Synchronous CR-OLAP Tool for Efficient Parallel Computing

Mani Sarma Vittapu, Ph.D.
Assistant Professor, Dept. of. ITSC
Addis Ababa Institute of Technology
Addis Ababa University, Addis, Ethiopia

Venkateswarlu Sunkari, Ph.D.
Assistant Professor,Dept. of. ITSC
Addis Ababa Institute of Technology
Addis Ababa University, Addis, Ethiopia

AtoYoseph Abate
HOD of ITSC
Addis Ababa Institute of Technology,
Addis Ababa University, Addis, Ethiopia

ABSTRACT
Real time OLAP , or RTOLAP, is the capability to quickly retrieve, aggregate, analyze and present multidimensional data for cubes whenever there are changes to the data in the relational data sources, without having to run heavy processing on the cube. A big advantage of real time OLAP is that it calculates all relevant data and provides immediate output. One of the main roles of an RTOLAP system is that data is stored directly in main memory, or in an in memory database, enabling quicker access to the data. Another factor affecting the speed of calculation is compression data is compressed, in such a way that it can be accessed must faster in its compressed form. Additionally, pre-calculated values are not stored, therefore avoiding "data explosion". In contrast to queries for online transaction processing (OLTP) system that typically access only a small portion of a database, OLAP queries may need to aggregate large portion of a database which often leads to performance issues. In this paper introduced 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) multicore processors. With increasing database size, CROLAP dynamically increases m to maintain performance. The distributed PDCR tree data structure supports multiple dimension hierarchies and efficient query processing on the sophisticated dimension hierarchies which are so central to OLAP system. It is particularly efficient for complex OLAP queries that need to aggregate large portions of the data warehouses. The static data cube approach proposed by Gray et.al. and materialize all or a subset of the cuboids of the data cube in order to ensure adequate query performance. Practitioners have called for some time for a real-time OLAP approach where the OLAP system gets updated instantaneously as new data arrives and always provides an up-to-date data warehouse for the decision support process. However, a major problem for real-time OLAP is the significant performance issues with large scale data warehouses. The main aim of our research is to address these problems through the use of efficient parallel computing methods. In this paper proposed a distributed data structure for real time OLAP. To our knowledge, the real-time OLAP system that has been parallelized and optimized for contemporary multi-core architectures allows for multiple insert and multiple query transactions to be executed in parallel and in real-time.

Keywords
RTOLAP,CROLAP,OLTP,MOLAP, PDCR, performance latency

1. INTRODUCTION
Online analytical processing (OLAP) is typically defined as the processing and analysis of shared multidimensional data. In practice, OLAP systems analyze data drawn from large, low-transaction and high-latency relational databases, such as data warehouses. The purpose of such analysis is to aggregate and organize business information into a readily accessible, easy to use multidimensional structure. OLAP systems store some or all of this aggregated information either within tables in a relational database (also known as relational OLAP, or ROLAP, storage) or in specialized data structures in multidimensional databases (also known as multidimensional OLAP, or MOLAP, storage). OLAP queries can be answered much more quickly than similar relational queries because the aggregations and computations have already been completed and the resulting derived values are readily available from a ROLAP table or MOLAP storage.Retrieving, analyzing, and aggregating large amounts of historical data can consume extensive time and resources. OLAP systems do not usually run against online transaction processing (OLTP) or other high-transaction, low-latency databases because the time and resources required can affect the performance of the relational database. Instead, OLAP systems typically run against data warehouses, which are updated relatively infrequently, to support the requirements of most commercial and financial analysis. Most OLAP systems rely on a "snapshot" approach, periodically retrieving and aggregating data for later presentation and analysis. Because OLAP systems typically rely on stored, derived values to answer queries, the aggregation process must also reasonably match the update latency of the underlying relational data source to avoid presenting overly "stale" data.Products that can perform aggregations quickly enough to provide multidimensional data from low-latency data sources have challenged this traditional view of OLAP in recent years. This functionality, which is referred to as real-time OLAP, is most often used in financial or industrial scenarios where multidimensional analysis of low-latency data is crucial to the organization's business intelligence requirements.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. 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 cannot 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. Some recent publications have begun to address this problem by providing "quasi real-time" incremental maintenance schemes and loading procedures for static data cubes). However, these approaches are not fully real- time. A major obstacle is significance performance issues with large scale data warehouses.

