1 26-50, PE2 51-75, and PE3 76-100. If PE0 is the hot spot PE, then with the ring configuration, we have the freedom (flexibility) to migrate data (heat) with keys say, 10-25 to PE1 or to migrate data with keys data 1-15 to PE3. The selection between them is mainly dependent on the migration decisions that required for balancing the whole system, and their cost. Thus, if for some reason we select the range 1-15 and migrated it to PE3, then PE3 will has two key ranges, adjacent range (to PE2) 76- 100 and non adjacent range (ring migration effect) 1-15. In the same time, the migration of range 1-15 to PE3 has to be reflected at the first level of the global index, which can be reflected by insertion of a new entry in it, so that the search will be directed to PE3 for such range. The cost of insertion of a new entry in the first level is equivalent to that of its updating, which is normally done after every migration that carried out in the system. Therefore, it has a negligible cost on the system performance and it could be imagined it as adding a new PE to the system. Furthermore, if after some period of time PE3 becomes the hot spot and PE0 becomes a cold PE, then PE3 has the capability to migrate heat (and data) from its key ranges to PE0 depending on the amount of heat to be migrated. It is possible to return back the whole key range (1-15) that adjacent to the current key range of PE0 or some part of it or all of it plus some key range, say 90-100, that originally belongs to the initial key range of PE3. Of course, the situation is completely dependent on the amount of heat to be migrated from PE3 to PE0, but, the example demonstrates the dynamic process that could be happen at the first level of the global index while the migration is taking place. Clearly, the number of entries in the first level (at every PE) will dynamically alternate between N and N+1, where N is the number of PEs, (and it will never exceed the value of N+1). The alternation depends on the system history in terms of the access pattern and the corresponding migration decisions.

With the linear configuration, in order to achieve a smoother heat distribution among the system PEs, the migration can be carried out with ripple effect, i.e., by cascading the migration from the hot spot PE to the coldest PE which can be several PEs away. For example, assume a system of 8 PEs with the hot spot at PE0 and the coldest PE is PE7. The following scenario will happen; PE0 will migrated data and its corresponding index branches to PE1, which in turns migrates data to PE2, the procedure is repeated until the heat and the corresponding data are migrated to PE7, see Fig. 2.

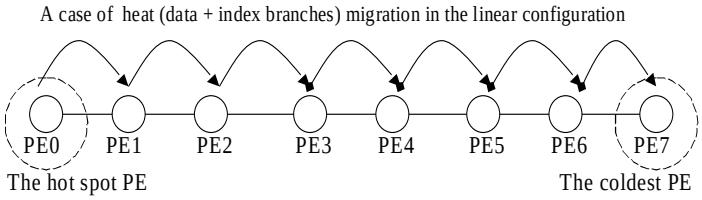


Fig. 2 A ripple-migration case in the linear configuration

```
FindAdjacentTree(SourcePE, DestinationRange)
  // for evert tree at the SourcePE check for the minimal adjacency.
   for(i=0,MinimumIndex=0;I<TreesNumber(SourcePE);i++)
   { Range=RootRange(SourcePE,i); // get a tree root range
     Adjacency=RangesAdjacency(Range,DestinationRange);
     If(i) { if(Adjacency<MinimumAdjacency)
           {MinimumAdjacency=Adjacent; TreeIndex=i;}}
     Else {MinimumAdjacency=Adjacency;TreeIndex=0;}}
     return TreeIndex; // return the tree number (index).
GetHeatFromTree( SourcePE, TreeIndex, RequiredHeat)
// Determine the tree branches from the tree of index = TreeIndex
// branches are extracted from the tree based on the required heat- see Sec. 3-1
//  return all branches that corresponds the required heat

GetHeatFromPE(SourcePE, DestinationRange, RquiredHeat)
HeatType AcquiredHeat=ZeroHeat; int Treeindex;
TreeIndex=FindAdjacentTree(SourcePE,DestinationRange);
 while (AcquiredHeat<RequiredHeat)
{AcquiredHeat+=GetHeatFromTree
            (SourcePE,TreeIndex,RequiredHeat);
 TreeIndex=1-TreeIndex; // get the other tree, if there is.
}
return All index branches to be migrated.
```

Fig. 3. A high level description of the heat extraction process from trees of a source PE.

Thus in the linear configuration, the ripple migration may be carried out on an expensive way as the heat (data) should moves (by data migration) for several PEs away, accordingly, this will increase the system response time during the reorganization process. While with the ring configuration it would be so easy to migrate directly the heat (data + index branches) from PE0 to PE7 because both of them are ring neighbors. Noting that as data are migrated from PE0 to PE7, there will be the possibility to introduce a non-adjacent range to the PE7's search structure. The given example shows that the ring configuration can provide a ripple migration in an inexpensive way. Especially, for systems of large size of PEs and highly skewed access patterns, where the cost of the ripple migration is proportionally to the amount of migrated heat (data) and the distance for which the heat should propagate. Fig. 3 outlines the simple procedure that deals with the heat extraction process from a source PE's trees that could result with the ring configuration.

