A Distributed Tree Data Structure For
Real-Time OLAP On Cloud Architectures

F. Dehne\textsuperscript{1}, Q. Kong\textsuperscript{2}, A. Rau-Chaplin\textsuperscript{2}, H. Zaboli\textsuperscript{1}, R. Zhou\textsuperscript{1}
\textsuperscript{1} School of Computer Science, Carleton University, Ottawa, Canada
\textsuperscript{2} Faculty of Computer Science, Dalhousie University, Halifax, Canada
E-mail: frank@dehne.net, qkong@cs.carleton.ca, arc@cs.carleton.ca
zaboli@graduate.org, xiaoyunzhou@cmail.carleton.ca

Abstract—In contrast to queries for on-line transaction processing (OLTP) systems that typically access only a small portion of a database, OLAP queries may need to aggregate large portions of a database which often leads to performance issues. In this paper we introduce CR-OLAP, a Cloud based Real-time OLAP system based on a new distributed index structure for OLAP, the distributed PDCR tree, that utilizes a cloud infrastructure consisting of \((m+1)\) multi-core processors. With increasing database size, CR-OLAP dynamically increases \(m\) to maintain performance. Our distributed PDCR tree data structure supports multiple dimension hierarchies and efficient query processing on the elaborate dimension hierarchies which are so central to OLAP systems. It is particularly efficient for complex OLAP queries that need to aggregate large portions of the data warehouse, such as “report the total sales in all stores located in California and New York during the months February-May of all years”. We evaluated CR-OLAP on the Amazon EC2 cloud, using the TPC-DS benchmark data set. The tests demonstrate that CR-OLAP scales well with increasing number of processors, even for complex queries. For example, on an Amazon EC2 cloud instance with eight processors, for a TPC-DS OLAP query stream on a data warehouse with 80 million tuples where every OLAP query aggregates more than 50\% of the database, CR-OLAP achieved a query latency of 0.3 seconds which can be considered a real time response.

I. INTRODUCTION

On-line analytical processing (OLAP) systems are at the heart of many business analytics applications. This paper reports on the results of a research project (supported by the IBM Centre For Advanced Studies Canada) to investigate the use of cloud computing for high performance, real-time OLAP. By exploiting multi-dimensional views of the underlying data warehouse, OLAP queries allow users to “drill down” or “roll up” on dimension hierarchies, “slice and dice” particular attributes, or perform various statistical operations such as ranking and forecasting. In contrast to queries for on-line transaction processing (OLTP) systems which typically access only a small portion of the database (e.g. update a customer record), OLAP queries may need to aggregate large portions of the database (e.g. calculate the total sales of a certain type of items during a certain time period) which may lead to performance issues. Therefore, most of the traditional OLAP research, and most of the commercial systems, follow the static data cube approach proposed by Gray et al. [1] and materialize all or a subset of the cuboids of the data cube in order to ensure adequate query performance. However, the traditional static data cube approach has several disadvantages. The OLAP system can only be updated periodically and in batches, e.g. once every week. Hence, latest information can not be included in the decision support process. The static data cube also requires massive amounts of memory space and leads to a duplicate data repository that is separate from the on-line transaction processing (OLTP) system of the organization. Practitioners have therefore called for some time for an integrated OLAP/OLTP approach with a real-time OLAP system that gets updated instantaneously as new data arrives and always provides an up-to-date data warehouse for the decision support process (e.g. [2]). Some recent publications have begun to address this problem by providing “quasi real-time” incremental maintenance schemes and loading procedures for static data cubes (e.g. [2], [3], [4], [5]). However, these approaches are not fully real-time. A major obstacle are significant performance issues with large scale data warehouses.

The aim of our research project is to help address the above mentioned performance issues for real-time OLAP systems through the use of efficient parallel computing methods. In a recent paper [6] we presented the first parallel real-time OLAP system designed to be executed on a multi-core processor. We documented significant performance increases with increasing number of processor cores. Our system won the 2012 IBM Canada Research Impact Of The Year Award and an IBM sponsored patent application has been submitted. In this paper, we report on the next phase of our project: to scale up our real-time OLAP system to utilize a collection of \((m+1)\) multi-core processors in a cloud environment. We introduce CR-OLAP, a Cloud based Real-time OLAP system that utilizes a new distributed index structure for OLAP, refered to as a distributed PDCR tree. CR-OLAP is designed for a cloud infrastructure consisting of \(m+1\) multi-core processors where each processor executes up to \(k\) parallel threads. With increasing database size, CR-OLAP will increase \(m\) by dynamically allocating additional processors within the cloud environment and re-arranging the distributed PDCR tree. This will ensure that both, the available memory and processing capability will scale with the database size. In order to ensure high throughput and low latency even for compute intensive OLAP queries that may need to aggregate large portions of the entire database, CR-OLAP utilizes several levels of parallelism: distributed processing of multiple query and insert operations among multiple processors, and parallel processing of multiple concurrent query and insert operations within each processor. For correct query operation, CR-OLAP ensures that the result for each OLAP query includes all data inserted prior but no data inserted after the query was issued within the input stream.