The remainder of this paper is organized as follows. In Section 2describes the PDCR tree data structure and in Section 3describing CR-OLAP system for real-time OLAP on cloud architectures. Section 4 shows the results of an experimental evaluation of CR-OLAP, and Section 5 concludes the paper.

2. PDCR TREES

The CR-OLAP runs on multiple nodes provided by cloud service providers. It containsseveral components including a distributed PDCR tree on m+1 nodes, a network communicator, a load balancer, a migration API and a message serialization API.The building of PDCR-tree starts on one node called master. When the tree grows bigenough, we start building the subtrees on other nodes called workers. After the initial load in ,the whole PDCR-tree is distributed on multiple nodes. The master contains the top part ofthe tree and we call it hat. Each worker contains a number of subtrees and each subtreeis the same as a small PDCR-tree. After the initial built, the system is ready to take clientrequests such as new data insertions and range queries. The master node also maintainsthe information of distributed subtrees. It uses a lookup map to record which workers thesubtrees are stored. When the master receives requests, the requests will be executed inparallel in the hat. If the master needs to dispatch the tasks to workers, it will look up itsubtree-worker map to find the workers storing the relevant subtrees. Each worker whoreceives its tasks will complete the tasks in parallel locally and return the results back tothe master. When the master node gathers all results returned by the workers, it generatesa final result for the request. The final result is sent over to the client at the end.A load balancer runs periodically and moves subtrees from workers with heavy loads toworkers with lighter loads. All instructions to trigger tasks are represented as messages (forexample,migration request, sending subtree, query request and insertion request). Eachmessage carries its own data members such as a destination worker id, a source worker idand task instructions. Sending or receiving messages between nodes is handled by a network communicator. We implemented a communicator with ZeroMQ, a high-performanceasynchronous messaging library. Every message is serialized to a string by a message serializer and pushed to a message queue. Then, the network communicator is responsible forretrieving messages from the queue and sending them to their destination nodes. Once adestination node receives a message the message de-serializer de-serializes the string andrecovers the original message. Then the worker performs the message task.

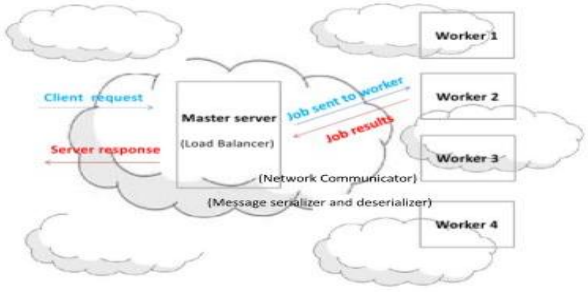


Figure 1.System Architecture

2.1 Distributed PDCR tree and it's data structure

The PDCR-tree is stored on cloud nodes. Each cloud node stores several subtrees of data. A distributed PDCR tree is a data structure containing a hat and a set of subtrees T, T={t1, t2, t3...ti}, where ti is a subtree. The hat is stored in the master node and the subtrees aredistributed between several worker nodes. A distributed PDCR-treewith a hat in the master and subtrees in several workers.Starting from the root directory node on the master, PDCR-tree is growing by insertingnew data one by one. A directory node has its node capacity, for example, maximum 15children nodes. When the number of children nodes exceeds the node capacity, a node splitwill be performed. There is two types of node split: vertical split (V-Split) and horizontal split (H-Split).An H-Split is performed when the number of children nodes of a directory node isfull, but the parent node of this directory node can still contain more children nodes. Ahorizontal link will be added between the two split nodes. The directory node capacity is 2 and the Cut level is 1. R is the root. A is the directory0node with depth 1 and contains 2 children nodes. If a new data were inserted to the nodeA, A would contain 3 nodes which exceeds its children capacity, but A's parent node R hasnot reach its node capacity yet, therefore, An H-split can be performed and split A to twonodes A and B. A has the sibling node link pointing to B. The 3 childrennodes are distributed between A and B that leads to minimal MDS overlap between A andB.If a directory node reaches its node capacity and its parent node also contains childrenwith full capacity, both a V-Split and an H-Split will be triggered. An H-split along with aV-split can make the two subtrees become independent. The directory node gets pushed 1level down in the tree. If a new data were added into node A, the numberof children nodes of A would exceed the capacity and the number of children nodes of R,the parent of A, also would exceed the capacity; hence, both the H-split and the V-split are called. The H-Split splits A to A and A0; The V-Split adds a new node C as the