It has been also observed that the system's migration cost, in the linear configuration, depends on the hot-spot locations in the system. If the hot spot occurs at the leftmost or the rightmost PE, it may more costly than if it were at some other location, say in the middle, as a result of the inflexibility imposed by the linear configuration. This dependency gives a different system behavior in terms of the migration cost and the average query response time for different hot-spot locations.

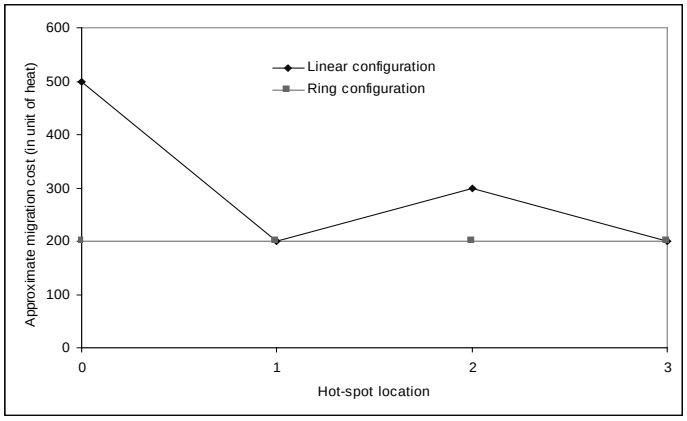


Fig. 4 Hot spot location and migration cost dependency.

Ideally, the hot-spot location should be opaque from the system performance, so that the system reaction to the access patterns is mainly correlated to their skew rather than their favourite locations. We generally can not predict where will the hot spot occur? But what we are seeking for providing a configuration that has the capability to remove such unwanted effect. In order to demonstrate this point, assume a system of 4 PEs and a heat distribution (400,300,200,100) that can be assigned to the PEs with the following combinations; [(400,300,200,100), (100,400,300,200), (100,200,400,300), (300,200,100,400)]. The first heat distribution represents the hot spot at PE0, while the second represents the hot spot at PE1, and so on. Assume further, it is required to balance the system so that every PE has 250 unit of heat (average heat) after the balancing process. We can apply, for example, the full-window algorithm [FK99-1], which is based on the linear configuration, with the following approximation *migration cost* ≈ *migrated Heat* (so that we can easily demonstrate the idea with such simple example). Fig. 4 shows the obtained migration cost against the hot spot location. It clarifies that there are some hot-spot locations that give minimal migration cost, e.g. hot spot at PE1 or at PE3. It also gives an information about the percentage of the maximal migration cost by its minimum, which equals about 250 %. This gives us the motivations to consider another configuration that capable to remove such dependency on the hot-spot locations.

To some extent, the system with the ring configuration has the capability to absorb such dependency. Since a ring configuration can be easily converted into a linear configuration by just imagining we have the ability to "cut" (ignore) only one neighborhood relationship between any ring neighbors. Therefore, the ring configuration can be converted to the linear configuration that gives the minimal migration cost. The resultant linear configuration can be dynamically changed as tracking the change in the hot-spot location, thus for every considered heat distribution in the given example, we obtain the same value of migration cost. With this virtual property of the ring configuration we can also, for example, exhaustively search for the "cut" that minimize the migration cost between the ring neighbors. This example shows, in addition, the "cutting" property that can be applied to any ring neighborhood with preserving the search structure organization.

From the above discussion, it is quit reasonable to support the configuration that has the capability to reduce the migration cost and in the same time has the capability to opaque the hot spot location from the system performance. In the following section, we consider a heat-balancing algorithm that capable to balance PEs configured in the ring configuration, and in addition has the capability to minimize the migration cost that may required to balance the given ring.

5. Online Heat Balancing

The heat balancing is an important factor that can determine the response times, speedups, and throughput of the system. The data reorganization should satisfactorily deal with this issue. Online heat balancing is done in four basic steps: monitoring PE workload, exchanging this information between PEs if it is necessary, calculating new distribution and making the work migrating decision, and the actual data migration. In this section, we consider the heat balancing strategies for Btree index over parallel shared nothing machines. In the next sub-section, we consider the "full-window" algorithm that we use it as a part of our proposed algorithm.