We evaluated CR-OLAP on the Amazon EC2 cloud for
a multitude of scenarios (different ratios of insert and query transactions, query transactions with different sizes of results, different system loads, etc.), using the TPC-DS “Decision Support” benchmark data set. The tests demonstrate that CR-OLAP scales well with increasing number of workers. For example, for fixed data warehouse size (10,000,000 data items), when increasing the number of workers from 1 to 8, the average query throughput and latency improves by a factor 7.5. A particular strength of CR-OLAP is to efficiently answer queries with large query coverage, i.e. the portion of the database that needs to be aggregated for an OLAP query. For example, on an Amazon EC2 cloud instance with eight workers and a stream of OLAP queries on a data warehouse with 80 million tuples, where each query has more than 50% coverage. CR-OLAP achieved a query latency of 0.3 seconds, which can be considered a real time response.

CR-OLAP also handles well increasing dimensionality of the data warehouse. For tree data structures this is a critical issue as it is known e.g. for R-trees that, with increasing number of dimensions, even simple range search (no dimension hierarchies, no aggregation) can degenerate to linear search (e.g. [7]). In our experiments, we observed that increasing number of dimensions does not significantly impact the performance of CR-OLAP. Another possible disadvantage of tree data structures is that they are potentially less cache efficient than in-memory linear search which can make optimum use of streaming data between memory and processor caches. To establish a comparison baseline for CR-OLAP, we implemented STREAM-OLAP which partitions the database between multiple cloud processors based on one chosen dimension and uses parallel memory to cache streaming on the cloud processors to answer OLAP queries. We observed that the performance of CR-OLAP is similar to STREAM-OLAP for simple OLAP queries with small query coverage but that CR-OLAP vastly outperforms STREAM-OLAP for more complex queries that utilize different dimension hierarchies and have a larger query coverage (e.g. “report the total sales in all stores located in California and New York during the months February-May of all years”).

The remainder of this paper is organized as follows. In Section II we review related work. In Section III we introduce the PDCR tree data structure and in Section IV we present our CR-OLAP system for real-time OLAP on cloud architectures. Section V shows the results of an experimental evaluation of CR-OLAP on the Amazon EC2 cloud, and Section VI concludes the paper.

II. RELATED WORK

In addition to the related work discussed in the introduction, there are many efforts to store and query large data sets in cloud environments. Hadoop[8] and its file system, HDFS, are popular examples of such systems which are typically built on MapReduce[9]. Related projects most similar to our work are Hive[10] and HadoopDB[11]. However, these systems are not designed for real-time (OLTP style) operation. Instead, they use batch processing similar to [2], [3], [4], [5]. The situation is similar for BigTable[12], BigQuery[13], and Dremel[14]. In fact, Dremel[14] uses a columnar data representation scheme and is designed to provide data warehousing and querying support for read-only data. To overcome the batch processing in Hadoop based systems, Storm [15] introduced a distributed computing model that processes in-flight Twitter data.

However, Storm assumes small, compact Twitter style data packets that can quickly migrate between different computing resources. This is not possible for large data warehouses. For peer-to-peer networks, related work includes distributed methods for querying concept hierarchies such as [16], [17], [18], [19]. However, none of these methods provides real-time OLAP functionality. There are various publications on distributed B-trees for cloud platforms such as [20]. However, these method only supports 1-dimensional indices which are insufficient for OLAP queries. There have been efforts to build distributed multi-dimensional indices on Cloud platforms based on R-trees or related multi-dimensional tree structures, such as [21], [22], [23]. However, these method do not support dimension hierarchies which are essential for OLAP queries.