parent of A and A0 and links C to the parent node R which A linked to before the split. The linkbetween A and B from previous H-Split can be removed. The subtrees C and B become independent. A and A0 is at the depth level of 2 of the hat on the master.Once the depth of directory node exceeds the Cut level, the directory node with itschildren nodes will be moved to the worker node. The depth of A and A0 exceeds the Cut level, so A and A0 with their data nodes are moved to a worker node is showing the continuously growing of the PDCR-tree. A new datais added to A on a worker and both the V-split and the H-split is executed. A gets split to A and A00, D is the result of the V-split. The link from A to A0 is removed and the subtree D and A0 is now independent. We also move one of the subtree (e.g. the subtree of A0)to another worker so the work load is spread out between workers.

2.2 Migration process and load balancer

Migration is a mechanism to reduce the load of a worker that is experiencing a high loadsuch as high memory usage or CPU usage. When the performance is slow due to the highload on a worker, a migration process is triggered to move one of subtrees to another workerhaving a lower load in the system. In the previous proof of independent subtrees, we know a subtree whose root is created from a V-split has no links with its neighbor subtrees. Theindependent subtrees can be the candidates to be moved between nodes and only parentlinks of the root of a migration subtree need to be maintained.

The load balancer runs periodically (e.g. every few minutes) to check the memoryusage on each node. Once the load balancer detects a worker having memory usage exceeda threshold, it will trigger the migration. The process choses a subtree (called migrationsubtree) and moves it from one worker (called source worker) to another worker (calleddestination worker). The migration process can also be triggered when the initial subtreeson a worker become independent. By doing that, the subtrees are distributed to multipleworkers so that the work load can be potentially spread out between workers. Building distributed PDCR-tree from master to worker the migration process sends a migration subtree to a destination worker. The migrationsubtree is a snapshot of the subtree at the time point when migration starts. To achieve this, we could choose to lock the whole subtree to perform migration. However, when a subtreeis very large, the insertions can be interrupted by the lock for a long period of time tillthe migration is completed. Instead, our method allows insertions to be still performedon the migration subtree in the source worker during a migration and all the queries that traverse the migration subtree can still include the most recent new data. Once a migrationstarted, if a directory node A needs to be updated, a copy of the node A will be storedto "Links to backup" before any update is made. The details are showed in Algorithm1 Node Backup. The node update can be any type of updates including MDSR update,measure update, node split and inserting a new data node under a directory node. If adirectory node has never been updated during a migration, the copy is not necessary to bemade. Creation of the backup for a directory node is done at the first update and is onlyperformed once since we only need one backup of a node to preserve its old structure beforethe migration.

Algorithm 1 Node Backup //Receive directory node Ptr and creates a backup of it ifneeded

1: FOR each directory node Ptr, before it gets updated (MDS update, Measure update, Split, or

Insertion) DO

2: If Creation-TS(Ptr) < TS(migration) Then

If (Link to backup(Ptr)==Null) Then

LOCK (Ptr).

Make a backup of Ptr.

Update Link to backup.

If (children of Ptr are data nodes) Then

Make a backup of them, and update their backup links.

Release the LOCK for Ptr.

When migrating the subtree from one worker to another, for those directory nodes/datanodes having no update, they are directly copied over to the destination worker; for thosenodes having been updated, their copies stored in the "Links to backup" shall be transferred to the destination worker. After the migration of a subtree is completed, in thedestination worker we apply the same insertions that were sent to the subtree in the sourceworker during the time of migration, therefore, the migrated subtree in the destination caninclude the new data coming into the system during a migration and be ready for taking

