Computer Science and Information Systems 15(2):393–419 https://doi.org/10.2298/CSIS170320007B

OLAPS: Online Load-Balancing in Range-Partitioned Main Memory Database with Approximate Partition Statistics

Djahida Belayadi[1], Khaled-Walid Hidouci[1], and Ladjel Bellatreche[2]

[1] Laboratoire de la Communication dans les Systmes Informatiques
Ecole nationale Supérieure d'Informatique, BP 68M, 16309, Oued-Smar
Algiers, Algeria
(d_belayadi,w_hidouci)@esi.dz
[2] LIAS/ISAE-ENSMA – Poitiers University
86960 Futuroscope, France
bellatreche@ensma.fr

Abstract. Modern database systems can achieve high throughput main-memory query execution by being aware of the dynamics of highly parallel hardware. In such systems, data is partitioned into smaller pieces to reach a better parallelism. Unfortunately, data skew is one of the main problems faced during parallel processing in a parallel main memory database. In some data-intensive applications, parallel range queries over a dynamic range partitioned system are important. Continuous insertions/deletions can lead to a very high degree of data skew and consequently a poor performance of parallel range queries. In this paper, we propose an approach for maintaining balanced loads over a set of nodes as in a system of communicating vessels, by migrating tuples between neighboring nodes. These frequent (or even continuous) data transfers inevitably involve dynamic changes in the partition statistics. To avoid the performance degradation typically associated with this dynamism, we provide a solution based on an approximate Partition Statistics Table. The basic idea behind this table is that both clients and nodes may have an imperfect knowledge about the effective 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 as opposed to the cost of efficient solutions from the state-of-art (which requires at least $O(logn)$ messages). We show through intensive experiments that our proposal supports efficient range queries, while simultaneously guaranteeing storage balance even in the presence of numerous concurrent insertions/deletions generating a heavy skewed data distribution.

Keywords: Load balancing, Parallel main memory database systems (MMBD), Range partitioning, Data skew, Range query.

1. Introduction

Memory resident database systems (MMDBS) store the data in main physical memory and thus, provide very high-speed access. Increasing main memory capacities of up to several Terabyte per server and highly parallel processing exploiting multi-core architectures dominate today's hardware environment and will shape database system technology

in the near future. Over the past decade, prominent research systems such as H-Store [27] and HyPer [28] reinvigorated research into main memory and multi-core data processing. Most major database vendors now have an in-memory database solution, such as SAP HANA [16] and Microsoft SQL Server Hekaton [14].

A frequently voiced problem in in-memory resident data processing is uneven distribution of data referred to as Data Skew. This well-known phenomenon can arise from natural data distributions and/or non-uniform insertions/deletions. It can result in some nodes taking a lot more time to perform their tasks, thereby introducing bottlenecks in the parallel processing, and then delaying completion of the overall tasks [48].

Several techniques were proposed to ensure efficient data partitioning across nodes (round-robin, hash partitioning, and range partitioning, etc.). The existing experiences show that the range partitioning is the best suitable for point and range queries in a highly dense domains [44]. But in the same time, it is very sensitive to data skew. In the case, where the partitioning by range creates partitions with size varying dramatically (due to the no-uniform distribution) the load balancing methods should be considered.

Data skew can be quantified by the values of an imbalance ratio. It represents the ratio of the loads of the largest and smallest partitions in the system. The higher ratio implies the greater the degree of imbalance. In order to decrease the imbalance ratio values, data may have to be moved from one node to another as the data volume grows or shrinks. Thus, a key requirement for a load balancing algorithm is maintaining load statistics (e.g., partitions boundaries and loads). In recent studies, the emphasis is mainly focused on minimizing the number of messages exchanged for maintaining the global load statistics.

Works in [18,29,11] use two universal load balancing primitives which are item exchange between neighbors and node migration from the most loaded area to the least loaded one. Despite the effectiveness of these approaches, their main drawback is the high cost of maintaining load statistics. They are based on skip graphs [2], where the partition changes are followed by an update of the data structure requiring $O(log\, n)$ messages. The proposal of [38,37] is based on the replication. The disadvantage of this proposal is the necessity to deal with consistency issues during the data update. Other approaches have been proposed to minimize the effect of skew, particularly with range partitioning, such as the virtual processors [20,38] and the histograms [31,21]. The common challenge between all of these works is how to maintain partition statistics with a low communication cost. In our previous work [5], we popose to reduce the cost of maintaining load statistics in a Scalable and Distributed Data Sructures [34] driven approach in which data is partitioned using dynamic bounds. Our current work is an improuvement of this one.

In this paper, we present $\mathcal{OLAPS}$, an **O**n-**L**ine balancing algorithm of skewed data with **A**proximate **P**artition **S**tatistics. It is a simple but powerful strategy for maintaining good load balancing of range partitioned In-memory resident data, despite the frequent and skewed changes in data volume that may occur. Our solution behaves exactly like a system of communicating vessels. Whenever a partition becomes overloaded, data transfers are performed, in background, from the more loaded nodes to the least loaded ones. As a result, the partition boundaries change. In some critical cases of very high degree of imbalance, these data transfers between nodes, therefore, adjustments of partition boundaries never stop. To efficiently implement range queries, even in such critical situations, we propose a key concept, based on the use of "approximate partition statistics" (called Partition Statistics Tables: $\mathcal{PST}$). Consider two sets of nodes and clients. The nodes are

dedicated to the data processing and storage. The clients send mainly insert, delete and range queries to the nodes. Each node or client has its own partition statistics table. Each entry $PST[i]$ in this table, is an estimate of partition boundaries and data size related to node N_i. After a balancing operation, the participating nodes may change their own boundaries. Those nodes do not need to inform the other ones by these changes. This will make the partition statistics stale. As a result, clients may address a wrong node when their $\mathcal{PST}$ are outdated. Nevertheless, whenever an interaction happens between two peers (node or client), they exchange their $\mathcal{PST}$ in order to correct each other. So, the more a peer is active, the more its $\mathcal{PST}$ converges towards the real state.

Our solution provides load balancing mechanism to handle the data skew for applications that do not require update queries. The main usages of our solution cconcern mainly insertions, deletions and data queries. These applications are useful in several fields, where range queries are used to extract information from a main-memory parallel database fed by sensors or other continuous data sources. Road traffic monitoring services are an example of such applications [52,35] as well as Big Data applications [8]. The captured information (car's serial number, location and so on) is inserted into a database in order to be on-line analyzed by many points and range queries. For example, a driver may send the query "return the speed observed by cameras that are between Exit 1 and Exit 10 ($1 \leq$ exit ≤ 10)", so that he may choose the right highway to avoid congestion down the road. A police patrolling a highway section with speed limit of 80 mph may ask the system to "return the list of cars running with a speed higher than 80 mph (speed > 80)".

Our contributions are summarized in the following points:

- We propose an on-line algorithm to ensure a load balancing of range partitioned data in a parallel main memory database.
- We provide a strategy for maintaining the global load statistics. This strategy does not require any extra communication cost, unlike the state-of-art solutions requiring at least $O(logn)$ additional messages.
- We use this strategy to design an algorithm for range queries which remains efficient even in the presence of rapid and continuous change of partition boundaries.
- We implement and test our fully decentralized algorithm under extensive experiments over a wide set of scenarios.

The remainder of this paper is organized as follows: Section 2 presents an abstraction of a parallel main memory database and some hypothesis. In Section 3, we detail our load balancing approach, we show how the approximate partition statistics concept reduce the maintenance cost of the load distribution information, and describe how the performance of range queries can be improved. Our proposal is evaluated under extensive experiments and different scenarios in section 4. Section 5 outlines related work, then, we conclude in Section 6 by summarizing the most important findings and suggesting some open issues.

2. System Model

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

- Let $N = \{N_1, N_2, \ldots, N_n\}$ be a set of n nodes connected by a fast local area network as in a Shared-Nothing architecture [51]. We consider a relation divided into n range

partitions on the basis of a key attribute, with boundaries $R_0 \leq R_1 \leq \ldots \leq R_n$. The node N_i manages a range [R_{i-1},R_i[. Data is In-memory resident, it is organized row wise. We consider that the nodes are ordered by their ranges, this ordering defines left and right relationships between them. At this level, we ignore the scalability of the environment, which means that there are no nodes that join or leave the cluster. However, our solution is can be easily extended to answer the scalability requirements;

- Let $C = \{C_1, C_2, \ldots, C_m\}$ be 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 and leave the system at any time;
- The nodes and the clients do not necessarily have the exact information about the data distribution across the nodes (partition statistics). Instead, they have their own potentially imperfect view about load distribution and partition boundaries (PST_N for nodes and PST_C for clients);
- Each node N_j has its own partition statistics table PST_{N_j}. An entry $PST_{N_j}[i]$ in this table ($i \neq j$), is an estimate of partition boundaries and data size related to the node N_i. The entry $PST_{N_j}[j]$ contains exact informations about partition boundaries and data size of the current node N_j. The nodes use their local partition statistics table PST_N mainly to estimate the average loads of the whole system;
- Each client C_j has its own partition statistics table PST_{C_j}. An entry $PST_{C_j}[i]$ in this table, is an estimate of partition boundaries and data size related to the node N_i. Clients use their partitions statistics tables PST_C to find targeted nodes during insertion, deletion and search operations;
- Whenever the volume of stored data exceeds a locally defined threshold, the affected node performs the load-balancing algorithm. It transfers the out-of-range data to its neighbors (left and/or right) based on the estimated average loads;
- Whenever an interaction between a node and another node or client occurs, the $\mathcal{PST}$ tables are exchanged. This makes it possible to asynchronously correct the content of these tables by the most recent contained values concerning the partitioning statistics;
- We ignore concurrency control issues and consider only the serial schedule of insertions and deletions, interleaved with the executions of the load-balancing algorithm. An architecture of our solution is presented in Figure 1.