III. PDCR TREES

Consider a data warehouse with the fact table $F$ and a set of $d$ dimensions $\{D_1, D_2, ..., D_d\}$ where each dimension $D_i, 1 \leq i \leq d$ has a hierarchy $H_i$ including hierarchical attributes corresponding to the levels of the hierarchy. The hierarchical attributes in the hierarchy of dimension $i$ are organized as an ordered set $H_i$ of parent-child relationships in the hierarchy levels $H_i = \{H_1, H_2, ..., H_\ell\}$ where a parent logically summarizes and includes its children.

The sequential DC tree introduced by Kriegel et al. [24] exploits knowledge about the structure of individual dimension hierarchies both for compact data representation and accelerated OLAP query processing. In our previous work [6], we introduced the PDC tree, a parallel DC tree for a single multi-core processor. We note that, our PDCR tree data structure is not just another distributed R-tree, but rather a multi-dimensional data structure designed specifically to support efficient OLAP query processing on the elaborate dimension hierarchies that are central to OLAP systems. We now outline a modified PDC tree, termed PDCR tree, which will become the building block for our CR-OLAP system. For a cloud architecture with multiple processors, each processor will store one or more PDCR trees. Our CR-OLAP system outlined in the following Section IV requires that a sub-tree of a PDCR tree can be split off and transferred to another processor. This required us to (a) devise an array based tree implementation that can be packed into a message to be sent between processors and (b) a careful encoding of data values, using compact IDs related to the different dimension hierarchy levels. Due to space limitations, we refer to the full version of this paper [25] for more details.

IV. CR-OLAP: CLOUD BASED REAL-TIME OLAP

CR-OLAP utilizes a cloud infrastructure consisting of $m+1$ multi-core processors where each processor executes up to $k$ parallel threads. One of the $m+1$ multi-core processors is referred to as the master, and the remaining $m$ processors are called workers. The master receives from the users the input stream of OLAP insert and query operations, and reports the results back to the users (in the form of references to memory locations where the workers have deposited the query results). In order to ensure high throughput and low latency even for compute intensive OLAP queries that may need to aggregate large portions of the entire database, CR-OLAP utilizes several levels of parallelism: distributed processing of multiple query and insert operations among multiple workers,
and parallel processing of multiple concurrent query and insert operations within each worker. With increasing database size, CR-OLAP will increase \( m \) by dynamically allocating additional processors within the cloud environment and re-arranging the distributed PDCR tree. This will ensure that both, the available memory and processing capability will scale with the database size.

We start by outlining the structure of a distributed PDCR tree and PDCR tree on \( m + 1 \) multi-core processors in a cloud environment. Consider a single PDCR tree \( T \) storing the entire database. For a tunable depth parameter \( h \), we refer to the top \( h \) levels of \( T \) as the hat and we refer to the remaining trees rooted at the leaves of the hat as the subtrees \( s_1, \ldots, s_n \). Level \( h \) is referred to as the cut level. The hat will be stored at the master and the subtrees \( s_1, \ldots, s_n \) will be stored at the \( m \) workers. We assume \( n \geq m \) and that each worker stores one or more subtrees.

CR-OLAP starts with an empty database and one master processor (i.e. \( m = 0 \)) storing an empty hat (PDCR tree). Note that, DC trees [24], PDC trees [6] and PDCR trees are leaf oriented. All data is stored in leaves called data nodes. Internal nodes are called directory nodes and contain arrays with routing information and aggregate values. Directory nodes have a high capacity and fan-out of typically 10 - 20. As insert operations are sent to CR-OLAP, the size and height of the hat (PDCR tree) grows. When directory nodes of the hat reach height \( h \), their children become roots at subtrees stored at new worker nodes that are allocated through the cloud environment.

For a typical database size, the hat will usually contain only directory nodes and all data will be stored in the subtrees \( s_1, \ldots, s_n \). After the initial set of data insertions, all leaf nodes in the hat will usually be directory nodes of height \( h \), and the roots of subtrees in workers will typically be directory nodes as well. Both insert and query operations are executed concurrently.

**Concurrent insert and query operations**

Each query operation in the input stream is handed to the master which traverses the hat. Note that, at each directory node the query can generate multiple parallel threads, depending on how many child nodes have a non-empty intersection with the query. Eventually, each query will access a subset of the hat’s leaves, and then the query will be transferred to the workers storing the subtrees rooted at those leaves. Each of those workers will then in parallel execute the query on the respective subtrees, possibly generating more parallel threads within each subtree.