3. CR_OLAP: CLOUD BASED REAL-TIME OLAP

The CR-OLAP is a parallel real-time OLAP system designed for cloud based distributedsystems. The key component is the distributed PDCR-tree which is an extension of thePDC-tree. Using the hierarchy of dimensions, the PDCR-tree can group the data at different hierarchy levels. The data set is partitioned and distributed on multiple cloud machines so that any insertions and aggregations operations can be executed in parallel. The algorithms are given to illustrate the parallel operations. The MDS ids are modified to bitrepresentation in order to improve the system performance. A MDSR (the range of MDSids) is used for describing directory nodes so that unordered dimensions can also be expressed in an ordered manner. This helps to reduce the system response time. Meanwhile, the CR-OLAP can handle range queries at any hierarchy level as well as point queries.The distributed PDCR tree is able to handle operations like Insertions and Aggregations.An insertion operation is to add a new data into the system; An aggregation operation includes SUM, MAX, MIN, MEAN, AVG etc. For an aggregation operation, the systemreports the results to the client. When clients send any requests to the system, these operations are queued in the shared input queue on the master node. The Task Assignment(Algorithm 2) is invoked to dispatch the requests to different processes. The correspondingprocesses all start from the master node and then the tasks are distributed to the relatedworker nodes. When each worker completes its task and sends back the results, the masternode will add each worker's result together and reply it to the client.

3.1 CR-OLAP Operations and Algorithms

Algorithm 2 Task Assignment //Picks the next task in the client queue

Repeat

1: Pick the next query q from client queue

2: Case"q type"== "Insertion Query", assign a thread for q, and run "Insertion Algorithm"

3: Case "q type"== "Aggregation Query", assign a thread for q, and run "Aggregation Algorithm"

Until (there is a query in Q)

Algorithm 3 Insertion Algorithm on the master //insertion of data node d in distributed

PDC-tree X 1: Start from R (the root of the hat), and set Ptr point to R.

Repeat

2: If MDS(d) is contained in the MDSR of only one of the children of Ptr Then set Ptr equal to

the directory node for that child.

3: If MDS(d) is contained in the MDSR of more than one of the entries of Ptr Then set Ptr equal

to the root of the child sub-tree with minimum number of data nodes.

4: If MDS(d) is not contained in any MDSR of a child of Ptr Then

a. Make a copy Ptr' of node Ptr.

b. For each child C of Ptr': Add the new data item d to C and calculate the MDS enlargement

and overlap caused.

c. Set Ptr = the child which causes minimal overlap.

UNTIL Ptr is a leaf node in the hat

5: If (Ptr is pointing to a data node) Then

a. Set Ptr=parent of Ptr.

b. Acquire a LOCK for Ptr.

c. Insert data item d into Ptr and update the measure(s), MDS(Ptr), and time stamp of Ptr

i.e. TS(Ptr).

d. If capacity of Ptr is exceeded Then

1. WHILE (Ptr ! = R) && (capacity of Ptr>= Cap) set Ptr=Parent of Ptr.

2. If (Ptr==Root) Then

a. Call V Split for the parent of d, and create a new node Y.

b. If (Depth of Y == Cut) Then

1. (Counter++, assign counter to the new sub-tree)

2. select next worker using load balancing strategy

3. Send children of Y to the chosen worker to initiate a sub-tree in it.

Else Call Algorithm 5 (Split Bubbleup) Until (capacity of Ptr<=Cap)

6: If (Ptr is pointing to a Sub-tree's number) Then

a. In worker-subtree map, find the worker, w, which stores t

b. If t is migrating, send the insert query to migrating destination worker

c. Send the insert query to w

Algorithm 4 Insertion Algorithm on Worker *//insertion of data node d in sub-tree t with*

root r 1: Start from r (the root of t), and set Ptr point to r.

2: If MDS(d) is contained in the MDSR of only one of the children of Ptr Then set Ptr equal to

the directory node for that child.

3: If MDS(d) is contained in the MDSR of more than one of the entries of Ptr Then set Ptr equal

to the the root of the child sub-tree with minimum number of data nodes.

4: If MDS(d) is not contained in any MDSR of a child of Ptr Then

a. Make a copy Ptr' of node Ptr.

b. For each child C of Pt': Add the new data item d to C and calculate the MDS enlargement

and overlap caused.

c. Set Ptr = the child which causes minimal overlap.