It should be noted that the model presented above is inspired from the Scalable and Distributed Data Structures (SDDS) [34] architecture. SDDSs are a family of data structures designed for efficient In-memory data management. The basic data unit in SDDS may be either a record or an object with a unique key. Those data are organized in larger structures called buckets and usually stored in RAM. Our model and SDDS's model are both based on shared-nothing architecture [6,7].

3. Description of our Approach

Load balancing algorithms can be classified into two general categories [9]: (i) diffusion, where every node balances its load concurrently with every other partner. (ii) Dimension exchange, where every node is allowed to balance load only with one of its neighbors at a time. Our algorithm falls into the second category.

In our solution, data is range partitioned over n nodes. Clients send insert, delete and range queries. There is no central directory to keep the partition statistics (i.e. partition

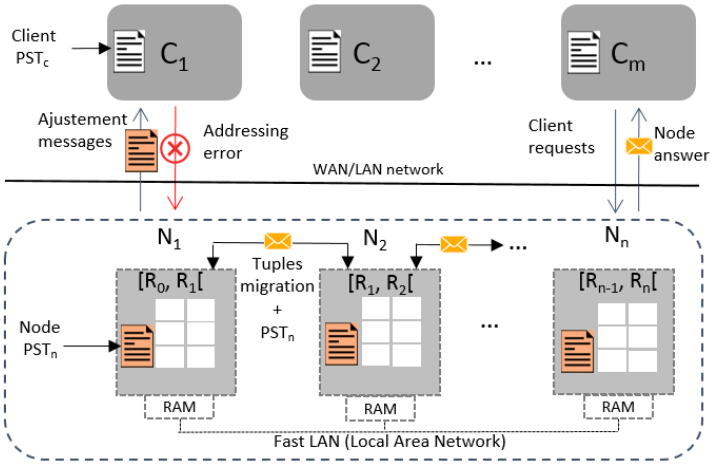

Fig. 1. $\mathcal{OLAPS}$ architecture with a set of n storage nodes and m clients, connected to each other with the Fast Local Area Network (Gigabit or InfiniBand).

boundaries and sizes). Instead, each node/client maintains a local table containing an estimate of the current partition statistics. This concept is similar to file images used in Scalable Distributed Data Structures (SDDS) [34] to avoid bottlenecks and central points of failure in a distributed environment.

The main feature of our $\mathcal{OLAPS}$ system is the $\mathcal{PST}$. Each client or node has its own and potentially imperfect partition statistics table. Clients use their PST_C to direct their insert, delete and search operations to the pretended concerned nodes. The nodes use their PST_N to estimate the balance state of the whole system and transfer the excess of data to their neighbors to achieve load balancing. These transfers, which can occur very frequently (or even continuously) necessarily produce dynamic changes in partition boundaries and sizes for the related nodes. Other nodes and clients of the system do not necessarily need to be aware of these changes. They can be accommodated with just an approximate image of the partition statistics (through their own $\mathcal{PST}$) while remaining effective in data management and manipulation.

We summarize in Table 1 the variables frequently used in this paper for easy reference.

3.1. Partition Statistics Tables ($PSTs$)

Both nodes and clients store in a local table the partition statistics. These statistics include an estimate of the number of tuples on each node ($PST[i].Load$) and the related partition boundaries ($PST[i].Lower\_Bound$ and $PST[i].Upper\_Bound$). Another field ($PST[i].Last\_Update$) is used to indicate the time when the entry $PST[i]$ was last modified. Whenever a message is sent from a node N_i to a node N_j ($i \neq j$), the partition statistics table PST_{N_i} is also included (piggybacked in the sent message), so that N_j can compare with its own statistics (using the $Last\_Update$ field) to hold the most recent value for each entry in its own table (PST_{N_j}). In this way, the most recent values,

Table 1. System Variables

Variable	Description
$PST_{N_j}[1,n]$	Partition statistics table of the node N_j.
$PST_{N_j}[i].Lower\_Bound$	The lower bound of N_i's range (from N_j's point of view).
$PST_{N_j}[i].Upper\_Bound$	The upper bound of N_i's range (from N_j's point of view).
$PST_{N_j}[i].Load$	The number of tuples stored in Ni (from N_j's point of view).
$PST_{N_j}[i].Last\_Update$	Last updating time of PST_{N_j}.
$PST_{C_j}[1,m]$	The partition statistics table of a client C_j.
$PST_{C_j}[i].Lower\_Bound$	The lower bound of N_i's range (from C_j point of view).
$PST_{C_j}[i].Upper\_Bound$	The upper bound of N_i's range (from C_j point of view).
$PST_{C_j}[i].Last\_Update$	Last updating time of the client $\mathcal{PST}$.
Av_R	The average loads of all the right nodes of N_i.
Av_L	The average loads of all the left nodes of N_i.
σ_i	The threshold of N_i.
ϕ	The imbalance ratio.
α	The degree of tolerance.
$\bar{L}$	The system average load.

regarding the boundaries of partitions and their sizes, are thus propagated in the system asynchronously.

An active node or client in the system will tend to have accurate partitioning statistics in its own $\mathcal{PST}$, and thus makes no errors when addressing a target node. On the other hand, an inactive node or client, during a certain period, may use an outdated $\mathcal{PST}$ and thus makes some errors, that necessitate additional redirections, when addressing other nodes. However, as the redirected messages also contain the $\mathcal{PST}$ tables of the targeted nodes, the partition statistics of the faulty node or client will be very quickly improved. Thus making successive addressing errors is very unlikely.

3.2. System Threshold

A node N_i attempts to shed its load whenever $PST_{N_i}[i].Load$ is greater than a local threshold σ_i. Each node has its own threshold. This value is calculated using the estimates in PST_{N_i}. Formally, we consider that for each node N_i:

– The threshold:

$$\sigma_i = \bar{L} + \delta \tag{1}$$

– The average load:

$$\bar{L} = \sum_{j=1}^{n} PST_{N_i}[j].Load/n \tag{2}$$

– The imbalance tolerance parameter:

$$\delta = (\alpha * \bar{L})/100 \tag{3}$$

where $\bar{L}$ is the system average load. It is the ratio between the total nodes load ($\sum_{j=1}^{n} PST_{N_i}[j].Load$) and the total number of nodes n. δ is called the 'imbalance

tolerance parameter'. This allows tolerating some imbalance by a parameter α (e.g., 10%, 20% or 50% of the average load). When α is low, the load balancing algorithm will often be invoked. When α is high, the load balancing algorithm invocations will be less frequent and thus clients addressing errors decrease.

It should be noted here that the value of the imbalance tolerance parameter may be fixed by the system administrator. This parameter depends on the imbalance degree that we can tolerate. In case of very sensitive applications, where we cannot tolerate a big difference between the largest and smallest load, this value must be zero or close to zero. However, when we can tolerate some imbalance and we focus on reducing the cost of data movement and the balancing algorithm invocation, the system administrator may be 50%, 70% or even a greater value. This parameter is a compromise between the imbalance of the system on one side and the addressing errors and the cost of data movement on the other side.

3.3. Client Operations

A client C_j uses its own PST_{C_j} to localize the targeted node(s) to which insert, delete and search queries are sent. 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 the key k is between its lower bound $PST_{C_j}[i].Lower\_Bound$ and its upper bound $PST_{C_j}[i].Upper\_Bound$. Then, it sends a message to N_i containing the new tuple to insert, delete or to search. The local PST_{C_j} table 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 $PST_{N_i}[i].Load$ and $PST_{N_i}[i].Last\_update$ fields in the case of an insert or delete request) and sends a positive acknowledge consisting of its PST_{N_i} to the client.
- If k is outside the node's range, an adjustment message including a negative acknowledge and the current PST_{N_i} is sent to the client.
- If a client receives a positive acknowledgment from a node, it just updates its table if it was outdated. Else, if it receives an adjustment message, it updates its table and repeats the operation until receiving a positive acknowledgment.

