OPTIMA: On-Line Balancing of Range-Partitioned Data with Imperfect Partitioning Vector

Djahida Belayadi, Khaled-Walid Hidouci, Khadidja Midoun
Ecole nationale Supérieure d'Informatique
Algeria
d_belayadi@esi.dz
w_hidouci@esi.dz
k_midoun@esi.dz

ABSTRACT: *Range query has a crucial role in large-scale data analysis. Unfortunately, the performance may be severely degraded by data skew. Such problem is often faced in large-scale parallel databases, peer-to-peer (P2P) systems as well as in Cloud computing. State-of-the-art methods designed to handle this problem offer significant improvements over naive implementations. However, performance could be further improved by reducing the cost of global skew knowledge and broadcasting. Ganesan, Bawa and Garcia-Molina proposed a load-balancing algorithm that guarantees a good ratio between the maximum and minimum loads among nodes. However, their algorithm requires global max-min load information to use local load balancing operations. Global load information can be found with O (log n) messages. In order to reduce this cost, we propose* OPTIMA*, a novel online load balancing approach for range partitioned data. Whenever a partition becomes overloaded, data transfers are performed, in background, from the most loaded nodes to the least loaded ones as in Ganesan et al., work. As a result, the partition boundaries and data sizes change. The key point of our proposal is the imperfect knowledge of the global load information (partition statistics). We introduce the concept of "Imperfect Partitioning Vector" (IPV), where, both nodes and clients have an approximate information about the load distribution. They can nevertheless locate any data with almost the same efficiency as using exact partition statistics. Furthermore, maintaining load distribution statistics do not require exchanging additional messages or maintaining a data structure as opposed to the cost of efficient solutions from the state-of-art (which require at least O (log n) messages).*

Keywords: Range Partitioning, Partition Statistics, Range Query, Imbalance Ratio

Received: 12 June 2018, Revised 15 July 2018, Accepted 20 July 2018

DOI: 10.6025/jcl/2018/9/4/135-147

1. Introduction

The explosion of Internet of Things technologies and social media has recently given rise to a new generation of big data applications. These applications tap into the wealth of historical and real-time data from geographically distributed data sources to provide actionable and real-time insights. In that case, data allocation and management is a key performance factor for distributed and parallel database [1]. The unabated and the rapid increase of the large flow of data has unleashed a new set of problems. The skewed distribution of large volumes of data is one of the main challenges that we face while ensuring optimal performance of querying the data.

Range partitioning maps tuples to partitions according to a partitioning key. This scheme is the most common type of partitioning and is often used with dates [2]. A key requirement in such systems is that the data has to be uniformly partitioned on all nodes. This requirement is challenging to enforce when the input data is skewed. Data skew is a well-known concern in range partitioning where only few partitions (nodes) may be more loaded than others. In that case, data migration approaches are an appealing solution. The out-of-range data has to be moved from the over-loaded area to under-loaded one in order to satisfy the storage balance requirement.

Data movement must be accompanied by a change in the partition statistics (partition boundaries, neighbors loads, and the position of the most and least loaded node, ..., etc.). All nodes/clients of the system must be aware of these changes in order to decide whether to call the balancing algorithm and address the right node. One of the dominant measures that has to be optimized in such a situation is communication cost. The focus is on the solutions that reduce the cost of maintaining data distribution information when resolving the skewed data problem.

Several methods have been proposed to deal with data skew problem while preserving data order. Data skew is a significant problem for multiple communities, such as parallel databases [3], Cloud computing [4], Peer-To-Peer systems [5], and text classification [6]. Most of the proposed approaches are based on tuples migration between nodes [7]–[10]. These works are generally using two universal strategies: node migration and neighbor item exchange. However, most of these approaches need global load statistics to maintain load balancing. The cost of getting perfect partition statistics in a fully distributed system is at least O (log n) messages (n is the number of nodes). Other approaches have been proposed to minimize the effect of skew, particularly, with range partitioning, such as the histograms [11]. The common challenge between all of these works is how to maintain partition statistics with a low communication cost. Work in [12] is an efficient technique to reduce the cost of maintaining partition statistics. Our work is complementary to this one. Ganesan et al., [10] proposed an online load balancing algorithm on a linearly ordered nodes/disks. Their algorithm called ADJUSTLOAD guarantees a low imbalance ratio between the maximum and the minimum load among nodes. Although, the ADJUSTLOAD algorithm is easy to state, each balancing operation may require global load information, that may be expensive in term of operations costs. Their algorithm uses a data structure called skip graph [13] to maintain load information and ensure efficient range queries. Each invoked balancing operation may require global information with a cost of O (log n) messages. Moreover, a change of partition boundaries between neighbors in load-balancing will necessitate a change in the two skip graphs used.

In this paper, we improve the Ganesan et al.'s work by reducing the cost of maintaining partition statistics. We propose OPTIMA, an on-line balancing technique of range partitioned data with approximate load information. Our algorithm uses the same primitive operations, NBRADJUST and REORDER as in Ganesan et al. The primitive NBRADJUST transfers the surplus of data from the current node to one of its neighbors. The primitive REORDER changes the nodes order to achieve the storage balancing requirements. As a result, the partition boundaries change as well as the data size of the affected partitions. However, our algorithm is not based on skip graphs to maintain partition statistics. The key point of our contribution is the imperfect partitioning vector (IPV), where, both nodes and clients have approximate information on data distribution statistics. Each entry IPV[i] in this vector, is an estimate of partition boundaries and data size related to node Ni. After a balancing operation, the participating nodes may change their own boundaries. These nodes do not need to inform the other ones by these changes. The clients use their IPV to direct the queries. As a result, clients may address the wrong node when their IPV are outdated. Nevertheless, whenever an interaction happens between two peers (node or client), they exchange their IPV in order to correct each other. Our solution outperforms the state-of-the-art methods in terms of communication cost. There is no additional cost for maintaing load statistic as in Ganesan et al.