UNTIL Ptr is a data node

5: Set Ptr=parent of Ptr.

6: If TS(migration) is set Then Call Algorithm Node Backup(Ptr)

7: Acquire a LOCK for Ptr.

8: Insert data item d into Ptr and update the measure(s), MDS(Ptr), and time stamp of Ptr i.e.

TS(Ptr).

9: If capacity of Ptr is exceeded Then

a. WHILE (Ptr ! = r) && (capacity of Ptr>= Cap) set Ptr=Parent of Ptr.

b. If (Ptr==r) Then

1. sends a request to the master to check up to which level in the hat, ancestors of Ptr

are full

2. If capacity of all directory nodes up to root in hat is full, Then Call Algorithm 6

V Split for the parent of d.

3. If capacity is not full up to root in hat, Then Call Algorithm 5 (Split Bubbleup)

UNTIL (Ptr==Parent of r (leaf of the hat))

4. Send a request to the worker queue to call Algorithm 5 (Split-bubbleup) UNTIL

(capacity of Ptr<=Cap) in the master for Ptr.

Else Call Algorithm 5 (Split Bubbleup) Until (capacity of Ptr<=Cap)

Algorithm 5 Split Bubbleup //Bubbles up the split of directory nodes starting from the

directory node Ptr Until a given condition is met.

Repeat

1: Acquire a LOCK for the parent of Ptr.

2: If capacity of Ptr is exceeded Then

a. MakePtr' the right sibling of Ptr and update the right sibling links accordingly.

b. Set the time stamp TS of Ptr' equal to the old TS value for Ptr and assign Ptr a new time

stamp TS representing the current update.

c. If TS(migration) is set Then Call Algorithm Node Backup(Parent of Ptr).

3: Insert a new link for Ptr' in the parent of Ptr.

4: Update the Measure and MDSR fields for the parent of Ptr.

5: Release the LOCK for Ptr.

6: Set Ptr = parent of PtrUNTIL(the given condition is true)

7: Release the lock on Ptr

Algorithm 6 V Split //Receives directory node Ptr, performs a H Split on Ptr, and creates

a new parent for Ptr, Ptr'

1. If TS(migration) is set Then

a. Call Algorithm 1 Node Backup(Ptr)

b. Call Algorithm 1Node Backup(Parent of Ptr)

2. Acquire a LOCK for Ptr and the parent of Ptr.

3. Split Ptr into two directory nodes Ptr and Ptr'(DC-tree split algorithm, sections 4.2 and 4.3).

4. Make Ptr' the right sibling of Ptr and update the right sibling links accordingly.

5. Set the time stamp TS of Ptr' equal to the old TS value for Ptr and assign Ptr a new time stamp

TS representing the current update.

6. Update MDSR of Ptr and Ptr', and their measures.

7. Create a new directory node D with a new TS, Add two entries for Ptr,Ptr' in D as its children,

and update MDSR(D) covers MDSR(Ptr) and MDSR(Ptr').

8. Replace the entry of Ptr in the old parent of Ptr with an entry for D.

9. Remove the Link from left sibling of Ptr to Ptr.

10. Release the LOCK for Ptr and the old parent of Ptr.

Algorithm 7 Aggregation Query Algorithm on the Master //Compute the aggregate value

for query q in partitioned PDC-tree X

1: Set Ptr=R, Push Ptr into a stack S for query q.

Repeat

2: Pop top item from stack S, call it Ptr'.

3: If the time stamp (TS) of Ptr' is smaller (earlier) than the time stamp (TS) of Ptr Then

3.1: Using the "Link to Sibling" field in directory nodes, traverse the list of siblings of Ptr.

Push all sibling nodes up to a node with its TS equal to TS of Ptr' (Push from right).

3.2: Push Ptr again into stack S.

Else

3.3: FOR each child C of Ptr DO

3.3.1: For each dimension of C where MDSR(C) and range MDS(q) are at different levels in

the dimension hierarchy, convert the lower level entry to the higher level.

3.3.2: If MDSR(C) is contained in range MDS(q) Then add Measure(C) to the result value.

3.3.3: If MDSR(C) overlaps range MDS(q) but is not contained in it, Then

If (C is pointing to a sub-tree) Then Send the Aggregation Query to the worker

Counter++

Else Push C into stack S UNTIL stack S is empty.