For each insert operation in the input stream, the master will search the hat, arriving at one of the leaf nodes, and then forward the insert operation to the worker storing the subtree rooted at that leaf. Figures 1 and 2 illustrate how new workers and new subtrees are added as more data items get inserted. Figures 1 illustrates insertions creating an overflow at node \( A \), resulting in a horizontal split at \( A \) into \( A_1 \) and \( A_2 \) plus a new parent node \( C \). Capacity overflow at \( C \) then triggers a vertical split illustrated in 2. This creates two subtrees in two different workers. As outlined in more details in the CR-OLAP "migration strategies" outlined below, new workers are requested from the cloud environment when either new subtrees are created or when subtree sizes exceed the memory of their host workers. Workers usually store multiple subtrees. However, CR-OLAP randomly shuffles subtrees among workers. This ensures that query operations accessing a contiguous range of leaf nodes in the hat create a distributed workload among workers.

![Fig. 1. Insertions triggering creation of new workers and subtrees. Part 1. (a) Current hat configuration. (b) Insertions create overflow at node \( A \) and horizontal split.](image)

![Fig. 2. Insertions triggering creation of new workers and subtrees. Part 2. (a) Same as Figure 1b with critical subtrees highlighted. (b) Insertions create overflow at node \( C \) and vertical split, triggering the creation of two subtrees in two different workers.](image)

For correct real time processing of an input stream of mixed insert and query operations, CR-OLAP needs to ensure that the result for each OLAP query includes all data inserted prior but no data inserted after the query was issued within the input stream. We will now discuss how this is achieved in a distributed cloud based system where we have a collection of subtrees in different workers, each of which is processing multiple concurrent insert and query threads. In our previous work [6] we presented a method to ensure correct query processing for a single PDC tree on a multi-core processor, where multiple insert and query operations are processed concurrently. The PDC tree maintains for each data or directory item a time stamp indicating its most recent update, plus it maintains for all nodes of the same height a left-to-right linked list of all siblings. Furthermore, each query thread maintains a stack of ancestors of the current node under consideration, together with the time stamps of those items. We refer to [6] for more details. The PDCR tree presented in this paper inherits this mechanism for each of its subtrees. In fact, the above mentioned sibling links are shown as horizontal links in Figures 1 and 2. With the PDCR tree being a collection of subtrees, if we were to maintain sibling links between subtrees to build linked list of siblings across all subtrees then
we would ensure correct query operation in the same way
as for the PDC tree [6]. However, since different subtrees of
a PDCR tree typically reside on different workers, a PDCR
tree only maintains sibling links inside subtrees but it does
not maintain sibling links between different subtrees. The full
proof that correct real time processing of mixed insert and
query operations is still maintained can be found in [25]
(omitted here due to space limitations).

Load balancing

CR-OLAP is executed on a cloud platform with \((m + 1)\)
processors \((m\) workers and one master). As discussed earlier,
CR-OLAP uses the cloud’s elasticity to increase \(m\) as the
number of data items increases. We now discuss in more
detail CR-OLAP’s mechanisms for worker allocation and load
balancing in the cloud. The insert operations discussed above
create independent subtrees for each height \(h\) leaf of the hat.
Since internal (directory) nodes have a high degree (typically
10 - 20), a relatively small height of the hat typically
leads to thousands of height \(h\) leaves and associated subtrees
\(s_1, \ldots, s_n\). The master processor keeps track of the subtree
locations and allocation of new workers, and it makes sure
that a relatively high \(n/m\) ratio is maintained.

As indicated above, CR-OLAP shuffles these \(n >> m\)
subtrees among the \(m\) workers. This ensures that threads of
query operations are evenly distributed over the workers.
Furthermore, CR-OLAP performs load balancing among the
workers to ensure both, balanced workload and memory utili-
ization. The master processor keeps track of the current sizes
and number of active threads for all subtrees. For each worker,
its memory utilization and workload are the total number
of threads of its subtrees and the total size if its subtrees,
respectively.