By the solution we propose, we are targeting parallel range partitioned databases that have to deal with the frequent end users requests while being continuously supplied in real time from multiple sources (e.g., Wireless sensor networks where data is inserted continuously in a parallel database). Note that in this case, the major risk that can be faced is data skew. This would influence negatively the performance of parallel processing (such as range queries). We propose to improve Ganesan et al.'s work and other works that use skip graphs to avoid the additional cost related to maintaining the partition statistics. This allows heavy loaded nodes to dynamically adjust the boundaries of their ranges by migrating data to less loaded ones. We avoid using a central site or indexes to maintain load information. OPTIMA may be applicable to a wide range of applications and is transparent to the users. An example of such applications includes the road traffic monitoring services [14] and basic database operations like parallel sorting and parallel join in a heavy skewed environment.

This paper is organized as follows: in Section 2, we present some preliminaries required to understand our proposal. We describe

the system model in Section 3. In Section 4, we present the OPTIMA approach. We experimentally evaluate it in Section 5. Finally, we discuss the related work in Section 6.

2. Background and Motivation

2.1. Definitions

Partition: A partition is a node which is associated with a unique partition boundary for a given relation. As used herein, "node" refers to a partition.

Range Partitioning: A technique that distributes the tuples of a table according to the intervals containing the values of one or more attributes (partitioning key).

Range Query: A well-known database operation. It returns all tuples between two specified values (upper and lower boundaries). For example, list all employees with 5 to 10 years' experience. Executing range queries in an environment suffering from data imbalance significantly affects the running time of this request.

Load Balance Operation: It alters defined partition boundaries and moves tuples between nodes according to the new boundaries with the effect of reducing the system skew for highly loaded nodes.

Imbalance Ratio: A value measured as the ratio between the largest and the smallest load. The higher the ratio, the greater the degree of imbalance.

Neighbor Item Exchange: Balancing data distribution is done by transferring tuples from overloaded nodes to the less loaded ones. The need to preserve data order (for efficient processing of client range queries) requires that any element exchange has to be performed only between logically neighboring nodes i.e. nodes managing contiguous intervals.

Node Reorder: There are situations where under-loaded nodes are not contiguous with the overloaded node. In that case, one of the under-loaded distant nodes can move away from its logical area, join the overloaded node and take some of their keys. This operation seems more efficient; however, a large number of message exchanges are necessary for the location of the remote node and for the maintenance of the overlay structure. Both of the two concepts (neighbor item exchange and node reorder) are presented in the figure 1.

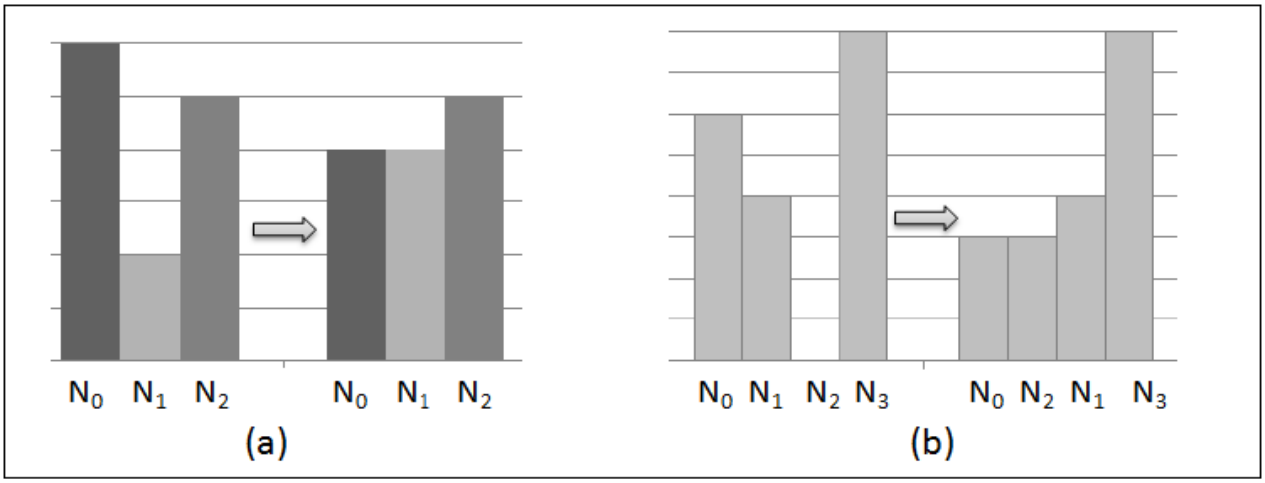

Figure 1. (a). Neighbor item exchange: the node N_0 averages its data with N_1. (b). Node reorder: N_2 changes its position and averages data with N_0

2.2. Load Balancing Solution of Ganesan et al.

Ganesan et al.'s work is believed to be representative of the prior art. It suggested an inspiring algorithm for reducing data skew.

It guarantees a small imbalance ratio between the largest load and the smallest one. This ratio is always bounded by a small constant which is 4.24. The algorithm uses two operations:

• **NBRADJUT:** The node N_i transfers its surplus of data to one of its neighbors (N_{i+1} or N_{i-1}, it depends on the least loaded neighbor). Data migration may change the boundaries of N_i and the neighbor receiving data.

• **REORDER:** The least loaded node (N_r) among all the nodes transfers its entire content to one of its neighbors and change its logical position to share data with the node performing the load balancing algorithm.

In both operations, the over-loaded node requires non-local information (neighbors load, the position of the most and least loaded node). A given node attempts to shed its load whenever its load increases by a factor δ. For some constant c, Ganesan et al. define a sequence of thresholds $T_i = c\delta^i$ *for all* $i < 1$. The node N_i attempts to trigger the ADJUSTLOAD procedure whenever its load $L(N_i)$ is greater than its threshold T_i. When $\delta = 2$, they call their algorithm the Doubling Algorithm. The ADJUSTLOAD procedure works as well when $\delta > \varphi = (\sqrt{5}+1)/2 = 1.62$, the golden ratio. They call their algorithm that operates at that ratio, the Fibbing Algorithm. They prove that the ADJUSTLOAD procedure running on that ratio would guarantee the imbalance ratio σ of $\delta^3 = 4.237$.

To analyze the cost of a load-balancing algorithm, Ganesan et al., propose three types of costs:

• **Data Movement:** Moving a key from one node to another is counted as a unit cost.