Range Query Execution: Range query is a well-known database operation. It returns all the data between two specified values (upper and lower boundary). A straight-forward mechanism for executing sequential range queries is scanning one partition at a time and returning results to the client during or after the scan of each partition [19]. This mechanism is relatively easy to implement. However, it does not take advantage of the system parallelism that reduces the total query execution time.

Instead, our algorithm sends parallel sub-queries to the subset of nodes supposed to be concerned by the specified range. In case of addressing error occurring for some sub-queries, the client C_j updates its PST_{C_j} and replaces the sub-queries. This method avoids the broadcast approach which generates a significant overload on all the nodes of the system. Note that bulk insertions and deletions are done in the same way as range queries.

We define $R = [a, b[$ as the range of the query. The flow of a range query is described in the following points:

1. The client uses its PST_{C_j} to split the initial range $R = [a, b[$ into S sub-ranges: $R_1 = [a, b_1[, R_2 = [b_1, b_2[, \ldots R_s = [b_{s-1}, b[$, Each sub-range will be sent to a corresponding node $N_1, N_2, \ldots N_s$.
2. Each node N_i receiving the request can behave as follows:
   - If the given sub-range R_d ($d \in [1, S]$) is completely included in its range, the node sends a positive acknowledgment, its PST_{N_i}, and all data in the specified sub-range to the client.
   - If the specified sub-range R_d is completely outside of the node range, the node answers by a negative acknowledgment (no data to send to the client) and its PST_{N_i}.
   - If only a part of the specified sub-range R_d is included in the node range, it answers with a negative acknowledgment, its PST_{N_i}, and data belonging to the part of the specified sub-range included in the node range.
3. In the client side, whenever an answer with a negative acknowledgment is received, the client PST_{C_j} is updated and the missing ranges are then regenerated and resubmitted to other targeted nodes.

Range Query Algorithm For the implementation of range query solution, a queue of submitted sub-ranges is used. The client expects a positive acknowledgment for each sub-range in the queue. The pseudo algorithm 1 describes the range query process in the client side.

Algorithm 1: RANGEQUERY (a, b)

```
1  Create queue(Q);
2  Result = { } // empty set;
3  Split R = [a, b[ into S sub-ranges according to client's PST_{C_j}:
     R_1 = [a, b_1[, R_2 = [b_1, b_2[, ... R_s = [b_{s-1}, b[;
4  //Each sub-range R_d (d ∈ [1, S]) constitutes a new range query (sub-query);
5  Enqueue the new sub-queries into Q;
6  while (Not Empty (Q)) do
7      Dequeue the first S sub-queries from Q;
8      Send each subquery to the appropriate node N_i (according to the client's PST_{C_j});
9      for (each response r) do
10         Result:= Result ∪ {data contained in r, if any};
11         Update the current PST_{C_j} if necessary;
12         if (r contains a negative acknowledgment) then
13             Split the interval contained in r, using the updated PST_{C_j};
14             Enqueue the new generated sub-queries into Q;
15         end
16     end
17     S = The total number of sub-queries generated in the for-loop;
18 end
19 Display (Result);
```

For example, consider the following range query: *Select all the tuples between the keys 500 and 2900*, over four processing nodes N_1, N_2, N_3, and N_4. The client C_j has the following information about the data distribution: N_1: $[1, 900[$, N_2: $[900, 2300[$, N_3: $[2300, 3000[$, and N_4: $[3000, 4000]$. The initial range $[500, 2900]$ is then split into 3 sub-ranges $R_1 = [500, 900[$, $R_2 = [900, 2300[$, $R_3 = [2300, 2900]$, Each sub-range is sent to the corresponding node N_1, N_2, N_3. However, the real view about the data distribution is: N_1: $[1, 900[$, N_2: $[900, 2300[$, N_3: $[2300, 2800[$, and N_4: $[2800, 4000]$. N_1 and N_2 answer the queries with a positive acknowledgment and the corresponding data set. However, N_3 answers the request with a negative acknowledgment and the corresponding data in its range ($[2300, 2800[$). The client, after updating its PST_{C_j}, it sends another range query (*Select all the data between 2800 and 2900*) to N_4 which will answers correctly the request.

The additional cost regarding this redirection is almost one message sent the node N_4. The experiments that we did show that the client may at least make one redirection to attend the right node.

This example is illustrated in Figure 2.

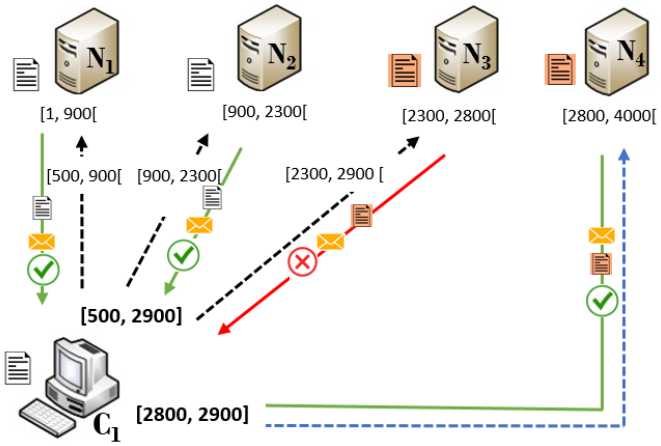

Fig. 2. Range query scenario with 4 nodes

3.4. Node Operations

Nodes use their own PST_N to estimate global average load, the left average load, and the right one. Based on these estimates, an overloaded node determines the amount of data to be transferred to each of its neighbors (left and right). The PST_N is also used to eventually correct outdated clients and neighboring nodes. A node can receive search, insert and delete operations from a client or data transfer operations from its two neighboring nodes.

The local PST_N is updated in the following cases:

- After each insert or delete operation. In that case, the local data size and the last updating time are adjusted accordingly. Entries which concern the other nodes are eventually updated using the received client's PST_C;
- After each range query sent by clients. In that case, only the older entries in the node table are eventually updated using the received client's PST_C;
- After 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 table entries are also eventually updated using the received node's PST_N.

LOADBALANCE Algorithm: As data is range partitioned over nodes, we can only move data from one node N_i to its left and/or right neighbor. Here, a neighboring concept refers to the node with the following range or the previous range. When a node N_i's load increases beyond the local threshold σ_i, it attempts to migrate data to its right and left neighbors (N_{i+1}, N_{i-1}). Algorithm 2 describes the process of correcting a state of imbalance caused by the insertion of new data sent by a client or an overloaded neighbor. The following points describe our algorithm:

- First of all, the algorithm calculates the average load of all the right neighbors (Av_R) as well as the average load of all the left neighbors (Av_L).
- The Boolean INSERT indicates whether the procedure was called after a client insert or the current node is receiving data from an over-loaded neighbor.
- If it is a post-insertion processing, the current node sends NB_R tuples to the right neighbor. NB_R value is calculated according to the expression cited in line 7 of the algorithm. It sends also NB_L tuples to the left neighbor according to the expression cited in line 6 of the algorithm. The calculation of these two values is carried out according to the information contained in the current node's PST_{N_i}.
- If it is a post-transfer processing, the current node sends all the excess data to its right neighbor if $Av\_R < Av\_L$, else, the data is sent to the left neighbor.

Each node receiving data (from a client or an overloaded neighboring node), updates its table and attempts to perform the LOADBALANCE procedure if it becomes over-loaded. Note that the local threshold may be wrong as the node table may be outdated. Despite this, the algorithm continues to run with this erroneous value without negatively influencing the quality of global balancing. Our experiments show that the PST_N converge gracefully towards the real state of the system and that the threshold also converges towards the true value.

4. Experimental Results

In this section, we present an experimental study to show the behavior of our approach in terms of load balancing quality and performance of data access operations (mainly range queries).