If a worker \(w\) has a memory utilization above a certain
threshold (e.g. 75% of its total memory), then the master
processor determines the worker \(w'\) with the lowest memory
utilization and checks whether it is possible to store an addi-
tional subtree from \(w\) while staying well below it’s memory
threshold (e.g. 50% of its total memory). If that is not possible,
a new worker \(w'\) is allocated within the cloud environment.
Then, a subtree from \(w\) is compressed and sent from \(w\) to \(w'\)
via message passing. As discussed earlier, PDCR trees are
implemented in array format and using only array indices as
pointers. This enables fast compression and decompression
of subtrees and greatly facilitates subtree migration between
workers. Similarly, if a worker \(w\) has a workload utilization
that is a certain percentage above the average workload of the
\(m\) workers and is close to the maximum workload threshold
for a single worker, then the master processor determines a worker
\(w'\) with the lowest workload and well below its maximum
workload threshold. If that is not possible, a new worker \(w'\)
is allocated within the cloud environment. Then, the master
processor initiates the migration of one or more subtrees from
\(w\) (and possibly other workers) to \(w'\).

V. EXPERIMENTAL EVALUATION ON AMAZON EC2

Software

CR-OLAP was implemented in C++, using the g++ com-
piler, OpenMP for multi-threading, and ZeroMQ [26] for
message passing between processors. Instead of the usual MPI
message passing library we chose ZeroMQ because it better
supports cloud elasticity and the addition of new processors
during runtime. CR-OLAP has various tunable parameters. For
our experiments we set the depth \(h\) of the hat to \(h = 3\), the
directory node capacity \(c\) to \(c = 10\) for the hat and \(c = 15\)
for the subtrees, and the number \(k\) of threads per worker to
\(k = 16\).

Hardware/OS

CR-OLAP was executed on the Amazon EC2 cloud. For
the master processor we used an Amazon EC2 m2.4xlarge
instance: “High-Memory Quadruple Extra Large” with 8 vir-
tual cores (64-bit architecture, 3.25 ECUs per core) rated at
26 compute units and with 68.4 GiB memory. For the worker
processors we used Amazon EC2 m3.2xlarge instances: “M3
Double Extra Large” with 8 virtual cores (64-bit architecture,
3.25 ECUs per core) rated at 26 compute units and with 30
GiB memory. The OS image used was the standard Amazon
CentOS (Linux) AMI.

Comparison baseline: STREAM-OLAP

As outlined in Section II, there is no comparison system for
CR-OLAP that provides cloud based OLAP with full real time
capability and support for dimension hierarchies. To establish
a comparison baseline for CR-OLAP, we therefore designed
and implemented a STREAM-OLAP method which partitions
the database between multiple cloud processors based on
zero chosen dimension and uses parallel memory to cache
streaming on the cloud processors to answer OLAP queries.
More precisely, STREAM-OLAP builds a 1-dimensional index
on one ordered dimension \(d_{stream}\) and partitions the data into
approx. 100 \(\times m\) arrays. The arrays are randomly shuffled
between the \(m\) workers. The master processor maintains the
1-dimensional index. Each array represents a segment of the
\(d_{stream}\) dimension and is accessed via the 1-dimensional
index. The arrays themselves are unsorted, and insert opera-
tions simply append the new item to the respective array. For query
operations, the master determines via the 1-dimensional index
which arrays are relevant. The workers then search those arrays
via linear search, using memory to cache streaming.

The comparison between CR-OLAP (using PDCR trees)
and STREAM-OLAP (using a 1-dimensional index and mem-
ory to cache streaming) is designed to examine the tradeoff
between a sophisticated data structure which needs fewer
data accesses but is less cache efficient and a brute force
method which accesses much more data but optimizes cache
performance.

Test data

For our experimental evaluation of CR-OLAP and
STREAM-OLAP we used the standard TPC-DS “Decision
Support” benchmark for OLAP systems [27]. We selected
“Store Sales”, the largest fact table available in TPC-DS.
It has eight dimensions: Store, Item, Address, and Promotion
are unordered dimensions, while dimensions Customer, Date,
Household and Time are ordered. TPC-DS provides a stream of
insert and query operations on “Store Sales” which was used
as input for CR-OLAP and STREAM-OLAP. For experiments
where we were interested in the impact of query coverage
(the portion of the database that needs to be aggregated for an
OLAP query), we selected sub-sequences of TPC-DS queries with the chosen coverages.

**Test results: impact of the number of workers (m) for fixed database size (N)**

We tested how the time of insert and query operations for CR-OLAP and STREAM-OLAP changes for fixed database size (N) as we increase the number of workers (m). Using a variable number of workers 1 ≤ m ≤ 8, we first inserted 40 million items (with d=8 dimensions) from the TPC-DS benchmark into CR-OLAP and STREAM-OLAP, and then we executed 1,000 (insert or query) operations on CR-OLAP and STREAM-OLAP. Since workers are virtual processors in the Amazon EC2 cloud, there is always some performance fluctuation because of the virtualization. We found that the total (or average) of 1,000 insert or query operations is a sufficiently stable measure. The results of our experiments are shown in Figures 3, 4, and 5.