• **Partition Change:** Moving data from one node to another changes node's boundaries. This change has to be broadcasted to the other nodes so that the next insertion or deletion goes to the right node.

• **Load Information:** When executing the load-balancing algorithm, the node has to know neighbors loads and maybe the positions of the most and least-loaded node.

Ganesan et al. use two skip graphs. Skip graphs are circular linked lists [15], in which every node has *log(n)* pointers. Routing between two nodes needs *O (log n)* messages. The first skip graph is used to get neighbors loads (one message) and to route range queries to the appropriate node. The second skip graph is used to get the positions of the most and least loaded node in the system *(O (log n)* messages for locality plus costs of updating the two skip graphs).

3. System Model

In this section, we define a simple abstraction of a parallel database and make some considerations:

$V = \{N_1, N_2, \ldots, N_n\}$, a set of n nodes connected by a fast-local area network as in a Shared-Nothing architecture. We consider a relation (or a data set) divided into *n* range partitions on the basis of a key attribute, with boundaries $R_0 < R_1 < \ldots < R_n$. The node Ni manages a range $[R_{i-1}, R_i]$. We consider that the nodes are ordered by their ranges, this ordering defines left and right relationships between them.

• $U = \{C_1, C_2, \ldots, C_m\}$, a set of *m* clients performing insert, delete or range queries. Point queries can be considered as special case of range queries, where upper and lower bounds are equal. The clients may join or leave the system at any time. At this level, we do not consider the scalability of the system, which means that the nodes do not leave and join the network. However, we believe that our solution may be easily adapted to a scalable environment.

• Nodes and clients do not necessarily have the real information about the data distribution across the nodes (partition statistics). Instead, they have their own potentially imperfect partitioning vector (*IPVn* for nodes and *IPVc* for clients), which may differ from the real one.

• Each node N_j has its own imperfect partitioning vector *IPVn*. An entry *IPVn*[*i*] in this vector (for *i* different from *j*), is an estimate of partition boundaries and data size related the node N_i. The entry *IPVn*[*j*] contains exact information about partition boundaries

and data size of the node N_j.

• Each client has its own imperfect partitioning vector IPVc. An entry $IPVc[i]$ in this vector, is an estimate of partition boundaries and data size related to the node N_i. Clients use their imperfect partitioning vector IPVc to find the right nodes for insert, delete or search operations.

• We assume that each node N_i sets a local load threshold. When the load goes outside this limit, the node performs a load-balancing operation with its neighbors. This operation updates the partition boundaries of the participating nodes.

• Our load balancing algorithm is invoked on a node at which the insert or delete is occurring or a node receiving data from its neighbors. At this stage, we assume that no central site is used to direct queries. We also ignore concurrency control issues and consider only the serial schedule of inserts and deletes, interleaved with the executions of the load-balancing algorithm. An architecture of our solution is presented in Figure 2.

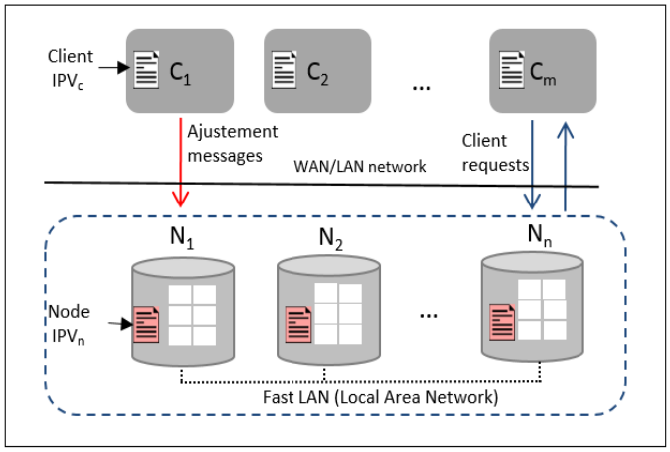

Figure 2. OPTIMA architecture

4. Our Approach

The main feature of the OPTIMA approach is the IPV concept, where each node or client has an imperfect knowledge about the partition statistics. Based on this knowledge, the node performs a load balancing whenever its load passes a local threshold. It invokes the NBRADJUST procedure that transfers the out-of-range data and its vector towards its neighbors if it is possible, else, it performs the REORDER procedure. The neighbor, after having received the data, performs the load balancing algorithm and eventually updates its vector. The process is repeated at each node receiving the data until the whole system is balanced.

Whenever a message is exchanged between two nodes or between a node and a client, the imperfect partitioning vector IPV is also included (piggybacked in the sent message), so that the node or the client receiving the message can compare it with its own statistics to hold the most recent value for each entry in its own vector (IPV). In this way, the most recent values, regarding the boundaries of partitions and their sizes, are thus propagated in the system asynchronously. We describe in the following sections the IPV concept.

4.1 Node Imperfect Partitioning Vector (IPVn)

Consider that the node partition statistics are encoded in a vector $IPVn[1, n]$. Each $IPVn[i]$ stores an estimate of the node Ni information. Node information is mainly the upper boundary ($IPVn[i]$.Upper_Bound), the local data size (IPVn[i].Load) and the last updating time ($IPVn[i]$.Last_Update). The last field ($IPVn[i]$.Last_Update) is used to indicate the time when the entry IPVn[i] was last updated. Each node has an index i so that $i \in [1, n]$, N_{i+1} is the logical successor and N_{i-1} is the logical predecessor. Node vector is updated in the following cases:

• Insert or delete queries. The client sends the data to insert or to delete with its IPVc. In that case, the local data size and the last updating time are adjusted accordingly. IPVn entries related to the other nodes are eventually updated using the received client

IPVc. The most recent value among the two vectors is used.

• Range queries sent by clients. In that case, only the older entries in the node vector are eventually updated using the received client IPVc.

• Data migration from the current node to one of its neighbors or vice versa. In that case, local data size, boundaries and last updating time are updated accordingly. Other vector entries are also eventually updated using the received node IPVn.