We ran our experiments in our laboratory on up to 10 nodes and 5 clients. Processing nodes software and clients software were executed on machines with Intel(R) Core(TM) i7-5500U CPU@2.40GHz and 8 GiB of RAM. Both nodes and clients were connected through a Gigabit Ethernet network. Algorithms are implemented in C language using the

Algorithm 2: LOADBALANCE (N_i)

if *(*$PST_{N_i}[i].Load > \sigma_i$*)* **then**
  Calculate Av_R;
  Calculate Av_L;
  $NB_R = i * (\bar{L} - Av\_R)$;
  $NB_L = (n - 1 - i) * (\bar{L} - Av\_L)$;
  $NB_L = (PST_{N_i}[i].Load - \bar{L}) * (NB_L/(NB_L + NB_R))$;
  $NB_R = (PST_{N_i}[i].Load - \bar{L}) - NB_L$;
  if *(*INSERT=*1)* **then**
    // New data insertion;
    The total number of tuples to be migrated is $NB_L + NB_R$;
    Send NB_L tuples and $N_i's$ table (PST_{N_i}) to the left neighbor;
    Send NB_R tuples and $N_i's$ table (PST_{N_i}) to the right neighbor;
  else
    //Data received from the neighbors;
    if *(*$Av\_R < Av\_L$*)* **then**
      Send $NB_L + NB_R$ tuples and $N_i's$ table (PST_{N_i}) to the right neighbor;
    else
      Send $NB_L + NB_R$ tuples and $N_i's$ table (PST_{N_i}) to the left neighbor;
    end
  end
  //Updating the node PST_{N_i};
  $nb\_tuple = PST_{N_i}[i].Load - (NB_L + NB_R)$;
  Update PST_{N_i}'s entries about N_{i+1} and N_{i-1} ;
  $PST_{N_i}[i].Load = PST_{N_i}[i].Load - nb\_tuple$;
  Update PST_{N_i}'s boundaries;
  Update PST_{N_i}'s last updating time;
else
  The system is balanced;
end

Parallel Virtual Machine (PVM) library. The generated workload involves the following types of clients:

1. *Right_Client*: a client that continuously inserts blocks of data to the right area of the system. At each block insert, tuples are generated in the assumed range of a node randomly selected from the most right nodes.
2. *Left_Client*: a client that continuously inserts blocks of data to the left area of the system. At each block insert, tuples are generated in the assumed range of a node randomly selected from the most left nodes.
3. *Alternate_Client*: a client that behaves alternately, as a *Left_Client* then as a *Right_Client*, for a given period.
4. *Random_Client*: a client that continuously inserts blocks of randomly generated data into randomly selected nodes.
5. *Range_Client*: a client that performs continuously range queries. The range boundaries are chosen randomly.

Each of these clients has its own approximated partition statistics table (PST_C). Initially the partition boundaries are all initialized by a uniform division of the domain of values according to the number of nodes in the system. Another parameter ("Client_throughput") controls the rate at which the client interacts with the system.

In addition to this, there is a main program (*Master_Client*) which allows launching nodes and other clients (depending on the chosen scenario) and periodically collects the performances measures such as the overall imbalance ratio, the number of message redirections (addressing errors) and some useful execution traces like number of data transfers between nodes.

Left, Right and Alternate clients are mainly used to produce skewed insertion patterns so that partition statistics (node's boundaries and sizes) change continuously during the simulation progress. The speed of change depends on the "Client_throughput" values for the clients.

To test our approach and validate it, we used a dataset of 1 million keys. The inserted keys are integers generated in a random way by the 5 types of clients cited above.

4.1. Evaluation Measures

The most important performance factors in this context are:

1. The imbalance ratio between the largest and smallest load.
2. The addressing errors made by clients.

The first one gives an indication of the load balancing quality.

$$\phi = maxLoad/minLoad \tag{4}$$

The *Master_Client* retrieves in regular time interval (for example every second) the largest load (*maxLoad*) and the smallest load (*minLoad*). If the ratio of these two values is close to 1 (meaning that the largest load is very close to the smallest one), the balancing of data distribution is then achieved.

The second measure indicates the number of redirection messages generated to correct out-dated clients. It is an additional cost induced by dynamic changes of the partition boundaries. In the tests we are going to perform, we show that these two measurements (1, 2) remain very low, even in the presence of very strong dynamic changes in the partition statistics.

Other measures are also considered to show the behavior details of our approach:

1. Data movement cost. All the load balancing algorithms need to move data from one node to another in order to achieve storage balance;
2. The number of invocations of LOADBALANCE algorithm.

These two measures contribute in quantifying the degree of data skew generated by our simulation tests. Indeed, we have been particularly interested in evaluating our approach in this context of a very skewed data distribution, in order to show the effective ability of the proposed approximated partitioning scheme to support such a dynamism.

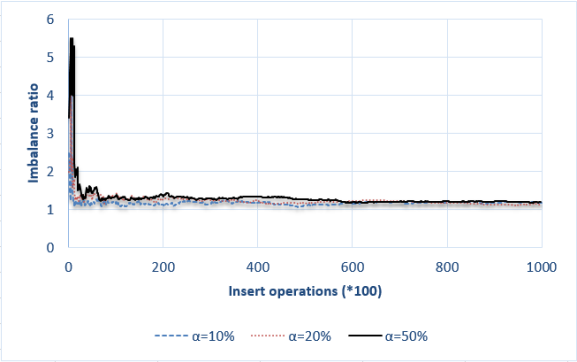

Fig. 3. Imbalance ratio with *Random_Clients*.

4.2. Performance Tests

Imbalance Ratio: In these experiments, we consider a sequence of block insert operations generated by some concurrent clients (Left, Right, Alternate and Random clients). We observe the evolution of the global imbalance ratio during these insertions.

In Figure 3, *Random_Clients* insert data uniformly over the set of nodes. In this test, there is no data skew, all nodes are uniformly addressed by the clients. We observe a very good global imbalance ratio. This confirms that our approach can be adapted to weakly dynamic systems.

In the next scenarios, we generate a high degree of data skew. For that, we use Left, Right, and Alternate clients to direct most of the insertions to only some nodes.

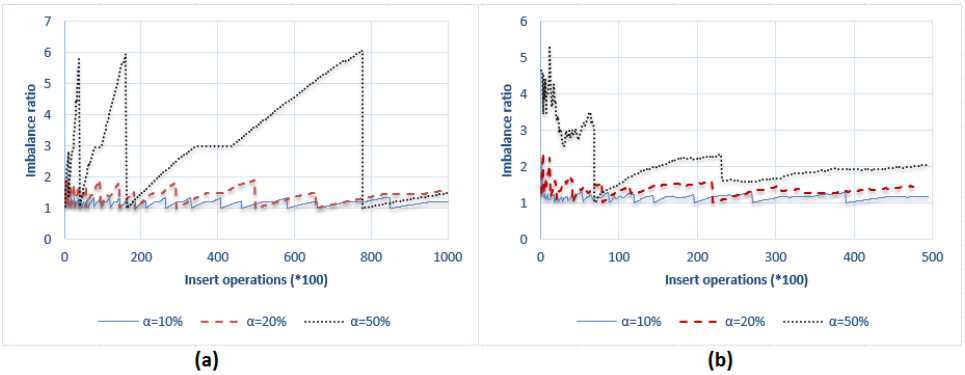

Fig. 4. Imbalance ratio with (a) a *Left_Client* and (b) both *Right_Client* and *Alter_Client* simultaneously.

Figures 4(a) and 4(b) show the imbalance ratios (Y-axis) against the number of insert operations (X-axis) during a run with a *Left_Client* and both *Right_Client* and *Alter-*

nate_Client simultaneously. The curves illustrate the imbalance ratios between the largest and the smallest load during the data insert. According to the results presented in these figures, we observe that the imbalance ratio fluctuates less and less with the amount of data inserted. In all cases, the system maintains a balancing level controlled by the imbalance tolerance parameter. As expected, the average ratio is very close to 1 in almost all cases, except when α is very large $\alpha = 50\%$ with a very high degree of data skew (4(a)).

It is noted that we have proposed in this work three values of the parameter α, however, the value of the imbalance tolerance parameter still a choice of the administrator, the latter can choose another totally different value. The choice of the α value is strongly dependent on the type of the applications. If we have applications that tolerate disequilibrium, α can be large, otherwise α should converge to zero to ensure optimal balancing.

Table 2. Comparison between our load balancing algorithm and two other ones

Procedure	Largest load (Tuples)	Mean load (Tuples)	Ratio (Largest Load/Mean Load)	Query performance
ADJUSTLOAD	1885	781	2.41	57%
GLOBALBALANCE	1090	781	1.40	28%
LOADBALANCE	812	781	1.04	-
ADJUSTLOAD	2102	1048	2.02	41%
GLOBALBALANCE	1452	1048	1.39	15%
LOADBALANCE	1244	1048	1.18	-

Simulations were run comparing performance of our LOADBALANCE algorithm and the GLOBALBALANCE procedure of [41] and the ADJUSTLOAD procedure of [18]. The data from Table 2 represents the comparative load balancing results of the three procedures. The comparison parameter is the imbalance ratio which measured here as the ratio between the largest load and the system average load.

The load balance ratios delivered by our LOADBALANCE procedure are very consistent, while there is considerable variance in the load balance ratios from the two other procedures. This relative stability ensures a greater reliability and predictability.

When the performance of the system is measured by query response time, it is proportional to the largest node load. The worst-case relative performance of the LOADBALANCE algorithm versus the ADJUSTLOAD procedure is the ratio of the highest load balance ratios obtained for the two algorithms, or 1.0-(1.04+2.41)=0.57. One can expect to reduce query response time up to 57% as compared to a system using the ADJUSTLOAD and up to 28% compared to a system using the GLOBALBALANCE.