4. IF query is distributed to workers Then

Wait all queries are finished on workers

Calculate the aggregated results.

5. Report the aggregation value to client.

Algorithm 8 Aggregation Query Algorithm on Worker //Compute the aggregate value for

query q in sub-tree ton worker w

1: Set Ptr=r, Set result value Total=0, Push Ptr into a stack S for query q.

Repeat

2: Pop top item Ptr' from stack S.

3: If the time stamp (TS) of Ptr' is smaller (earlier) than the time stamp (TS) of Ptr Then

3.1: Using the "Link to Sibling" field in directory nodes, traverse the list of siblings of Ptr.

Push all sibling nodes up to a node with its TS equal to TS of Ptr' (Push from right).

3.2: Push Ptr again into stack S.

Else

3.3: FOR each child C of Ptr DO

3.3.1: For each dimension of C where MDSR(C) and range MDS(q) are at different

levels in the dimension hierarchy, convert the lower level entry to the higher level.

3.3.2: If MDSR(C) is contained in range MDS(q) Then

add Measure(C) to Total.

3.3.3: If MDSR(C) overlaps range MDS(q) but is not contained in it, Then

Push C into stack S UNTIL stack S is empty.

3.2 CR-OLAP query types

The PDCR tree is designed to answer a set of queries in parallel. For a fact table with d dimensions in a data warehouse, the set of queries Q is defined as {q1, q2, q3,..qd} where $1 \leq i \leq d$, and qi is a set of values to be searched in dimension i. The set of values in qi canbe represented in the following formats:

1. Multiple ranges of values covering a contiguous range of values in a dimension i.This type of query value is used only for ordered dimensions such as date, time, etc. .For example, assume we have an ordered date dimension with the concept hierarchyYear-Month-Day, a query containing a set of values {[2011-*-*, 2013-*-*]} meansdate from 2011 to 2013. To use MDS ids to represent this query, it becomes {[2011-0-0, 2013-0-0]}. The 0s in the ids can be interpreted as * that we usually have it inSQL queries and it means All.

2. Multiple MDS Ids at any level of the hierarchy of a dimension i.Each MDS Id in a hierarchy level l in dimension i covers many distinct MDS Ids inthe level l+1 of the dimension. This type of query values is used for both ordered andunordered dimensions such as Location, Product, Date, etc. For example, assume wehave a store dimension having the hierarchy Counry-Province-City-StoreId, a querycontaining a set of values {Alberta, Ontario} means all stores located in the provincesof Alberta and Ontario. The MDS Ids that represent the query is {Canada-Alberta-0-0, Canada-Ontario-0-0}. In the implementation, 'Canada', 'Alberta', 'Ontario' and'0' is presented in total 64-bits integer. There are two types queries that the CR-OLAP system can support to answer. One is arrange query and the other one is a point query.

1. Range queries for a dimension i:

qi={ [low IDi1 , high IDi1] , [low IDi2 , high IDi2], ... , [low IDin , high IDin] } , where low IDij represents the lower bound of the jth given range in dimension i, andhigh IDij represents the upper bound of the jth given range in a dimension i.

2. Point queries for dimension i:

qi={ IDi1 , IDi2 ,... , IDin } , where IDij represents the jth given MDS ID in dimension i. Note that IDs can be in different levels of dimension i.

4. EXPERIMENT AND TESTING OF CR-OLAP

4.1 Introduction

The network communicator, the message serializer and the PDCR-tree is developed as separate APIs. The ZeroMQ library is used for thenetwork communicator and the boost serialization library is used for the message serializer.We conducted a large amount of tests to measure the system response time and throughput.The tests are performed on the Amazon cloud environment and astream OLAP system using One-Dimensional Index which is a linear array structure to handledata insertions and search queries in parallel. We explained that the CR-OLAP systemis efficient by comparing it against the Stream-OLAP system.