Figure 3 shows the time for 1,000 insertions in CR-OLAP (PDCR-tree) and STREAM-OLAP (1D-index) as a function of the number of workers (m). As expected, insertion times in STREAM-OLAP are lower than in CR-OLAP because STREAM-OLAP simply appends the new item in the respective array while CR-OLAP has to perform tree insertions with possible directory node splits and other overheads. However, STREAM-OLAP shows no speedup with increasing number of workers (because only one worker performs the array append operation) whereas CR-OLAP shows a significant speedup (because the distributed PDCR tree makes use of the multiple workers). It is important to note that insertion times are not visible to the users because they do not create any user response. What is important to the user are the response times for OLAP queries. Figure 4 shows the time for 1,000 OLAP queries in CR-OLAP and STREAM-OLAP as a function of the number of workers (m). Figure 5 shows the speedup measured for the same data. We selected OLAP queries with 10%, 60% and 95% query coverage, which refers to the percentage of the entire range of values for each dimension that is covered by a given OLAP query. The selected OLAP queries therefore aggregate a small, medium and large portion of the database, resulting in very different workloads. We observe in Figure 4 that CR-OLAP significantly outperforms STREAM-OLAP with respect to query time. The difference in performance is particularly pronounced for queries with small or large coverages. For the former, the tree data structure shows close to logarithmic performance and for the latter, the tree can compose the result by adding the aggregate values stored at a few roots of large subtrees. The worst case scenario for CR-OLAP are queries with medium coverage around 60% where the tree performance is proportional to \(N^{1-\frac{d}{m}}\). However, even in this worst case scenario, CR-OLAP outperforms STREAM-OLAP. Figure 5 indicates that both systems show a close to linear speedup with increasing number of workers, however for CR-OLAP that speedup occurs for much smaller absolute query times.

**Test results: impact of growing system size (N & m combined)**

In an elastic cloud environment, CR-OLAP and STREAM-OLAP increase the number of workers (m) as the database size (N) increases. The impact on the performance of insert and query operations is shown in Figures 6 and 7, respectively. With growing system size, the time for insert operations in CR-OLAP (PDCR-tree) approaches the time for STREAM-OLAP (1D-index). More importantly however, the time for query operations in CR-OLAP again outperforms the time for STREAM-OLAP by a significant margin, as shown in Figure 7. Also, it is very interesting that the for both systems, the query performance remains essentially unchanged with increasing database size and number of workers. This is obvious for STREAM-OLAP where the size of arrays to be searched simply remains constant but it is an important observation for CR-OLAP. Figure 7 indicates that the overhead incurred by CR-OLAP’s load balancing mechanism (which grows with increasing m) is balanced out by the performance gained through more parallelism. CR-OLAP appears to scale up without affecting the performance of individual queries.

**Test results: impact of the number of dimensions**

It is well known that tree based search methods can become problematic when the number of dimensions in the database increases. In Figures 8 and 9 we show the impact of increasing d on the performance of insert and query operations in CR-OLAP (PDCR-tree) and STREAM-OLAP (1D-index) for fixed database size \(N = 40\) million and \(m = 8\) workers. Figure 8 shows some increase in insert time for CR-OLAP because the PDCR tree insertion inherits from the PDC tree a directory node split operation with an optimization phase that is highly sensitive to the number of dimensions. However, the result of the tree optimization is improved query performance in higher dimensions. As shown in Figure 9, the more important time for OLAP query operations grows only slowly as the number of dimensions increases. This is obviously for the array search in STREAM-OLAP but for the tree search in CR-OLAP this is an important observation.

**Test results: impact of query coverages**

Figures 10, 11, 12, and 13 show the impact of query coverage on query performance in CR-OLAP (PDCR-tree) and STREAM-OLAP (1D-index). For fixed database size \(N = 40\)Mil, number of workers \(m = 8\), and number of dimensions \(d = 8\), we vary the query coverage and observe the query times. In addition we observe the impact of a “*” in one of the query dimensions. Figures 10 and 11 show that the “*” values do not have a significant impact for CR-OLAP. As discussed earlier, CR-OLAP is most efficient for small and very large query coverage, with maximum query time somewhere in the mid range. (In this case, the maximum point is shifted away from the typical 60% because of the “*” values.) Figures 12 and 13 show the performance of STREAM-OLAP as compared to CR-OLAP (ratio of query times). It shows that CR-OLAP consistently outperforms STREAM-OLAP by a factor between 5 and 20.