5.1. Heat Balancing of the Linear Configuration.

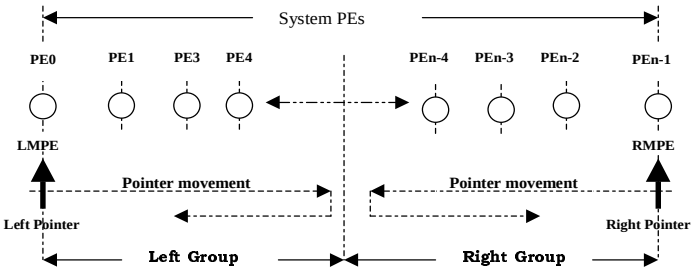


Figure 5. The "full window" algorithm notation,

In [FK99-1] & [FK99-2] they propose three algorithms based on the linear configuration, however, the three algorithms are cored with the following idea. They observe the exceptions in the migration directions that exist in the linear configuration. These exceptions exist at the "right most" PE (RMPE) and the "left most " PE (LMPE), where the RMPE can only migrate –as a source or a destination- data to its left neighbor, while the LMPE can only migrate with its right neighbor. Their algorithm starts by recording the possible migrations at both the RMPE and LMPE of the system with their neighbors. Then, by virtual dropping these PEs from the system, there will be new RMPE and LMPE, at which the process can be repeated again. This can be demonstrated by two pointers with their movements are shown in Fig. 5. The

process is repeated until the system is virtually vanished after storing the migration sequence -which is required to balance the given system- into a structure called "migration directory". By executing these decisions, the system is heat-balanced as evenly as possible. They refer to this algorithm as the *"full-window"* algorithm.

5.2 Heat Balancing of the Ring Configuration: Two-Bags Algorithm.

In this section we consider the ring configuration for a system of N PEs. First we formulate the problem, then we discuss the proposed solution.

Problem statement

Given a system of N PEs configured in a ring configuration R that described in Section 4, and the corresponding PEs heat distribution. The problem is to find the migration decisions that are required for heat-balancing the ring R with the minimal cost of migration decisions, so that, the system performance during the reorganization time is kept within accepted ratings. The problem belongs to the optimization problems that can be solved using a greedy algorithm.

The problem solution can be considered if we succeed to answer the following basic questions:

(1) Where should we start the balancing process; particularly, at which PE we should start the balancing process.
(2) If we succeed to select one PE or some PEs, then, how can we extend (progress) this selection process to include more PEs, so that the system is heat-balanced.
(3) If we succeed to extend the selection process, then, how can we minimize the corresponding migration cost.

To answer the first the question; the initial consideration for any heat balancing process should intuitively focus on the hot spot PE, where the system's main problem comes from. Besides, if it were required to include some threshold in the heat balancing process, then it would be better to include the hot spot PE under any threshold consideration. This could be achieved easily if we first select the hot spot PE as the starting point of the proposed algorithm. As we select the hot spot PE, there is a need to migrate its excess heat (= its heat – average heat) to one of its neighbors. Hence the next questions come, which neighbor we should select, the clockwise or the anti-clockwise neighbor, of course, the one with which the system's migration cost is minimal and the system is heat balancing. Accordingly, the answer to the last questions are; the selection progress is based on the minimization of the migration cost. In addition to these questions, the migration cost can be considered as the objective function while heat-balancing the system. Thus, we need to evaluate the migration cost during the balancing process, where the migration cost is proportionally related to the heat to be migrated which in turn proportionally related to the data pages and the corresponding index branches that have such heat. Therefore, we can approximate the objective function by; *migration cost* ≈ *migrated heat*, so that, the cost evaluation is achieved from the given heat distribution, thus with the minimal effort. For example, if the hot spot PE has the heat of 400 with the average heat of 250, then the amount of heat to be migrated from the hot spot is 150 (400-250), which gives also its migration cost by the above approximation = 150. The previous discussion guides our consideration to the problem solution.

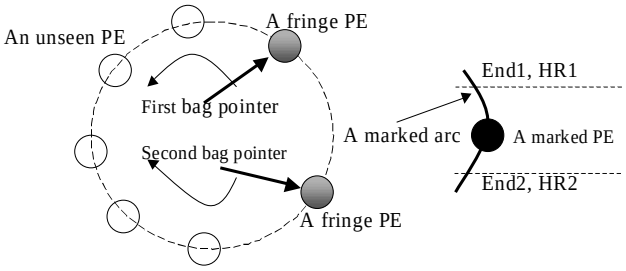


Fig.6 Two bag procedure

Algorithm basic: Definitions and basic procedures

1- *Marked Arc*: is the arc that connects all the selected PEs so far. The Initial "marked arc" = {the hot spot PE}. As the selection is taking place, the arc is dynamically expanded, by including more PEs, in the direction(s) that minimize the migration cost. During the course of the algorithm, the PEs may be thought of as divided into three categories as follows. (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. See Fig. 6.

2- *Heat Requirement (HR)*: it represents the amount of the excess/missing heat that required for balancing a "marked arc". Every "marked arc" has two ends; therefore, there are two heat requirements one for each end and each requirement has its corresponding cost to satisfy it. Noting that, it is possible to have a "marked arc" with one (or both) of its end has (have) zero heat requirement(s), which means the arc is already balanced at such end (ends), and in terms of requirements and cost, if there is not requirement then there is not cost.