4.2 Client Imperfect Partitioning Vector (IPVc)

The data stored in the nodes is manipulated through the insert, delete and range queries sent by the clients. Consider that the client partition statistics are encoded in a vector *IPVc*[1, *m*], where each *IPVc*[*i*] stores the information about the node N_i. Node information is essentially its upper boundary (*IPVc*[*i*].*Upper_Bound*), data size (*IPVc*[*i*].*Load*) and the last updating time (*IPVc*[*i*].*Last_update*). Inserting, deleting or searching for a tuple with a given key *k* are performed as follows:

• The client searches for a node N_i so that: *IPVc*[*i*-1].Upper_Bound < *k* < IPV c[i].Upper_Bound. Then, it sends a message to N_i containing the tuple to insert, delete or to search. The local IPVc vector is also included in the same message.

• A node N_i receiving the request, checks whether the included key *k* fits its range, if so, it executes the specified request (insert, delete or just point-search), updates eventually its partition statistics (the *IPVc*[*i*].Load and *IPVc*[*i*].Last_update fields in the case of an insert or delete request) and sends a positive acknowledge consisting of its IPVn to the client.

• If *k* is outside the node's range, a vector adjustment message (*VAM*) including a negative acknowledge and the current *IPVn* is sent to the client.

• If a client receives a positive acknowledgment from the node, it just updates its vector if it was outdated. Else, if it receives an adjustment message, it updates its vector and repeats the operation until receiving a positive acknowledgment.

4.3 Data Load Balancing Algorithm

Our load balancing algorithm uses the two universal load balancing primitives, REORDER and NBRADJUST as in Ganesan et al.'s work. However, we use the IPV concept to maintain the global load information instead of using the skip graphs. Our algorithm, that we call ***DataLB***, is presented bellow (Algorithm 1). A node N_i executes the load balancing algorithm whenever its load increases beyond a threshold T_i. The algorithm uses its partitioning vector to check if data can be shared with the lightly loaded neighbors (line 2). If the load of one of its neighbors is less than half of N_i's load, then N_i performs the primitive NBRADJUST to average out the load with it (lines 3-10). Else, N_i attempts to perform REORDER with the least loaded node in the system. The algorithm gets the position of the least-loaded node (N_r) from the node *IPVn* (line 12), if the load of N_r is less than a quarter of N_i's load, N_r sends all of its data to its lightly loaded neighbors (line 15) and changes its position to take over half of N_i's load (line 16-17). If N_i is unable to perform neither item exchange nor node reorder, we conclude that the system load is balanced.

5. Experimental Evaluation

In this Section, we present results from our simulation of the *OPTIMA* approach on a network of 8 nodes and 2 clients. Processing node software and client software were executed on machines with Intel(R) Core(TM) i7-5500U CPU@2.40GHz and 8GiB of RAM. Both nodes and clients were connected through a Gigabit Ethernet network. Algorithms are implemented in C language using the Message Passing Library (Open MPI).

In the experiments, we present the algorithm for the insert-only case. This case is simpler to analyze and provides general ideas on how to deal with the general case. It is also of practical interest because in many applications, as in a file sharing, deletions rarely occur. In order to evaluate the new approach in heavy skewed environment, the system is studied under a simulation model that we call HOTSPOT. All insert operations are directed to a single hot node. We use a sequence of 5*104 frequent insert operations. We consider that the messaging costs are independent of the network and CPU performance. The basic performance factors of our load balancing mechanism are:

The imbalance ratio between the most loaded node and the least loaded one.

```
1 Let N_j be the lightly loaded of N_i +1 and N_i -1;
2 if (IPVn[i].Load/2 > IPVn[j].Load) then
3       NB = (IPVn[i].Load - IPVn[j].Load)/2;
4       Send NB tuples to N_j;
5       IPVn[i].Load = IPVn[i].Load - NB;
6       IPVn[j].Load = IPVn[j].Load + NB;
7       Update IPVn[i].Upper_Bound;
9       DataLB (N_i, IPVn[i]);
10      DataLB (N_j, IPVn[j]);
11 else
12      Find N_r so that ∀ k ∈ [1; n]; IPVn[k].Load > IPVn[r].Load;
13      if (IPVn[i].Load/4 > IPVn[r].Load) then
14      Let N_j be the lightly loaded node between N_r+1 and N_r-1;
15      Send IPVn[r].Load tuples to N_j;
16      N_j changes its position to be N_i's neighbor;
17      Send IPVn[i].Load/2 to N_r;
18      IPVn[i].Load = IPVn[i].Load - (IPVn[i].Load/2);
19      IPVn[r].Load = IPVn[i].Load/2;
20      Update IPVn[i].Upper_Bound;
21      Update IPVn[j].Upper_Bound;
22      Update IPVn[r].Upper_Bound;
23      DataLB (N_i, IPV [i]);
24      Rename nodes appropriately after the REORDER;
25    else
26      The system is balanced;
27   end
28 end
```

Algorithm 1. *DataLB* (Node N_i)

• The number of addressing errors, that a client may make when sending a request to the nodes.

• Data movement, all the load balancing algorithms need to move data from one node to another in order to achieve balance.

• The number of invocation of *DataLB* algorithm.

5.1. Imbalance Ratio

First of all, we evaluate the imbalance ratio σ. We measure the imbalance ratio as the ratio between the largest and smallest load after each insert operation. We consider that, at the beginning, all loads are at least 1. As the system's thresholds are an infinite, increasing geometric sequence, as in Ganesan et al.'s work, we measure the imbalance ratio with three values of the factor δ (δ =φ, δ=2, δ = 4). φ is the golden ratio, $\varphi = (\sqrt{5}+1)/2 = 1.62$. Figure 3 shows the imbalance ratios (Y-axis) against the number of insert operations (X-axis).