Addressing Errors: After a balancing operation, three nodes at least change their partition boundaries and partition sizes due to inter-nodes data migration. The affected nodes have to update their tables. Clients having outdated tables can thus address wrong nodes that have changed their partition boundaries. We measure the total number of times the clients send their requests to the wrong nodes and hence made addressing errors, dur-

ing the simulation tests. Recall that when a client addresses a wrong node, it receives an adjustment message and thus sends a new query to another node.

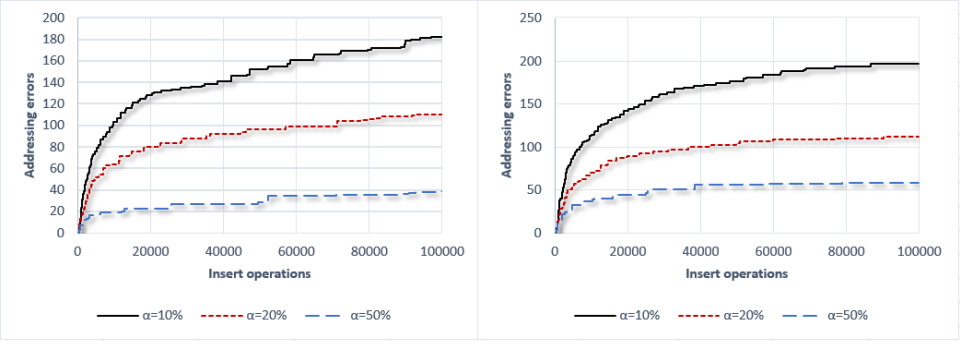

Fig. 5. Client addressing errors with a (a) *Right_Client* and a (b) *Left_Client*. The curves are obtained with the three values of α.

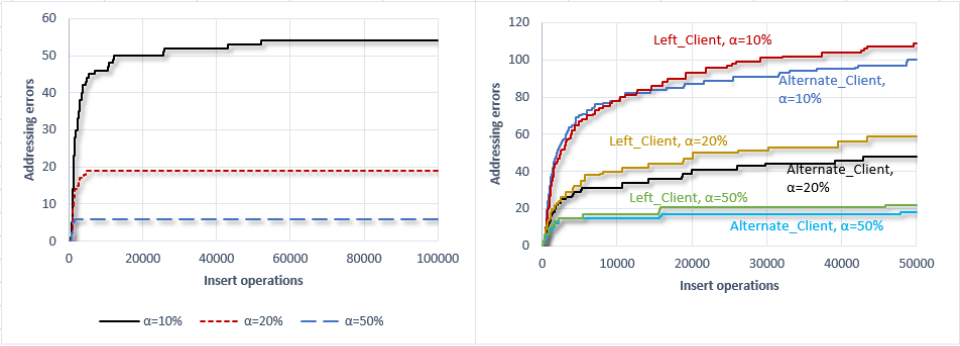

Fig. 6. Client addressing errors with a (a) *Alternate_Client*, (b) both *Left_Client* and *Alternate_Client*. The curves are obtained with the three values of α.

In Figure 5 and 6, we show the total amount of addressing errors made by the clients during the previews experiments. The results depicted in figures 5(a) and 5(b) correspond to the case of high degree of data skew. Insertions are all sent to only one area, using left or right client. The results obtained in the case of a moderate degree of data skew (a mixture of alternating and skewed insertions) are presented in Figure 6(b). For a low degree of data skew (alternating insertions between left and right areas) the measures are drawn in Figure 6(a).

In all cases and as expected, the maximum addressing error values are obtained with small imbalance tolerance parameter ($\alpha = 10\%$), whereas, the minimum values were obtained with high imbalance tolerance parameter ($\alpha = 50\%$). For instance, inserting 50 000 blocks of data using a parameter $\alpha = 10\%$, generates around 150 redirections in the case of high degree of data skew. It generates around 100 redirections in the case of moderate degree of data skew and only 50 redirections with a low degree of data skew. Thus the probabilities for a client to make an addressing error (and thus generate a redirection message) are respectively 0.003, 0.002 and 0.001 (according to the data skew degree).

According to the shape of the different curves, we also notice that addressing errors become less and less frequent over time. This is explained by the fact that the partition statistics tables (PST) tend to converge towards the real view of the partition statistics.

Addressing Errors of Range Query The next experiments are conducted to show the very low impact of our LOADBALANCE algorithm (mainly the dynamic changes in the partitioning statistics) on the performance of the range queries sent by the clients. We did not measure the queries response time because it is highly dependent on the material used while, we mainly focus on the data balancing quality.

In these experiments, we generate random range queries sent by a *Range_Client* at regular period. In parallel, a *Left_Client* or *Right_Client* insert blocks of data continuously generating thus a high degree of data skew and a rapid change in the partition statistics inside the nodes. We have therefore measured the addressing errors made by the *Range_Client*. The client PST_C, even if it is outdated at the beginning, it is quickly updated. When the nodes are very active, their tables remain of very good quality, so, they quickly correct the clients who contact them. The strong activity of the nodes is related to the fact that we generate high imbalance situations by inserting data in a non-uniform way. In that case, to maintain balancing, the nodes have to be very active. As a result, their tables converge rapidly towards the real view. It is, therefore, a strong point of our method that ultimately guarantees a few addressing errors in case of range queries (at most 2 errors) whatever the situation:

- In case of high dynamic system (presence of a moderate or large degree of data skew), the nodes will be very active and therefore they will quickly adjust the clients partition statistics.
- In a stable system (no or low data skew), the boundaries practically do not change, so the clients partition statistics tables remain accurate

Figures 7(a) and 7(b) plot the number of addressing errors generated during a run of random range queries. In Figure 7(a), a *Left_Client* is sending 10^5 data packets to the left side of the cluster in order to cause a high degree of skew. In Figure 7(b), a *Right_Client* is instead used. During both situations, a R*ange_Client* is sending its random range queries. The plots present the addressing errors made by this last client. Recall that for range queries, addressing errors are the number of additional rounds needed to send back the additional parallel sub-queries related to the missing ranges.

4.3. Behavior Details of LOADBALANCE Algorithm

In this section, we study the data movement cost incurred by the LOADBALANCE algorithm. In the experiments presented in the tables 3, 4, 5, we start from a balanced state of

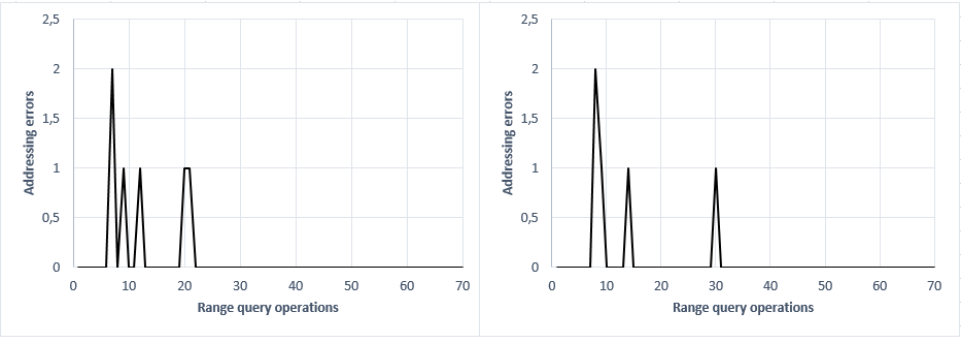

Fig. 7. Addressing errors made by a *Range_Client* with (a) *Left_Client* and (b) *Right_Client*.

the system where all the loads are close to each other. We then run a single operation of inserting a data packet (containing a block of tuples to be inserted) and observe the behavior of our system (i.e., number of tuples migrated between the nodes and the number of invocations of LOADBALANCE algorithm). The tables are obtained from three different values of the imbalance tolerance parameter, $\alpha = 10\%$, $\alpha = 20\%$ and $\alpha = 50\%$ of the average load. We note that the number of migrated tuples with $\alpha = 10\%$ is relatively small compared to the other costs. However, when α is greater, the amount of data migrated is greater because the number of LOADBALANCE invocations is smaller.

Table 3. The effect of inserting a data packet into one node during a balanced state ($\alpha = 10\%$).

Data size	67 499	125 194	131 993
Packet size	6 800	6 800	13 200
Tuples migrated	25 651	10 693	42 855
Algorithm invocations	8	3	7
Packet size/ data size	0.10	0.05	0.10
Tuples migrated/(data size + Packet size)	0.35	0.08	0.30

The aim behind these experiments is to evaluate the system reactivity (how do the system react against an insert operation which destabilizes the current balanced state). So, this involves measuring the number of tuples migrated as well as the number of algorithm invocations. What should be noted in these tables is that the number of algorithm

Table 4. The effect of inserting a data packet into one node during a balanced state ($\alpha = 20\%$).

Data size	87 497	147 590	161 243
Packet size	6 800	6 800	40 500
Tuples migrated	25 085	9 736	132 040
Algorithm invocations	6	2	7
Packet size/ data size	0.08	0.05	0.25
Tuples migrated/(data size + packet size)	0.27	0.06	0.65

Table 5. The effect of inserting a data packet into one node during a balanced state ($\alpha = 50\%$).