**Test results: query time comparison for selected query patterns at different hierarchy levels**

Figure 14 shows a query time comparison between CR-OLAP (PDCR-tree) and STREAM-OLAP (1D-index) for selected query patterns. For fixed database size \(N = 40\)Mil, number of workers \(m = 8\) and \(d = 8\) dimensions, we test for dimension Date the impact of value “*” for different hierarchy levels. CR-OLAP is designed for OLAP queries such as “total sales in the stores located in California and New York during
February-May of all years’ which act at different levels of multiple dimension hierarchies. For this test, we created 7 combinations of “-,” and set values for hierarchy levels Year, Month, and Day: ‘-’, ‘-’, ‘-’, ‘-’, ‘-’, ‘-’, ‘-’. We then selected for each combination queries with the same hierarchy levels and different levels of the hierarchy. The test results are summarized in Figure 14. The main observation is that CR-OLAP consistently outperforms STREAM-OLAP even for complex and very broad queries that one would expect could be easier solved through data streaming than through tree search.

VI. CONCLUSION

We introduced CR-OLAP, a Cloud based Real-time OLAP system based on a distributed PDCR tree, a new parallel and distributed index structure for OLAP, and evaluated CR-OLAP on the Amazon EC2 cloud for a multitude of scenarios. The tests demonstrate that CR-OLAP scales well with increasing database size and increasing number of cloud processors. CR-OLAP has the potential to enable OLAP systems with real-time OLAP query processing for large databases.

ACKNOWLEDGMENT

The authors would like to acknowledge financial support from the IBM Centre for Advanced Studies Canada and the Natural Science and Engineering Research Council of Canada. We thank the research staff at the IBM Centre for Advanced Studies Canada, and in particular Stephan Jou, for their support and helpful discussions.

REFERENCES

Fig. 3. Time for 1000 insertions as a function of the number of workers. ($N = 40Mil, d = 8, 1 \leq m \leq 8$)

Fig. 4. Time for 1000 queries as a function of the number of workers. ($N = 40Mil, d = 8, 1 \leq m \leq 8$)

Fig. 5. Speedup for 1000 queries as a function of the number of workers. ($N = 40Mil, d = 8, 1 \leq m \leq 8$)

Fig. 6. Time for 1000 insertions as a function of system size: $N$ & $m$ combined. ($10Mil \leq N \leq 80Mil, d = 8, 1 \leq m \leq 8$)

Fig. 7. Time for 1000 queries as a function of system size: $N$ & $m$ combined. ($10Mil \leq N \leq 80Mil, d = 8, 1 \leq m \leq 8$)

Fig. 8. Time for 1000 insertions as a function of the number of dimensions. ($N = 40Mil, 4 \leq d \leq 8, m = 8$)

Fig. 9. Time for 1000 queries as a function of the number of dimensions. The values for “1D-index 95% coverage” are 828.6, 1166.4, 1238.5, 1419.7 and 1457.8, respectively. ($N = 40Mil, 4 \leq d \leq 8, m = 8$)

Fig. 10. Time for 1000 queries (PDCR tree) as a function of query coverages: 10% – 90%. Impact of value "*" for different dimensions. ($N = 40Mil, m = 8, d = 8$)

Fig. 11. Time for 1000 queries (PDCR tree) as a function of query coverages: 91% – 99%. Impact of value "**" for different dimensions. ($N = 40Mil, m = 8, d = 8$)

Fig. 12. Time comparison for 1000 queries (Ratio: 1D-index / PDCR tree) for query coverages 10% – 90%. Impact of value "*" for different dimensions. ($N = 40Mil, m = 8, d = 8$)

Fig. 13. Time comparison for 1000 queries (Ratio: 1D-index / PDCR tree) for query coverages 91% – 99%. Impact of value "**" for different dimensions. ($N = 40Mil, m = 8, d = 8$)

Fig. 14. Query time comparison for selected query patterns for dimension Date. Impact of value "**" for different hierarchy levels of dimension Date. ($N = 40Mil, m = 8, d = 8$).