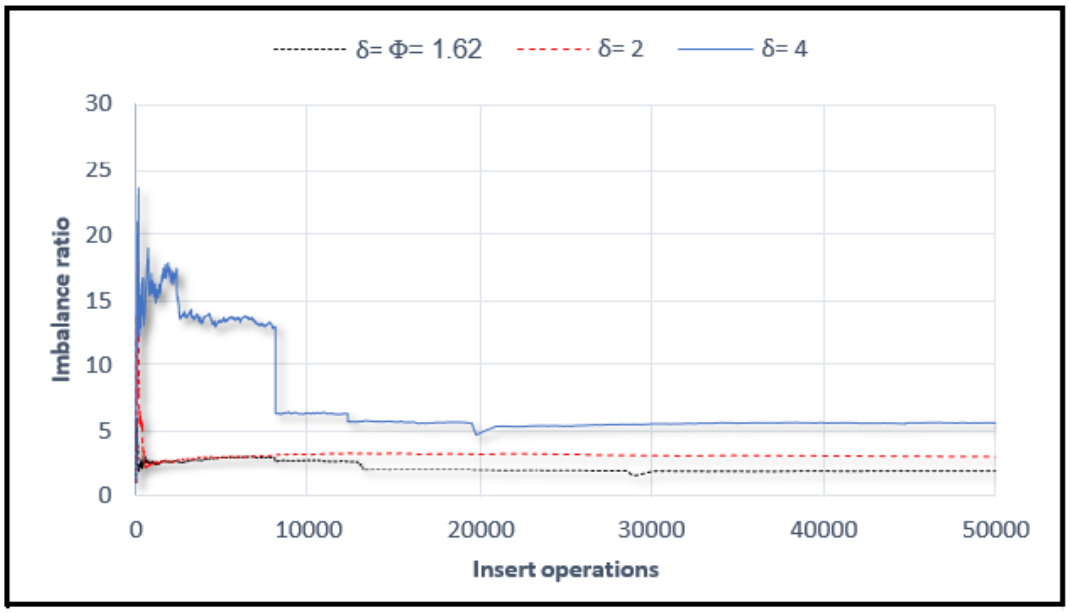

Figure 3. The imbalance ratio when δ = φ, δ =2 and δ = 4.

When δ = φ, the curve shows that the imbalance ratio is always bounded by a constant 6 and it converges to 1.8 after 2*104 operations unlike the imbalance ratio of Ganesan et al., that is bounded by 4.24 and converges towards 3.3. The spikes in the curve mean that an invocation of DataLB algorithm has been lunched. The curve presents several variations at the beginning because the nodes are in the growing phase where data is loaded. At the beginning, all the insert operations are sent towards the node N_1. The successive invocations of the load balancing algorithm lead to a variation of the partition boundaries (the range of N_1 will be reduced so that the data will be oriented towards one of the neighbors). When δ = 2, the growing phase presents big variations in the ratio, thereafter, the ratio values converge to 2. However, the results of the 3rd experiment δ = 4 are different from the previous ones, the ratio values are larger and converge towards 5.

5.3 Client Adjustment Messages

After a balancing operation, two nodes at least change their partition boundaries due to the data migration, which leads to changing the partitioning vectors of these two nodes. The client with an outdated vector can address a wrong node that has changed its boundaries. In our set of experiments, we were interested in determining how efficiently a client obtains a true view about the nodes. We measure the number of times the client sends a query to a wrong node and hence makes an addressing error and receives a vector adjustment message (VAM). The results showed in figure 4 present a rapid increase of addressing errors number in the growing phase (from 1 to 1000 insert operations). This comes back to the fact that there is a frequent invocation of the balancing algorithm, and therefore a frequent change of the partition boundaries. We consider that client's vector converges to the true state after the growing phase.

5.4 Performance Analysis

For balancing loads among nodes, we are also concerned with the minimization of the movement cost as much as possible. After measuring the imbalance ratio and the client vector adjustment, we next measure the data movement cost. Figure 5 plots the cumulative number of tuples migrated by our algorithm (Y-axis) against the number of insert operations (X-axis) during a run with δ = 1.62 and δ = 2. We observe that in the growing phase, the number of migrated tuples for different values δ is rising, this is

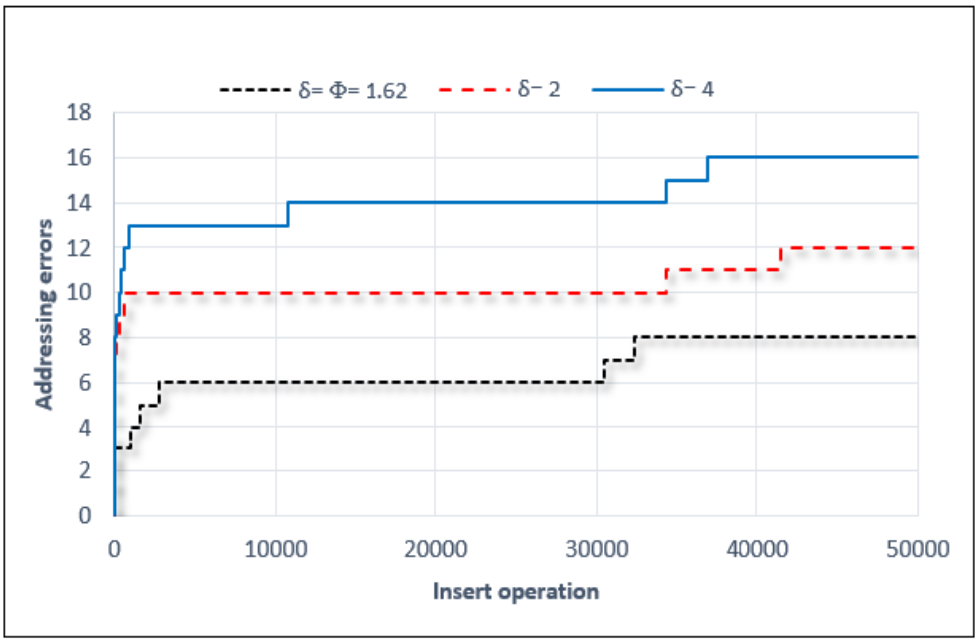

Figure 4. The number of addressing errors when $\delta = \varphi, \delta = 2$

because keeping the system tightly balanced causes a larger number of rebalancing operations. Figure 6 plots the data movement cost when $\delta = 4$.