Data size	99 897	10 6697	119 895
Packet size	6 800	13 200	30 000
Tuples migrated	5 471	38 910	94 897
Algorithm invocations	1	4	6
Packet size/data size	0.07	0.12	0.25
Tuples migrated/(data size + packet size)	0.05	0.32	0.63

invocations gives us an estimation of the number of messages exchanged between nodes due to a single insert operation. The number of messages is the number of algorithm invocations plus 1. This measure indicates how many nodes have been affected by the insertion of a data packet in the starting node. For example, in the first column of the table 3, we have:

- Data size: 67 499, the total number of tuples before the insert query;
- Packet size: 6 800, the number of tuples that will be inserted;
- Tuples migrated: 2 565 tuples migrated after the insert query;
- Invocations: 8 is the number of LOADBALANCE invocations, thus, 9 messages are required for the transfer between nodes;
- Packet size/data size: 0.1, the ratio between the packet size and the data size before insertion;
- Tuple migrated / (data size + packet size): 0.35, The ratio between the amount of data migrated and the new data size.

A first analysis shows that the number of data transferred depends strongly on the size of the inserted packet and on the total number of data already present before insertion. In this case, for the same value of α, there is a correlation between the ratio of the packet size to the data size and the ratio of the number of tuples migrated to the sum of data size and the packet size.

In the above experiments, we measure the number of invocations of LOADBALANCE algorithm. We create four imbalance situations: 1) a strong imbalance on the right side of the cluster, where the right half of the set of nodes is much more loaded than the left one. 2) A strong imbalance on the left side. 3) An alternate send of the requests, sometimes the right half sometimes the left half of the cluster. 4) A combination of two clients, one that inserts just in the left half of the cluster and another that inserts alternately.

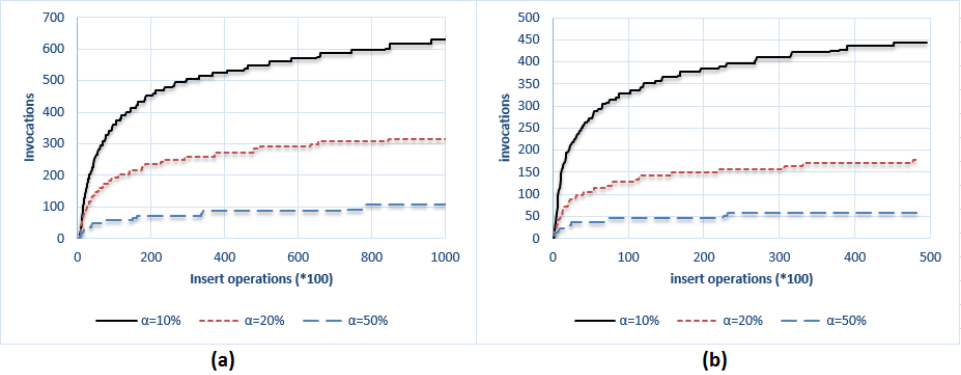

Fig. 8. LOADBALANCE Algorithm invocations during a run with (a) a *Right_Client* and (b) both *Alternate_Client* and *Left_Client*. The number of invocations of the LOADBALANCE algorithm decreases as α increases.

Figures 8(a) and 8(b) show the number of invocations of our algorithm for different degrees of α. A *Right_Client* (figure 8(a)) and both *Alternate_Client* and *Left_Client* (figure 8(b)) were sending 10^5 insert queries. We observe that the number of invocations of the LOADBALANCE algorithm decreases as α increases.

4.4. Discussion and Comparison with Existing Approaches

Our results presented above validate the analysis of the previous sections. They show that our load balancing method is very efficient and that no cost is needed for maintaining load statistics. If the system requires randomly chosen partition boundaries as in P2P network (e.g., [46,43]), efficient load balancing technique needs $O(log n)$ messages, where n is the number of nodes. Methods that use skip graphs like [18,29,11] need to have global load statistics to address the problem of data skew. Skip graphs are circular linked lists, where each node maintains roughly $O(log\ n)$ skip pointers, to enable the list traversal. Skip pointers are randomized and routing between any two nodes requires $O(log\, n)$ messages.

The team Ganesan et al. [18] propose a balancing mechanism that guarantees a good imbalance ratio (bounded by 4.24). The researchers, in this work, use two universal load balancing primitives which are neighbor item exchange and node reorder. Each node has to periodically update a central directory with its current load. When its load crosses a local threshold, it contacts the directory to locate the next least loaded node and performs load exchange with it. The major drawback is that their algorithm requires global knowledge of the maximum and minimum load. Since there is no central site in the P2P networks, Ganesan et al.'s proposal works on two skip graphs. Each node maintains $O(log n)$ skip pointers allowing it to have a fast traversal of the list. Partition changes are followed by an update of the data structure which requires $O(log\ n)$ messages. However, in our proposal, maintaining global load statistics does not need exchanging any messages.

Our contribution is based on the partition statistics tables ($\mathcal{PST}$), where each peer has imperfect load statistics. Maintaining the $\mathcal{PST}$ does not require any message exchange

between the nodes, this table is updated during the load balancing phase, it will be transmitted with the data packets sent to the neighbors.

In figures 9(a) and 9(b), we evaluate our $\mathcal{OLAPS}$ approach in a similar environment as in [18]. We generate 256 virtual machines where clients insert a set of 10^6 tuples. After achieving a balancing state, the ratios are around 1. However, in Ganesan's work, despite the imbalance ratio is bounded by 4.24, it is a little bigger than 3 in the steady phase.

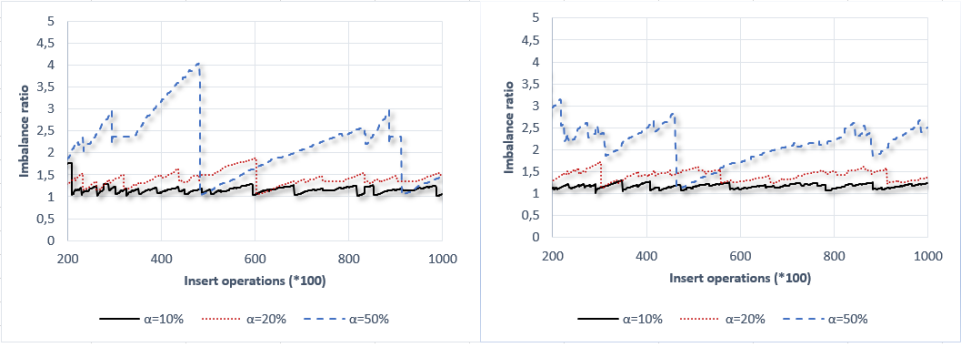

Fig. 9. Imbalance ratio with (a) *Left_Client* and (b) both *Right_Client* and *Alternate_Client*.

In Table 6, we compare our work with three other works. The comparison is based on several criteria which are mainly: perfect partition statistics, addressing errors, cost of maintaining load information and load balancing strategy (NIE: Neighbor Item Exchange or NM: Node Migration). Our algorithm uses imperfect partition statistics which consists of the partition statistics table. However, all the other works depend on a real view about the system state. To get the right system state, the cost is $O(logn)$ messages in the studied works. However, there is no need to exchange messages between peers to get the load statistics in our solution. When a client or a node needs information about the state of the system, it can easily find it in its local table, peers do not have to get this information from a central server or from neighbors. Indeed, there is need to exchange any messages. Having an imperfect view about the data distribution may lead the clients to address the wrong node. Adjustment messages help the clients to update their tables and locate the desired node. The problem of addressing errors does not exist in the three-prior works, because either the clients send their requests to a central site or they use a data structure like skip graph to get access with $O(logn)$ messages. The last comparison criterion is the load balancing strategy, our strategy consists of item migration between nodes, however, the other solution combines between this strategy and node migration from the most loaded area to the least loaded one.

5. Related Work

Research on parallel queries such us range queries [45,33,53], join queries [12,15,22], parallel sorting [3,3] ...etc, has long been a crucial problem. Reducing the data skew

Table 6. Comparison between our approach and three previous works

Works	$\mathcal{OLAPS}$	Ganesan et al.[18]	Chawachat et al.[11]	Konstantinou et al.[29]
Perfect partition statistics	No	Yes	Yes	Yes
Addressing errors	Yes	No	No	No
Cost of maintaining load statistics	No cost	$O(log\ n)$	$O(log\ n)$	$O(log\ n)$
Load balancing strategy	NIE	NIE and NM	NIE and NM	NIE and NM

effect is an interesting challange in these systems. There have been many studies of load balancing for range-partitioned data. These studies have focused first on load balancing algorithms and how to migrate data in an efficient way. Thereafter, the focus has shifted to the solutions that reduce the cost of maintaining data distribution information. The problem of data skew has been addressed in two main ranges of research:

Peer-to-Peer Systems: in P2P networks, A number of recent load balancing approaches have been proposed [17,1,11,50,37,36]. Structured P2P networks [30] 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 [42]. The basic load balancing approach in structured P2P networks or distributed hash tables (DHTs) [54] is consistent hashing [17]. 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 [23] and skip graph [2] are data structures for range searching that are not based on DHTs. Both of them are adapted from Skip Lists [39]. 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 [10], Baton [25], Chordal graphs [26]. 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 $O(logn)$ steps and also that update operations have a cost of $O(logn)$. 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 $O(logn)$ 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.