We can also define a "marked arc" by its two ends *(End1* and *End2*) and the corresponding heat requirements (*HR1* and *HR2*), so that any marked arc can be described as "marked arc"= (*End1*, *End2*, *HR1*, *HR2*).

3- *Heat collection procedure* (Two bags)*:* during the construction of a "marked arc", we have to satisfy its heat requirements and evaluate their cost. The procedure can be abstractly formulated as collecting the required heat from the current "fringe" (and "unseen") PEs so that the encountered "marked arc" is heat-balanced. This is equivalent to attaching some of the "fringe" (and "unseen") PEs to the encountered "marked arc", and therefore, we satisfy its heat requirements. Since the configuration gives us the chance to simultaneously collect heat at both ends of a "marked arc", we propose a mechanism called two bags, where the heat is collected through a structure called "heat bag" with the following components:

```
typedef        struct
{ PEType       Pointer; // a pointer to the current PE.
  DirectionType Direction; // pointer-advance direction .
  HeatType     HeatRequirement; // current requirement
  HeatType     CollectedHeat; // collected heat so far
```

```
    HeatType    ExcessHeat; // the accumulated excess heat.
    CostType    Cost; // cost evaluation of the collected heat.
    FlagType    Activity; // 1 active, 0 inactive.
} BagType;
```

The "*Pointer*" component points to the current PE during the heat collection process, while "direction" gives its advance direction for the future collection (clockwise or anti-clockwise). The component "*HeatRequirement*" gives the current heat requirement from the current "fringe" and "unseen" PEs, while "*CollectedHeat*" indicates the collected heat by a bag. During traversing the "fringe" and "unseen" PEs, the accumulated excess-heat at the current PE is calculated by, *ExcessHeat* +=Heat(PE)- average heat. The function of this component is as follows, if we accept a bag's collected heat, we can simply generate a new heat requirement for a new "marked arc" as follows, *HR*= - *ExcessHeat* of such bag. The "*Cost*" component evalutes the migration cost corresponding to the collected heat so far. The "*Activity*" component indicates that if the corresponding bag is active or not.

Assume we have a heat requirement *HR*, in order to collect heat from the both ends of the "marked arc", we support two bags, one for each side. The heat requirement at each bag is initially assigned to *HR*. One bag's pointer moves from the "fringe" PE of *End1* towards *End2* of the "marked arc", while the other bag's pointer moves from the "fringe" PE of *End2* towards *End1* of the "marked arc", see Fig. 6. At any time the heat is collected and with the aid of the cost evaluation at each bag we can accept one of them or both, depending on the heat requirement and their cost. In the same time, once the collected heat of one bag is accepted, then the "*HeatRequirement*" at every bag is updated accordingly. While the heat collection is taking place and as the result of opposite movements of the bags' pointers, it is possible that the two bags' pointers collide at some PE, which may lead to invalid heat-collection. However, we solve such problem by providing; a locking mechanism and an activity-resetting mechanism. The activity-resetting mechanism checks and decides the following decisions:

(1) Reset the bag activity that has zero "CollectedHeat", which means this bag is not capable to collect heat with the given requirement and its pointer movement direction.

(2) Reset the bag activity that has the more highly cost, which means this bag is capable to collect heat but in expensive way and the other bag can collect such heat with less cost. In this case, the resetting means reset such bag to its last accepted heat-collection.

The mentioned mechanisms insure the heat collection is always carried out without any contradiction.

Algorithm

The algorithm starts by picking up the hot spot PE from the given ring R and then "branches out" from the ends of the arc constructed so far ("marked arc") by choosing new members (neighbors) at each iteration. The key step in the algorithm is the selection of PEs from the "fringe" and "Unseen" PEs according to the heat requirements at the "marked arc" ends and their cost evaluation. The initial state of the algorithm is as follows; we have only one heat requirement, HR0, which equals: *Heat (the hot spot PE) – average heat* and it can be satisfied through the two bags procedure, which in turn generate a new "marked arc" with new two requirements. The requirement satisfaction, which yields the generation of new "marked arc", can be repeated again until there is no more heat requirements. The general structure of the algorithm can be described as follows.

Pick up the hot spot PE and initiate a marked arc;
HR0 = Heat (the hot spot PE) – average heat
Apply two bags procedure to satisfy HR0
While "algorithm does not terminate" do
  For each of the two requirements
    Satisfy the current requirement by using the two-bags procedure and evaluate the corresponding cost.
  Accept the "marked arc" that gives the minimal cost.
  Generate new requirements.
  If the current "marked arc" has zero requirements
    Store it and generate a new "marked arc" with its requirements.
End.

The algorithm terminates if one of the following conditions is true

- The set of the "unseen" and "fringe" PEs is empty.
- The heat standard deviation of the Unseen PEs less or equal a predefined value, where, we consider the heat-standard deviation across the PEs is the balancing threshold. So that, the data migration can be initiated or generally controlled by this predefined value.