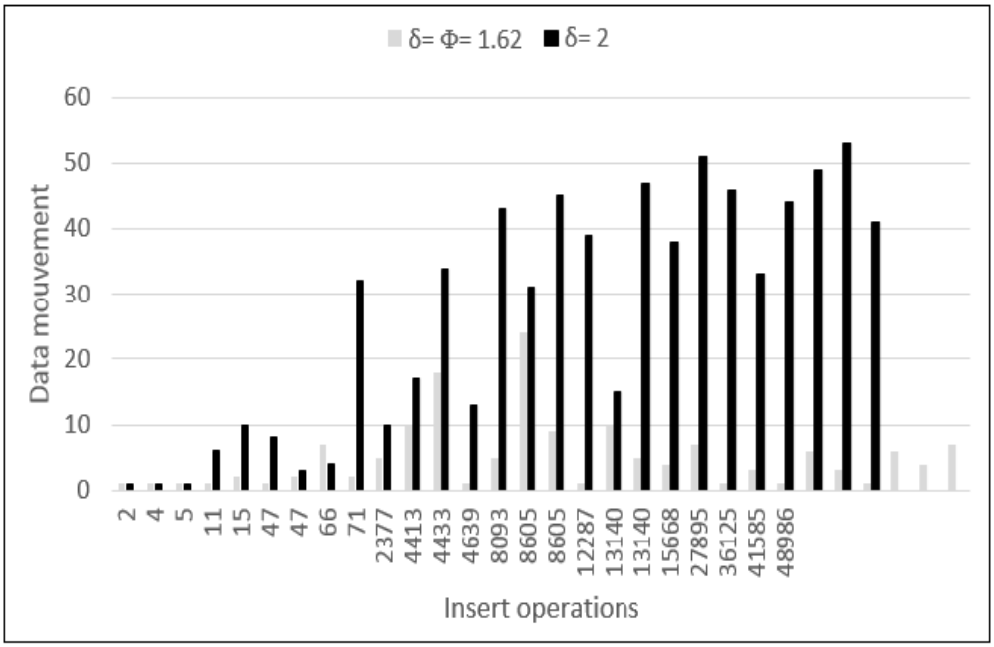

Figure 5. Data movement cost when $\delta = \varphi$, $\delta = 2$.

Figure 7 illustrates the number of invocations of our load balancing algorithm (Y-axis) against the number of insert operations (X-axis) during a run. The observation we make is that the number of invocations of the algorithm increases for the three values of δ during the growing phase. The number of algorithm invocations presented in the figure 7 was measured when $\delta = 1.62$, $\delta = 2$ and $\delta = 4$. We observe that the number of invocations is relatively high when $\delta = 1.62$. This comes back to the fact that we are supporting some imbalance situations.

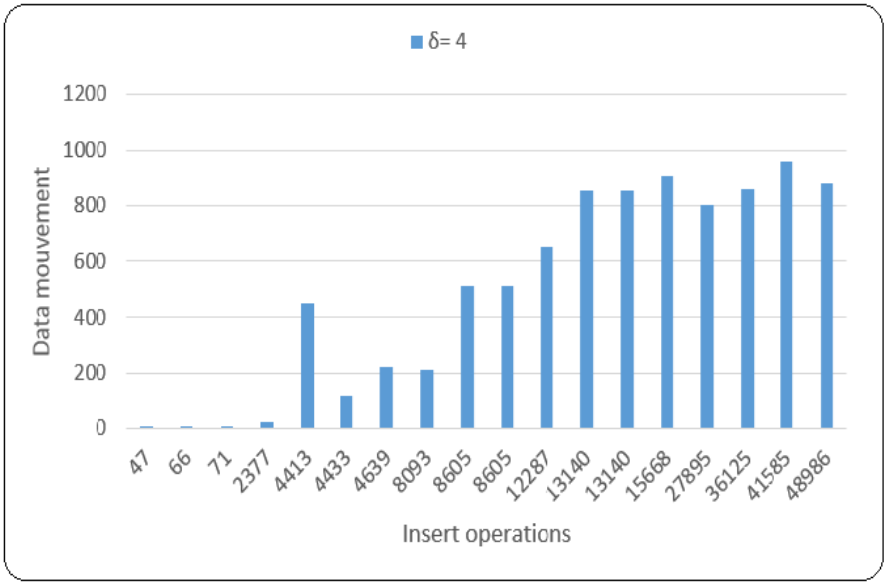

Figure 6. Data movement cost when δ = 4

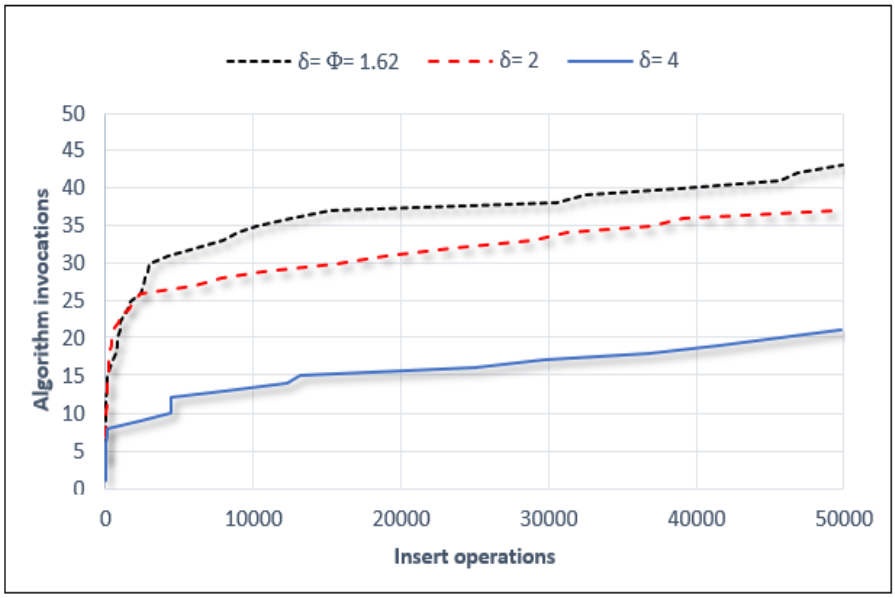

Figure 7. Number of DataLB invocations when δ =φ, δ =2 and δ = 4.