In [49], authors propose a new structured P2P network named the Well-Distribution Algorithm for an Overlay Network (Waon). Each node modifies its location on Waon's identifier ring dynamically to achieve uniform distribution of objects. It can also support range queries for searching objects. However, Dynamic load balancing mechanism of Waon is not efficient because each node uses only partial knowledge of the network. In addition, the number of communication messages necessary for maintaining the network and searching objects is O(log n), where n denotes the number of nodes. In [50]'s work, an efficient dynamic load balancing scheme for Waon is proposed. Each node collects

load values from other nodes in order to obtain entire network information. The main drawbacks of this work are the calculation time and communication data required for this process ($O(logn)$).

The work in [29] evaluates the performance in terms of bandwidth cost and convergence speed of balancing range query capable data structures using successive item exchanges and node migrations. This approach maintains load information using a skip graph. To perform node migration from the most loaded area to the least loaded one, the overloaded node locates the least loaded node by sending probing messages to its O(log n) neighbors. The main drawback of this work is the costly maintenance of the node migrations and the big number of exchanging messages to locate the overloaded node.

Another way to address the problem of data skew is the replication of some data. However, replication needs to change the underlying routing protocol to handle multiple replica locations during item search and insertions. The work in [37] proposes a load balancing technique with overlapping regions between nodes. Participant nodes share some regions with their neighbors which allow them to reshape their loads capacity function locally. The approach uses the kernel density estimation in deriving the optimal overlap ranges. The main drawback of this technique and that of [38] which uses also the data replication is the necessity to deal with consistency issues during object updates.

The use of a virtual server scheme [20] for structured P2P networks has been proposed as a dynamic load balancing technique. In a network based on such a technique, the overloaded node transfers virtual servers to another physical node for load balancing. [24] work is based on the virtual server's technique, where each participating peer is based on a partial knowledge of the system state to estimate the probability distributions of the capacities of peers and the loads of virtual servers. The principal drawbacks of these techniques are the cost of transferring virtual servers between physical nodes and the extra communication required by the virtual servers for dynamic load balancing.

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. The work in [41] proposes 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.

Work in [13] presents an approach to dealing with skew in parallel joins in database systems. The basic idea is that each processor is allocated a subrange of the join attribute value. The values that delineate the boundaries of these ranges need not be equally spaced in the join attribute domain; this allows the values to be chosen so as to equalize the number of tuples mapped to each subrange. The proposed work is interesting, many recent works are based on this work. Combining this work with our concept (approximate partition statistics tables) could significantly improve the cost of maintaining partitioning information in the processing of parallel joins.

SAP HANA is an emergent in-memory, column-oriented, relational database management system. Like any other MMDBS, this system suffers from unbalanced data distribution on main memory. The work in [32] proposes that each HANA server returns the query results to the client with its memory and CPU resources consumption status. The client when detecting that the targeted node does not have enough memory resource, it tries to send the current query to other nodes.

Balancing a large and dynamic evolution of data distribution has always been an area of interest of many researchers. Load balancing in Cassandra [40] is executed during node additions, the load balancing algorithm transfers half the data from the most loaded node to the new node. The load balancing can also be performed manually with a script executed by the user. This operation unfortunately does not minimize the amount of data transferred between the nodes and does not guarantee the data balancing. Voldemort [47] manages load balancing with the same way[3], its main disadvantages are the centralized execution generally triggered by the user. It uses a static number of partitions and distributes them uniformly on the nodes. Riak[4] uses the notion of vnodes, inspired by the virtual nodes of Chord [46]. It assigns them to the physical nodes in multiple ways. Load balancing is performed only when adding or deleting nodes. MongoDB [4] uses its cluster-balancing module, its balancing strategies is a little bit similar to ours. The balancing module migrates data between different nodes when the ratio of the number of tuples from the largest fragment to the smallest reaches a certain threshold. However, this operation lacks flexibility outside the threshold parameter.

Discussion: the above state-of-art presents the most important existing approaches dealing with the problem of data skew in a range-partitioned data system. These approaches are targeted towards peer-to-peer systems and parallel databases. The conclusion we could make is that the cost of maintaining load statistics is at least $O(logn)$. Our solution considers main memory databases and provides load balancing mechanism to handle the data skew for applications with dominant append operations and range queries. Also, these applications do not require update queries, but mainly insert, delete and search queries like in data warehouses. We provide a strategy for maintaining the global load statistics without any extra communication cost, unlike the state-of-art solutions that require at least $O(logn)$ additional messages.

6. Conclusion

In this paper, we presented $\mathcal{OLAPS}$, an on-line-load balancing algorithm for skewed data in a parallel In-memory database. By the solution we propose, we were targeting applications that require mainly insert, delete and range queries. When the range queries are operating on a range partitioned system suffering from a very high degree of imbalance, the challenge is to efficiently ensure a best performance. In order to reduce the cost of maintaining load statistics as low as possible, we propose to use an approximate partition statistics table $\mathcal{PST}$. Both clients and nodes have an imperfect view about the load distribution across nodes. Based on this table, a node performs a load balancing whenever its data size passes a local threshold. It changes its range and transfers the out-of-range data to its neighbors. The transfer process continues until the nodes satisfy the threshold. The

[3] https://github.com/voldemort/voldemort/wiki

[4] http://riak.basho.com/

experiments that we set in our laboratory, show that our solution guarantees an imbalance ratio close to 1, which means that the storage balance is achieved successfully. They show also that the access costs of the read-only clients remain very low, despite the high degree of data skew generated in these experiments. In particular, we have obtained a quasi-null degradation of parallel range query costs over a very dynamic range partitioned database where boundaries continuously change (for load balancing purposes).

While the present solution describes an approach for eliminating data skew, one could apply the same techniques to minimize execution skew. Furthermore, we also plan to address the issue of handling data skew in parallel join algorithms using the $\mathcal{PST}$ concept. Our load-balancing algorithm is a standard mechanism, we believe that it can be adapted to several other environments such as Cloud Computing environments as well as peer-to-peer networks and other systems.

References

1. Antoine, M., Pellegrino, L., Huet, F., Baude, F.: A generic API for load balancing in structured P2P systems. In: 26th IEEE International Symposium on Computer Architecture and High Performance Computing Workshop, SBAC-PAD Workshop 2014, Paris, France, October 22-24, 2014. pp. 138–143 (2014)
2. Aspnes, J., Shah, G.: Skip graphs. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA. pp. 384–393 (2003)
3. Axtmann, M., Sanders, P.: Robust massively parallel sorting. In: Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2017, Barcelona, Spain, Hotel Porta Fira, January 17-18, 2017. pp. 83–97 (2017)
4. Banker, K.: MongoDB in action. Manning Publications Co. (2011)
5. Belayadi, D., Hidouci, W.: Dynamic range partitioning with asynchronous data balancing. In: 2016 Intl IEEE Conferences on 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), Toulouse, France, July 18-21, 2016. pp. 1214–1220 (2016)
6. Bellatreche, L., Benkrid, S., Ghazal, A., Crolotte, A., Cuzzocrea, A.: Verification of partitioning and allocation techniques on teradata DBMS. In: 11th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP). pp. 158–169 (2011)
7. Bellatreche, L., Davis, T., Djahida, B.: Parallel and distributed data warehouses. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems. Springer New York (2018)
8. Bellatreche, L., Mohania, M.K.: Big data analytics and knowledge discovery. Concurrency and Computation: Practice and Experience 28(15), 3945–3947 (2016)
9. Berenbrink, P., Friedetzky, T., Hu, Z.: A new analytical method for parallel, diffusion-type load balancing. J. Parallel Distrib. Comput. 69(1), 54–61 (2009)
10. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: supporting scalable multi-attribute range queries. In: Proceedings of the ACM SIGCOMM 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 30 - September 3, 2004, Portland, Oregon, USA. pp. 353–366 (2004)
11. Chawachat, J., Fakcharoenphol, J.: A simpler load-balancing algorithm for range-partitioned data in peer-to-peer systems. Networks 66(3), 235–249 (2015)
12. Coman, A., Sander, J., Nascimento, M.A.: Adaptive processing of historical spatial range queries in peer-to-peer sensor networks. Distributed and Parallel Databases (2007)