During the "marked arc" generation, it is possible to obtain a new "marked arc" that has zero requirements at both of its ends and the algorithm-termination conditions are not satisfied. This means the current "marked arc" is heat-balanced, therefore, we store the current "marked arc" and generate a new one from the set of the "fringe" and "unseen" PEs, then select the local hot-spot PE (from the "fringe" and "unseen" PEs) and repeat the whole algorithm until one of its termination conditions is reached. Thereby, the output of the mentioned algorithm is, in general, a set of "marked arcs", which in turn utilizes the "cutting" property of the ring configuration. It should be noted that the achieved "cutting" is mainly a heat-distribution dependent. Each obtained "marked arc" can be considered as a set of PEs that configured with a linear configuration rather than a ring configuration. Then, it would be easy to apply a linear-configuration algorithm, e.g. the full-window algorithm, on every obtained "marked arc" and produce the corresponding migration decisions that required for heat-balancing the given arc. The accumulation of the partial decisions forms the migration decision that required for balancing the given ring with the given threshold.

Procedure "produce migration decisions"
For every "marked arc" generated by the above algorithm
  Apply the full-window algorithm on it and produce the corresponding migration decisions.
  Store the current decisions into the migration directory.

It should be noted that, we select the full-window algorithm to acquire its advantage in minimizing the balance's convergent time. Besides, it may be useful to store the decisions into a

structure called "migration directory", so that, it would possible to apply some profitability analysis on the obtained migration decisions, e.g., we can drop some decisions without losing too much gain and saving additional cost. However, in order to clarify first, the effect of the ring configuration and the corresponding heat-balancing algorithm without any profitability analysis we drop such analysis from our simulation.

6. Simulation Result

In this section, we describe our experiments to study the performance of the online data reorganization with the mentioned configurations; the linear and ring configurations. We use the full-window algorithm as a heat-balancing strategy for the linear configuration, while we use, for the same purpose, the proposed algorithm, 2 bags, for the ring configuration. We evaluate the system performance in each configuration, where the metric used is the impact on the response time of queries (during the reorganization process) and the system migration cost. Table 1 shows the major system, database and query configuration parameters with their default values and variation settings.

Table 1: The major parameters and their used values.

Parameter	Default values / variation
System Parameters:	
Number of PEs in the cluster	16 /32/64
Index node size	4K page
Network bandwidth	120 Mbits/s
Time to read or write a data page	8 ms
Database Parameters:	
Number of records	1 million, 4 Byte as a key size.
Query Parameters:	
Zipf distribution decay factor	0.3; $0.1 \rightarrow 0.9$
Mean arrival rate	20.0/ 25 per second.
Mean service time	500 ms
The hot-spot location	0/ $0 \rightarrow 15$ for 16 PEs.
Strategies threshold = Heat standard deviation/average heat.	0.07/ $0.01 \rightarrow 0.2$

We first create an initial Fat-Btree with the tuple key values generated using a uniform distribution. Then we generate range queries using Zipf- distribution, the queries are generated with skew that defined by the skew factor (τ) of Zipf-distribution. Therefore, there are more range queries are issued at one PE than the other PEs, depending on the skew factor τ. The heat skew will initiate the migration of branches between the PEs, depending on the balancing strategy's threshold. We model each of the PEs as a resource and the queries as entities. We assume the heat balancing is done in centralized scheme and it is initiated after every 100*N queries, where N is the PEs number. In addition, we assume the data migration is done in some semi-incremental approach, in which the one-step migration granularity, at any time, is within two limits. The minimum granularity of such step (other than zero) is only one index branch, while the maximum is only one sub-tree at the root node. These limits are globally unified across the system PEs; thus, any amount of heat (to be migrated) is normalized such that the corresponding migration granularity is interpreted within the mentioned two limits. It is has been shown in [FK99-2], such semi-incremental has the effect on the system performance rather than for example coarse migration (one step migration).

In the first set of experiments, we study the effect of the ring configuration compared to the linear configuration on the migration issues. Noting that, we express the system's migration cost as the summation of the individual cost (time) that is required to carry out every migration issued by the system. Meanwhile, we express the migration workload at any PE as the summation of the individual time in which such PE is involved in the migration process as a source or a destination. In order to visualize first the flexibility of the ring configuration, we consider the following experiment. We record the migration workload at every PE in the system under the default values of Table 1, Figure 7 shows the obtained results in both configurations. The figure indicates the capability at the hot-spot PE (PE0) to migrate its excess heat to PE1 and PE15 (its ring neighbors) as a result of the ring configuration. In the same time, it shows the capability at the coldest PE (PE15) to help the hot spot PE (and other PEs, e.g. PE1) by distributing some of its (their) excess heat to the unseen neighbors of PE0 (and other, e.g. PE1), e.g. PE14 and PE13. Thus, the migration workload at the mid-PEs (e.g. PE7 and PE8) are greatly reduced as indicated which in turn reduces the overall migration workload. The arrows in the figure indicate the direction of the heat migration (propagation). In addition, it implies the effect of the ripple migration on the migration workload at every PE with both configurations as indicated by bell-like curve (between PE0 and PE15) with the linear configuration and those with the ring configuration. For example, the bell-like curve that between PE2 and PE10, and the curve (folded) that between PE1 and PE11.