4.2 Experimental environment

On the Amazon cloud, there are various types of instances available. We selected theinstance of m2.4xlarge for the master node and the M3.2xlarge instance for the worker _nodes. The M2.4xlarge is an optimized instance for memory-intensive applications. Itcontains 68G memory and 8 virtual CPUs which has total 26 elastic computing units(ECU).Each ECU provides the equivalent CPU capacity of a 1.0 - 1.2 GHz 2007 Opteron or2007 Xeon processor. The M3.2xlarge instance also has 8 virtual CPUs but it only has30G memory size. The CC2.8xlarge instance and the CR1.8xlarge instance are claimedas cluster instances. The cluster instances can be built in one network which could have10G network speed. They are 32 cores with hyper threading enabled and also have highmemory storages. The details are listed in the Table 4.1.We also performed experiments on the Ontario Research and Education VCL cloud. Instance Family Instance Type CPU Arch vCPU ECU Memory (GiB) Network Memory optimized m2.4xlarge 64-bit 8 26 68.4 High General purpose m3.2xlarge 64-bit 8 26 30 High Compute optimized cc2.8xlarge 64-bit 32 88 60.5 10 Gigabit. Memory optimized cr1.8xlarge 64-bit 32 88 244 10 Gigabit. The instance we choose has 16 cores, 32G memory, 2199 MHZ cpu. A Linux operating system CentOS 6.3 is installed on each instance with the GNU GCC4.7 compiler. It supports Open MP to handle multi-threading to parallelize processes. A ZeroMQ 3.2 is installed as message passing middleware to transfer the data or transaction instructions between machines.

Table 4.1: Specifications of Amazon cloud instances used in our experiments

Instance Family	Instance Type	CPU Arch	vCPU	ECU	Memory(GiB)	Network
Memory optimized	m2.4xlarge	64-bit	8	26	68.4	High
General purpose	m3.2xlarge	64-bit	8	26	30	High
Compute optimized	cc2.8xlarge	64-bit	32	88	60.5	10 Gigabit
Memory optimized	cr1.8xlarge	64-bit	32	88	244	10 Gigabit

4.2.1 Experimental data

In all experiments of the CR-OLAP, we still use the TPC-DC benchmark which was used inthe experiments of the PDC-tree for multi-core processors. The underlying business modelof the TPC-DS is a retail product supplier. We select the "Store Sales" fact table which isthe largest fact table among all seven fact tables. The store sales fact table contains 8 dimensions and several measures such as quantity, net paid, net profit, etc. The graph 4.1 liststhe dimensions and the hierarchy schema of each dimension that was used in the system.The 8 dimensions include item, store, customer, date dim, time dim, promotion, house-hold demographics, customer address. In the graph, the dimensions in left side contain ordered data and the dimensions in right side contain unordered data.Multiple tests were conducted to evaluate the time of completing insertions and querytransactions using distributed PDCR-tree index. The following scenarios are selected forour experiments.

(1) Increasing number of dimension

(2) Increasing number of workers

(3) Increasing data size

(4) Increasing data coverage for queries

(5) Using combination of star (*) at different hierarchy levels for queries

(6) Running on different types of cloud instances

(7) Increasing number of cut level

(8) Increasing number of directory node capacity in master/workers

HammedZaboli performed the experiments of the CR-OLAP for item 1 to 5, I performedthe experiments of the CR-OLAP for item 6 to 8.

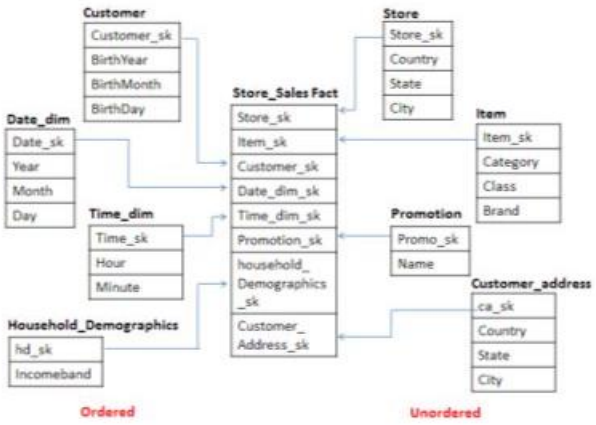


Figure 4.1: Store Sales Fact table and 8 dimensions and each with its hierarchy scheme