13. DeWitt, D.J., Naughton, J.F., Schneider, D.A., Seshadri, S.: Practical skew handling in parallel joins. In: 18th International Conference on Very Large Data Bases, August 23-27, 1992, Vancouver, Canada, Proceedings. pp. 27–40 (1992), http://www.vldb.org/conf/1992/P027.PDF
14. Diaconu, C., Freedman, C., Ismert, E., Larson, P., Mittal, P., Stonecipher, R., Verma, N., Zwilling, M.: Hekaton: SQL server's memory-optimized OLTP engine. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013. pp. 1243–1254 (2013)
15. Doulkeridis, C., Vlachou, A., Kotidis, Y., Vazirgiannis, M.: Efficient range query processing in metric spaces over highly distributed data. Distributed and Parallel Databases (2009)
16. Färber, F., Cha, S.K., Primsch, J., Bornhövd, C., Sigg, S., Lehner, W.: SAP HANA database: data management for modern business applications. SIGMOD Record 40(4), 45–51 (2011)
17. Felber, P., Kropf, P., Schiller, E., Serbu, S.: Survey on load balancing in peer-to-peer distributed hash tables. IEEE Communications Surveys and Tutorials 16(1), 473–492 (2014)
18. Ganesan, P., Bawa, M., Garcia-Molina, H.: 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, Toronto, Canada, August 31 - September 3 2004. pp. 444–455 (2004)
19. Gao, J., Steenkiste, P.: An adaptive protocol for efficient support of range queries in dht-based systems. In: 12th IEEE International Conference on Network Protocols (ICNP 2004), 5-8 October 2004, Berlin, Germany. pp. 239–250 (2004)
20. Godfrey, B., Lakshminarayanan, K., Surana, S., Karp, R.M., Stoica, I.: Load balancing in dynamic structured P2P systems. In: Proceedings IEEE INFOCOM 2004, The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, March 7-11, 2004. pp. 2253–2262 (2004)
21. Gupta, A.W.: Efficient query processing using histograms in a columnar database (Jan 25 2018), uS Patent App. 15/706,511
22. Gupta, H., Chawda, B., Negi, S., Faruquie, T.A., Subramaniam, L.V., Mohania, M.K.: Processing multi-way spatial joins on map-reduce. In: Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013. pp. 113–124 (2013)
23. Harvey, N.J.A., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: Skipnet: A scalable overlay network with practical locality properties. In: 4th USENIX Symposium on Internet Technologies and Systems, USITS'03, Seattle, Washington, USA, March 26-28, 2003 (2003)
24. Hsiao, H., Liao, H., Chen, S., Huang, K.: Load balance with imperfect information in structured peer-to-peer systems. IEEE Trans. Parallel Distrib. Syst. 22(4), 634–649 (2011)
25. Jagadish, H.V., Ooi, B.C., Vu, Q.H.: BATON: A balanced tree structure for peer-to-peer networks. In: Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005. pp. 661–672 (2005)
26. Joung, Y.: Approaching neighbor proximity and load balance for range query in P2P networks. Computer Networks 52(7), 1451–1472 (2008)
27. Kallman, R., Kimura, H., Natkins, J., Pavlo, A., Rasin, A., Zdonik, S.B., Jones, E.P.C., Madden, S., Stonebraker, M., Zhang, Y., Hugg, J., Abadi, D.J.: H-store: a high-performance, distributed main memory transaction processing system. PVLDB 1(2), 1496–1499 (2008)
28. Kemper, A., Neumann, T.: Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In: Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany. pp. 195–206 (2011)
29. Konstantinou, I., Tsoumakos, D., Koziris, N.: Fast and cost-effective online load-balancing in distributed range-queriable systems. IEEE Trans. Parallel Distrib. Syst. 22(8), 1350–1364 (2011)
30. Korzun, D., Gurtov, A.: Structured peer-to-peer systems: fundamentals of hierarchical organization, routing, scaling, and security. Springer Science & Business Media (2012)

31. Koudas, N., Muthukrishnan, S., Srivastava, D.: Optimal histograms for hierarchical range queries. In: Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA. pp. 196–204 (2000)
32. Lee, J., Kwon, Y.S., Färber, F., Muehle, M., Lee, C., Bensberg, C., Lee, J., Lee, A.H., Lehner, W.: SAP HANA distributed in-memory database system: Transaction, session, and metadata management. In: 29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013. pp. 1165–1173 (2013)
33. Lim, J., Bok, K., Yoo, J.: An efficient continuous range query processing scheme in mobile p2p networks. The Journal of Supercomputing pp. 1–15 (2017)
34. Litwin, W., Neimat, M., Schneider, D.A.: Rp*: A family of order preserving scalable distributed data structures. In: VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile. pp. 342–353 (1994)
35. Luo, J., Pan, Q., He, Z.: VANET middleware for service sharing based on OSGI. Comput. Sci. Inf. Syst. 12(2), 729–742 (2015)
36. Mirrezaei, S.I., Shahparian, J.: Data load balancing in heterogeneous dynamic networks. CoRR abs/1602.04536 (2016)
37. Mizutani, K., Inoue, T., Mano, T., Akashi, O., Matsuura, S., Fujikawa, K.: Stable load balancing with overlapping id-space management in range-based structured overlay networks. Information and Media Technologies 11, 1–10 (2016)
38. Pitoura, T., Ntarmos, N., Triantafillou, P.: Saturn: Range queries, load balancing and fault tolerance in DHT data systems. IEEE Trans. Knowl. Data Eng. 24(7), 1313–1327 (2012)
39. Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. Commun. ACM 33(6), 668–676 (1990)
40. Redmond, E., Wilson, J.R.: Seven databases in seven weeks: a guide to modern databases and the NoSQL movement. Pragmatic Bookshelf (2012)
41. Rishel, W.S., Rishel, R.B., Taylor, D.A.: Load balancing in parallel database systems using multi-reordering (Sep 30 2014), uS Patent 8,849,749
42. Risson, J., Moors, T.: Survey of research towards robust peer-to-peer networks: Search methods. Computer Networks 50(17), 3485–3521 (2006)
43. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001, Proceedings. pp. 329–350 (2001)
44. Silberschatz, A., Korth, H.F., Sudarshan, S.: DatabaseSystem Concepts (2010)
45. Sioutas, S., Triantafillou, P., Papaloukopoulos, G., Sakkopoulos, E., Tsichlas, K., Manolopoulos, Y.: ART: sub-logarithmic decentralized range query processing with probabilistic guarantees. Distributed and Parallel Databases (2013)
46. Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM. pp. 149–160 (2001)
47. Sumbaly, R., Kreps, J., Gao, L., Feinberg, A., Soman, C., Shah, S.: Serving large-scale batch computed data with project voldemort. In: Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012. p. 18 (2012)
48. Sun, J., Afnan, O., Lin, Y.: Data skew finding and analysis (Jun 30 2016), uS Patent App. 15/199,507
49. Takeda, A., Oide, T., Takahashi, A.: Simple dynamic load balancing mechanism for structured P2P network and its evaluation. IJGUC 3(2/3), 126–135 (2012)
50. Takeda, A., Oide, T., Takahashi, A., Suganuma, T.: Efficient dynamic load balancing for structured P2P network. In: 18th International Conference on Network-Based Information Systems, NBis 2015, Taipei, Taiwan, September 2-4, 2015. pp. 432–437 (2015)
51. Valduriez, P.: Shared-nothing architecture. In: Encyclopedia of Database Systems, pp. 2638–2639 (2009)

52. Wang, F.: Parallel control and management for intelligent transportation systems: Concepts, architectures, and applications. IEEE Trans. Intelligent Transportation Systems 11(3), 630–638 (2010)
53. Zeng, X., Hu, M., Yu, N., Jia, X.: An efficient and secure range query scheme for encrypted data in smart grid. In: International Conference on Mobile Ad-Hoc and Sensor Networks. pp. 1–18. Springer (2017)
54. Zhang, H., Wen, Y., Xie, H., Yu, N.: Distributed Hash Table - Theory, Platforms and Applications. Springer Briefs in Computer Science, Springer (2013)

Djahida Belayadi received her bachelor's degree from Ecole Nationale Supérieure d'Informatique, Algiers, Algeria, in 2013. She is currently working toward the PhD degree in the same school. Her current research interest includes parallel databases and various distributed systems issues.

Khaled-Walid Hidouci is currently an Associate Professor at Ecole Nationale Supérieure d'Informatique (ESI) since 1993. His areas of interest mainly concern programming, distributed computing and database systems. Since 2010, he has been leading the 'Advanced Databases' (BDA) team at the Laboratory of Communication in Computer Systems.

Ladjel Bellatreche is a Professor at National Engineering School for Mechanics and Aerotechnics (ENSMA), Poitiers, where he joined as a faculty member since Sept 2010. He leads the Data and Model Engineering Team of Laboratory of Computer Science and Automatic Control for Systems (LIAS).

Received: Mart 20, 2018; Accepted: April 25, 2018.