In order to evaluate the system dependency on the hot-spot locations, we design the considered query set, that follows Zipf-distribution, such that the hot-spot location can be changed from 0 to N-1, where N is the number of PEs, a query set (i), i= 0.. N-1, represents the hot-spot at PEi. Figure 8 shows the system's average migration cost against the hot-spot location for both configurations. The figure affirms the existence of such dependency with the linear configuration, for example ratio migration cost when the hot spot at PE0 to that when the hot spot at PE14 is about 3, which means the system performance may be unfortunately degraded, if PE0 were the hot spot PE. As indicated also, this dependency can be removed by the ring configuration which has a nearly constant migration cost, so that the system reaction to the access pattern is mainly correlated to its skew, regardless of the hot-spot locations.

In order to study the system's migration cost under different environments of skewed access patterns and hot-spot locations, we first assume PE0 is the hot spot PE (the worst case for the linear configuration, see Fig. 8). Figure 9-a compares the system's migration cost associated with the considered configurations under a variable skew factor with PE0 the hot spot at PE. From this figure, we can observe that as the data skew increases, the system's migration cost increases in a nearly linear relationship. It mainly shows the effectiveness of the ring configuration in reducing the worst cases of the migration cost in the linear configuration. However, in order to

cover a wide range of hot-spot locations, we consider the following experiment based on the average measurements. For each considered value of the skew factor, we change the hot-spot location from 0 to N-1 and record the corresponding migration cost one for each hot-spot location. Then we take the average of these values to represent the cost under the considered skew factor. Figure 9-b compares the average migration cost associated with the considered configurations under variable skew factor. The figure demonstrates the superiority of the ring configuration in reducing the migration cost under a wide range of access pattern skew and the hot-spot location. We also observe that the average reduction in the migration cost is about 43%. Furthermore, the migration workload can be also controlled through the strategy threshold. Fig. 13-a shows the threshold sensitivity of the proposed strategy, it indicates that the heat balancing decisions can be tuned to cover a wide range of the system requirements. To demonstrate scalability, we repeated the described experiment for different numbers of PEs. Fig. 10 shows the scalability of the proposed strategy for clusters of 32 and 64 PEs with the mentioned average measurement. The main results affirms that for skewed access patterns as the number of PEs increases, the migration workload increases and thus heat balancing.

In order to evaluate the system performance with the considered configurations, we first consider the system "without" reorganization. Figure 11 gives the query's average response time against its arrival rate. To demonstrate the performance with reorganization, we select the arrival rates of 20/sec and 25/sec to carry out our experiments, where the system originally at these rates is dead (without reorganization), see Fig. 11. By heat balancing, the system can answer queries at such arrival rates and its responses, while reorganization is taking place, are given in Fig. 12. The measure is achieved by recording the query average response time from the time at which migration process has been initiated to the time at which the system is ready to operate in steady state. The result basically affirms the effectiveness of the heat balancing in correcting such degradation in the performance. It affirms in addition the effectiveness of using the ring configuration as a result of reducing the migration cost. Furthermore, Fig. 13-b shows the ability to tune heat-balancing process to meet the balancing requirements.

6. Conclusion

Heat balancing is necessary because the optimal data placement changes with time and usage of the parallel database system, and significant performance improvements can be obtained by heat balancing. In this paper, we have proposed a heat-balancing model for ring configured shared nothing machines, that has the capability to distribute the given heat as evenly as possible in minimum convergent time. In addition, it has the capability to minimize the system's migration workload, so that the performance during reorganization is kept within accepted ratings. The model includes the data migration with the minimal cost of index modification. The model has the advantages that can be reflected on the system performance, so that it can be employed in many practical applications. The Heat balancing has been studied before, but no work addresses the critical issue of distributing the given heat as evenly as possible across the PEs with minimal index modification and the capability to minimize the required migration decisions. This paper fills an essential void.

References

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

[**B90**] H. Boral et al., "Prototyping Bubba, a Highly Parallel Databse System", *IEEE Trans. On Knowledge and Data Eng., Vol. 2, No. 1, March 1990.*

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

[**D90**] D.J. DeWitt et al.,"The GAMMA Database Machine Project". *IEEE Trans on Knowledge and Data Eng., Vol. 2, No. 1, March 1990.*

[**FK99-1**] H. Feelifl and M. Kitsuregawa. "Online Heat Balancing for Parallel Indexed Database On Shared Nothing System", *Proceedings of 1999 database workshop, pp. 85-92, 1999.*

[**FK99-2**] H. Feelifl and M. Kitsuregawa. "The Simulation Evaluation of Heat Balancing Strategies for Btree Index over Parallel Shared Nothing Machines" Technical report of IEICE, DE99-88,1999-10, pp. 7..12.,

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

[**SD92**] B. Salzberg and A. Dimock. "Principles of transaction-based on-line reorganization". *Procs. of the 18th Inter. Conf. on VLDB, pages 511-520, 1992.*

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

[**TOK97**] T. Tamura, M. Oguchi, M. Kitsuregawa "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.*.