6. Related Work

In this section, we describe the current researches that relate to ours achieved load balancing method while supporting range queries.

Cloud Computing: Data-load balancing is one of the key problems of Cloud Computing technology. Clients from different parts of the world are demanding for various services in a rapid rate. That is why efficient solution for handling data skew in Cloud Computing environment should be considered [4]. With recent advances in Cloud Computing, MapReduce has emerged as a powerful tool for distributed and scalable processing of voluminous data [16]. The real-world data are not necessarily uniform and often exhibit remarkable skew. Such skewed distribution of the input or intermediate data can make a small number of mappers or

reducers take a significantly longer time to complete than others. There have been many pioneer works dealing with the data skew in MapReduce [17]–[19]. Entity resolution [20] is to identify the entities referring to the same entity in the dataset. Many data-load balanced MapReduce algorithms have been proposed in this emerging field of research. Authors in [21] propose two load balancing approaches for data skew handling in MapReduce-based entity resolution: *BlockSplit* and *PairRange*, both approaches are able to deal with skewed data distributions and effectively distribute the workload among all reduce tasks by splitting large blocks. Many other techniques have been proposed such us [21], [22].

Peer-To-Peer network: In P2P networks [23], a number of recent load balancing approaches have been proposed [7], [24]–[28]. Structured P2P networks [29] are an efficient tool for storage and location of data since there is no central server which could become a bottleneck. Many researches have been proposed on search methods in Structured P2P networks, Readers looking for more information are referred to the survey on searching in P2P networks by [5]. The basic load balancing approach in structured P2P networks or distributed hash tables (DHTs) [30] is consistent hashing [24]. Unfortunately, consistent hashing approach destroys data order by randomizing placement of data items which is not applicable to some applications. For example, to support range searching in a database application, the items need to be in a specific order. SkipNet [31] and skip graph [13] are data structures for range searching that are not based on DHTs. Both of them are adapted from Skip Lists [15]. The computational cost of object search in a skip graph and SkipNet networks is O (log n), where n is the number of nodes in the network. Many other data structures answer range queries and ensure a good load balancing in their experiments, e.g., Mercury [32], Baton [33], Chordal graphs [34]. However, Mercury requires extra communication cost to estimate the density of the nodes on the identifier ring. Baton is based on a binary balanced tree structure. It guarantees that exact queries and range queries can be answered in O (log n) steps and also that update operations have a cost of O (log n). Chordal graph is based on hierarchical neighborhood search to provide incoming nodes an overview of the network topology. Both Baton and Chordal graph have a cost of O (log n) for searching and routing. However, our method requires a very low cost for maintaining load statistics to answer range queries and ensure data load balancing.

A concurrent work to Ganesan et al.'s work is presented by Karger and Ruhl [35]. The load balancing algorithm is randomized, it offers a high bound on the imbalance ratio (more than 128). In [7], the team Jakarin et al. has improved the work of Ganesan et al. They consider that ADJUSTLOAD algorithm is not straightforward to be implemented in real networks because it is recursive. So, they present a non-recursive approach using the same primitive operations as in Ganesan et al.'s work. The proposed algorithm guarantees an unbalance ratio of 7.464. However, Jakarin et al use the skip graphs just like Ganesan et al.'s work. The main drawback is the costly maintenance operations of these data structures.

Parallel/Distributed Databases: A load balance operation in a parallel database is performed as a transaction. Its consequences do not affect other concurrent transactions until all tuples are moved, partition boundaries are updated, and the transaction is committed. Read requests for tuples that are being moved to another node are serviced from existing versions residing at the old node. Write requests for tuples that are being moved either abort, wait, or are forwarded to the new location. Work in [3] propose a multi-reorder operation that finds a sequence of multiple adjacent nodes that have a small average load of any such sequence. This technique uses partition statistics which include an estimate of the number of tuples stored on each node for every relation in the database. Based on this information the system skew is calculated. The problem that could be noted is the cost of maintaining partition statistics.

Discussion: The above state-of-art presents the existing approaches dealing with the problem of data skew in a range-partitioned data system. These approaches are targeted towards Cloud computing, P2P systems, and parallel databases. The conclusion we could make is that the cost of maintaining load statistics is at least O (log n). Our solution provides load balancing mechanism to handle the data skew for applications where append operations and range queries are dominant. This concerns applications that do not require updating queries but mainly insert, delete and search queries. We provide a strategy for maintaining the global load statistics without any extra communication cost, unlike the state-of-art solutions requiring at least O (log n) additional messages.

7. Conclusion

The present paper relates to load balancing in parallel database systems. We proposed an effective on-line data load balancing algorithm that deals with the problem of skewed data. Our experimental results that we set in our laboratory show that our approach does not need extra cost of maintain partition statistics as opposed to the cost of efficient solutions from the state-of-art. OPTIMA procedure needs a very low overhead (or almost no cost) to locate the data even in the presence of high degree

of skew. Since IPV vectors spread between nodes by being encapsulated in ordinary messages, a variant manipulating aggregated information is being investigated to minimize the size of the exchanged IPVs. This should drastically reduce the network load of the proposed method, degrading, however a little bit the accuracy of the node's addressing scheme.

References

[1] Abdalla, H. I. (2011). Improving Data Management in a Distributed Environment., *J. Digit. Inf. Manag.*, 9 (3).

[2] Bensberg, C., Becker, J., Mueller, C., Thumfart, A. (2017). *Dynamic range partitioning*, November.

[3] Rishel, W. S., Rishel, R. B., Taylor, D. A. (2014). Load balancing in parallel database systems using multi-reordering, US Patent 8,849,749, September 2014.

[4] Sun, J., Afnan, O., Lin, Y. (2016). *Data skew finding and analysis*, US Patent App. 15/199,507, Jun-2016.

[5] Risson, J., Moors, T. (2006). Survey of research towards robust peer-to-peer networks: Search methods, *Comput. Netw.*, 50 (17) 3485–3521.

[6] Chang, F., Guo, J., Xu, W., Yao, K. (2015). A feature selection method to handle imbalanced data in text classification, *J Digit Inf Manage*, 13 (3) 169–175.

[7] Chawachat, J., Fakcharoenphol, J. (2015). A simpler load-balancing algorithm for range-partitioned data in peer-to-peer systems, *Networks*, 66 (3) 235–249, October 2015.

[8] Konstantinou, I., Tsoumakos, D., Koziris, N. (2011). Fast and cost-effective online load-balancing in distributed range-queriable systems, *IEEE Trans. Parallel Distrib. Syst.*, 22 (8) 1350–1364.

[9] Hsiao, H.-C., Chung, H.-Y., Shen, H., Chao, Y.-C. (2013). Load rebalancing for distributed file systems in clouds, *IEEE Trans. Parallel Distrib. Syst.*, 24 (5) 951–962.

[10] Ganesan, P., Bawa, M., Garcia-Molina, H. (2004). Online balancing of range-partitioned data with applications to peer-to-peer systems, *In*: Proceedings of the Thirtieth international conference on Very large data bases-Vol. 30, 2004, 444–455.

[11] Koudas, N., Muthukrishnan, S., Srivastava, D. (2000). Optimal histograms for hierarchical range queries, *In*: Proceedings of the nineteenth ACM SIGMOD-8 SIGACT-SIGART symposium on Principles of database systems, 196–204.

[12] Belayadi, D., Hidouci, W. (2016). Dynamic Range Partitioning with Asynchronous Data Balancing, *In*: Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016 Intl IEEE Conferences, p 1214–1220.

[13] Aspnes, J., Shah, G., Skip graphs, *ACM Trans. Algorithms Talg*, 3 (4) 37.

[14] Wang, F.-Y. (2010). Parallel control and management for intelligent transportation systems: Concepts, architectures, and applications, *IEEE Trans. Intell. Transp. Syst.*, 11 (3) 630–638.

[15] Pugh, W. (1990). Skip lists: a probabilistic alternative to balanced trees, *Commun. ACM*, 33 (6) 668–676.

[16] Ekanayake, J., Pallickara, S., Fox, G. (2008). Mapreduce for data intensive scientific analyses, *In*: eScience, 2008. eScience'08. IEEE Fourth International Conference on, p. 277–284.

[17] Huang, T.-C., Chu, K.-C., Huang, G.-H., Shen, Y.-C., Shieh, C.-K. (2017). Smart Partitioning Mechanism for Dealing with Intermediate Data Skew in Reduce Task on Cloud Computing, *In*: *A*dvanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference on, 2017, p. 819–826.

[18] Ramakrishnan, S. R., Swart, G., Urmanov, A. (2012). Balancing reducer skew in MapReduce workloads using progressive sampling, *In*: Proceedings of the Third ACM Symposium on Cloud Computing, p 16.

[19] Kwon, Y., Balazinska, M., Howe, B., and Rolia, J. (2012). Skewtune: mitigating skew in mapreduce applications, *In*: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, 2012, p 25–36.

[20] Goeppinger, C. W., Hwang, J. C., Ayumi, A. Y., McGrath, J. H., Benner, T. B., Pirio, G. (2017). Entity resolution techniques and

systems, 9734207, August 2017.

[21] Kolb, L., Thor, A., Rahm, E. (2012). Multi-pass sorted neighborhood blocking with MapReduce, *Comput. Sci.-Res. Dev.*, 27 (1) 45–63.

[22] Myung, J., Shim, J., Yeon, J., Lee, S. (2016). Handling data skew in join algorithms using MapReduce, *Expert Syst. Appl.*, 51, 286–299.

[23] Rodrigues, R., Druschel, P. (2010). Peer-to-peer systems, *Commun. ACM*, 53 (10) 72–82.

[24] Felber, P., Kropf, P., Schiller, E., Serbu, S. (2014). Survey on load balancing in peer-to-peer distributed hash tables, *IEEE Commun. Surv. Tutor.*, 16 (1) 473–492.

[25] Mirrezaei, S. I., Shahparian, J. (2016). Data Load Balancing in Heterogeneous Dynamic Networks, *ArXiv Prepr. ArXiv160204536*.

[26] Antoine, M., L. Pellegrino, F. Huet, and F. Baude, A generic api for load balancing in structured p2p systems, *In*: Computer architecture and high performance computing workshop (sbac-padw), 2014 International Symposium on, 2014, p. 138–143.

[27] Takeda, A., Oide, T., Takahashi, A., Suganuma, T. (2015). Efficient Dynamic Load Balancing for Structured P2P Network, *In*: Network-Based Information Systems (NBiS), 2015 18th International Conference on, 2015, p. 432–437.

[28] Mizutani, K., Inoue, T., Mano, T., Akashi, O., Matsuura, S., Fujikawa, K. (2016). Stable Load Balancing with Overlapping ID-space Management in Range-based Structured Overlay Networks, *Inf. Media Technol.*, 11, p. 1–10, 2016.

[29] Korzun, D., Gurtov, A. (2012). Structured peer-to-peer systems: fundamentals of hierarchical organization, routing, scaling, and security. Springer Science & Business Media.

[30] Zhang, H., Wen, Y., Xie, H., Yu, N. (2013). *Distributed hash table: Theory, platforms and applications*. Springer.

[31] Harvey, N. J., Jones, M. B.,Saroiu, S., Theimer, M., Wolman, A. (2003). Skipnet: A scalable overlay network with practical locality properties, *Networks*, 34 (38).

[32] Bharambe, A. R., Agrawal, M., Seshan, S. (2004). Mercury: supporting scalable multi-attribute range queries, *ACM SIGCOMM Computer Communication Review,* 34, p. 353–366.

[33] Jagadish, H. V., Ooi, B. C., Vu, Q. H. (2005). Baton: A balanced tree structure for peer-to-peer networks, *In*: Proceedings of the 31st International Conference on Very large data bases, p. 661–672.

[34] Joung,Y.-J. (2008). Approaching neighbor proximity and load balance for range query in P2P networks, *Comput. Netw.*, vol. 52 (7) 1451–1472.

[35] Karger, D. R., Ruhl, M. (2006). Simple efficient load-balancing algorithms for peer-to-peer systems, *Theory Comput. Syst.*, 39, 6, p .787–804.