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

[YKM99] H. Yokota, Y. Kanemasa, J. Miyaazaki. "Fat-Btree: An Update-Conscious Directory Structure". *Procs. of IEEEthe 15th IEEE Conf. on Data Engineering, pp. 448-457,1999.*

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

[**V93**] P. Valduriez, "Parallel Database Systems: Open Problems and New Issues," Distributed and Parallel Databases 1, No. 2, 137-165 (April 1993), Kluwer Academic Publishers, Boston, MA.

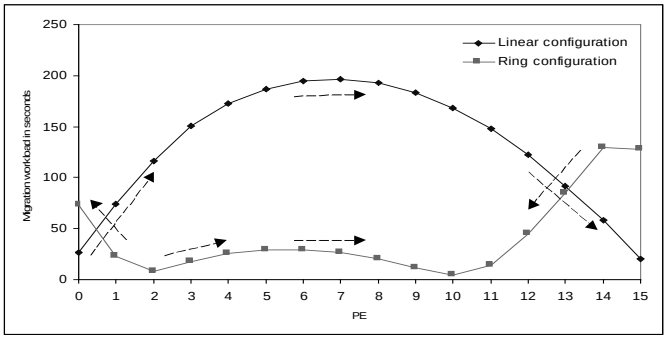


Figure 7: The ring configuration effect on the PEs' migration workload.

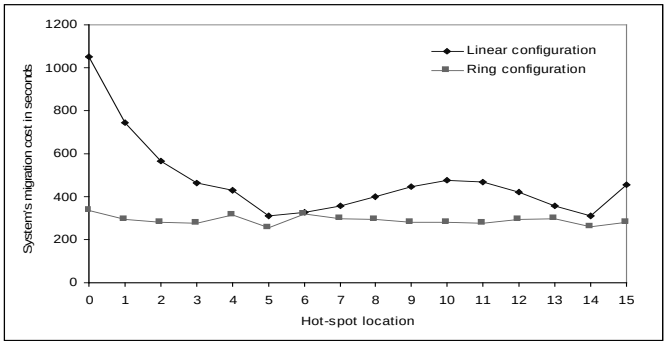
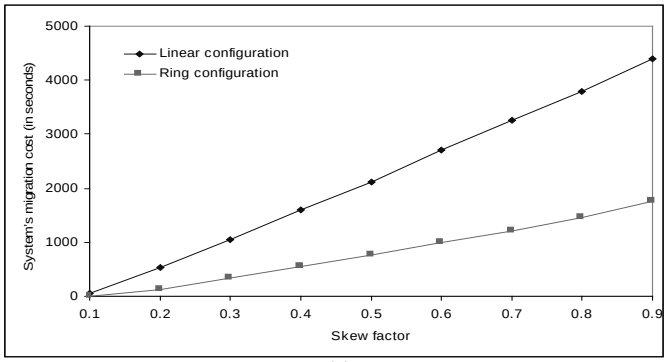
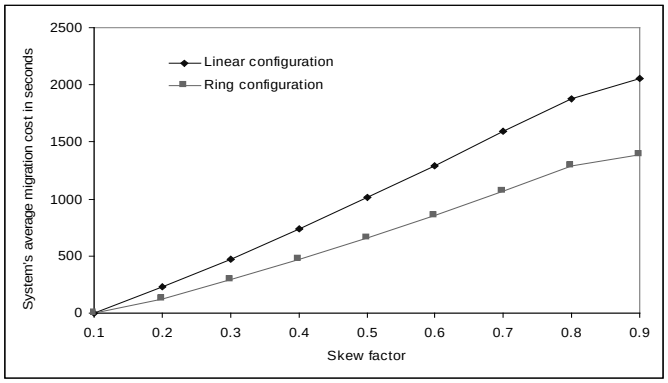


Figure 8: Effect of the hot-spot location on the system's migration cost.



(a)



(b)

Figure 9: Comparison of configuration's migration cost under variable skew access patterns. (a) PE0 is the hot spot PE. (b) average hot-spot location measurement.

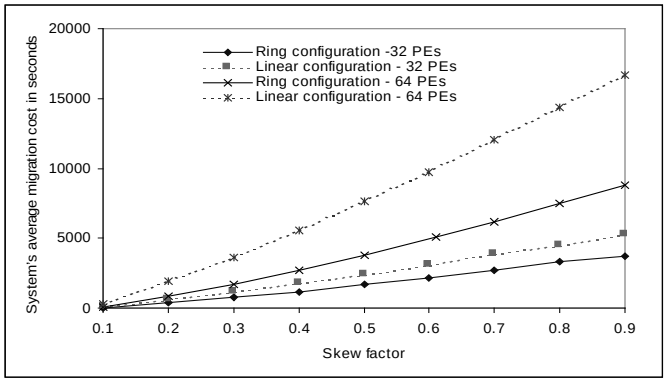


Figure 10: Migration cost on clusters of 32 and 64 PEs.

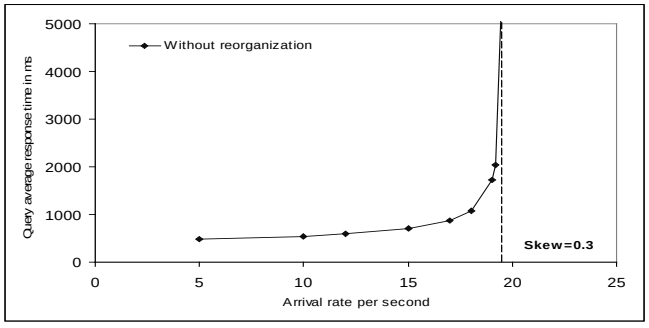
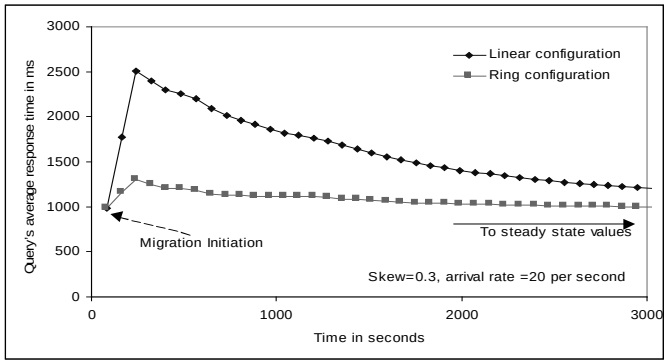
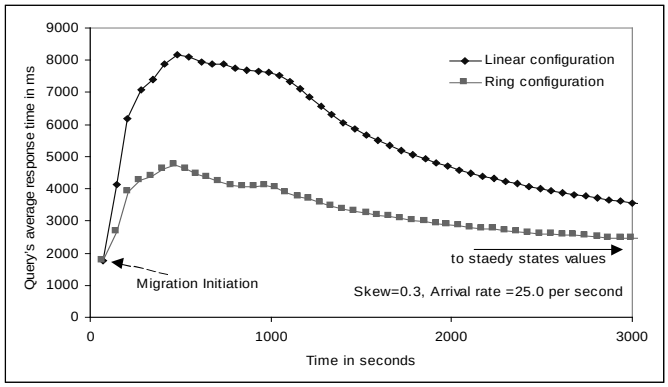


Figure 11: The considered system without reorganization: Query's average response time with its arrival rate.

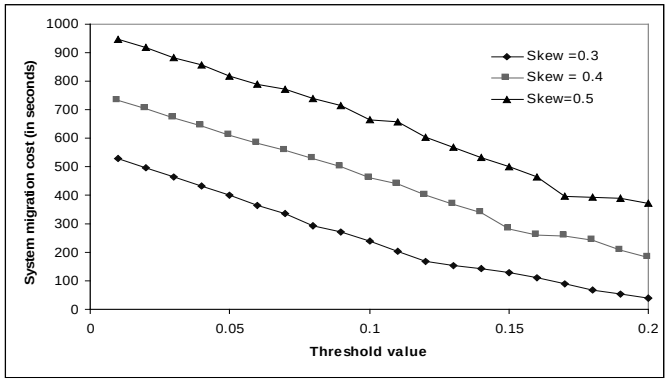


(a)

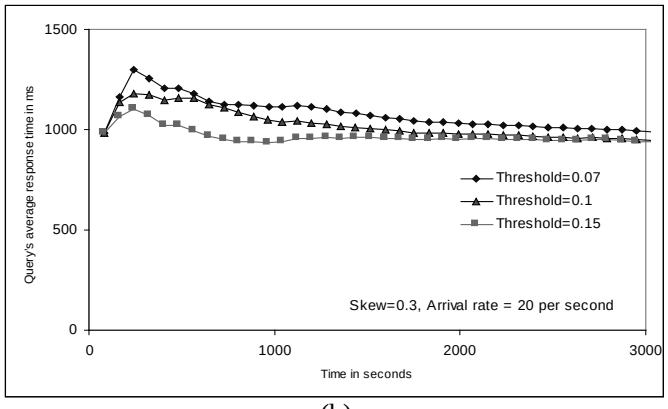


(b)

Figure 12: Migration cost effect on the system performance during reorganization process (a) arrival rate =20, arrival rate=25.



(a)



(b)

Figure 13: Threshold effect on the ring-configuration performance (a) Migration decisions cost (b) System response time during reorganization process.