4.2.2 Comparison baseline

In addition, a Stream-OLAP system with a 1-dimensional array index (1D-Index) is implemented and used to be compared with the CR-OLAP system with a PDCR-tree index.Data nodes in the 1D-Index structure are as the same as they are in the PDCT-Tree index. The Stream-OLAP system creates an array for every value in the highest hierarchylevel of a dimension and data nodes are stored in the related arrays according to the values in the highest hierarchy level of a dimension. For example, the 1D-Index builds anarray containing all data nodes who have the value of "2012" in the highest hierarchy level"Birth Year" of the "customer" dimension. If there are 10 different values for "BithrYear"level, the Stream-OLAP creates 10 arrays for the value of each year. Data nodes in eacharray are not sorted and are inserted at the end of the array as they arrive. Arrays are evenlydistributed between the workers to assure parallel processing of insertions and queries. Ineach array, multiple queries may search the arrays in parallel. The 1D-index structures issued to compare its performance with the PDCR-tree's performance and to evaluate the impact of a single-dimensional index versus a multi-dimensional index for Hierarchical multidimensional databases. The 1D-index and the PDCR-tree are different from a B-tree andan R-tree since the later indices do not designed for the data having hierarchy structures. Mr. Kong performed the experiments of the Stream-OLAP for item 1 to 5.

4.3 Analysis of Results

In the following section, we will demonstrate the experiments results. All tests are performed on m+1 machine (a single master node plus m worker nodes) with 16 threads oneach node to concurrently process tasks.

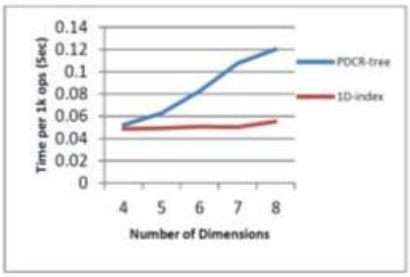
Test 1: Increasing the number of dimensions

The test is to evaluate the impact on the number of dimensions. The master node isusing an m2.4xlarge instance and all the worker nodes are using the m3.2xlarge instances.The test is performed on 8 workers (m = 8) with 40 million tuples from the fact table (N =40M) as initial insertions followed by three sets of 1000 queries (q = 1000). The numberof dimensions d is increased from 4 to 8 ($4 \leq d \leq 8$). The three sets of queries includenon-star regular queries, with the coverage 10%,60%, and 95% respectively. The coverage is measured by the percentage of the number ofquery values over the cardinality of the values of an attribute in a certain hierarchy levelof a dimension. For example, an unordered dimension Customer Address has 50 differentvalues of the states in US. When the values of all queries cover 48 states, we say thecoverage is 95%.Figures 4.2 and 4.3 demonstrate the results of the test.

Test 2: Increasing number of workers

The test is used to evaluate the impact by the increasing number of workers. The master node is using a m2.4xlarge

instance and all worker nodes are using the m3.2xlargeinstances. The test is performed on 10 million tuples from the fact table (N = 10M) with8 dimensions (d=8) as initial insertions followed by three sets of 1000 queries (q = 1000).The number of workers m is increased from 1 to 8 ($1 \leq m \leq 8$). The three sets of queriesare no-star regular queries with coverage 10% (low), 60% (medium) and 95% (high) respectively. We choose 10 million data size as the fixed data size so that it does not exceedthe memory size of a single worker machine.Figure 4.4: Time for 1000 insertions as a function of the number of workers.(N = 10M,k = 16,d = 8)Figure 4.5: Time for 1000 queries with different query coverage's as a function of the number of workers.(N = 10M,k = 16,d = 8)Figure 4.6: Speedup for 1000 queries with different query coverage as a function of the number of workers.(N = 10M,k = 16,d = 8)Figure 4.4 shows the Stream-OLAP outperforms the CR-OLAP. The 1D-Index does not get speedup on insertion time. We did a test to break down the execution time of everyoperation involved in the Stream-OLAP and found it spent most time on data serializationand network communication. The actual data appending operation only took very littletime. Each insertion requires serialization and data transportation from the master to theworkers. Therefore, no matter how many workers are used, the total time on completing1000 insertions is very close. The PDCR-tree is slower in insertions since it has overheadlike node splits during insertions. But while increasing the number of workers, the PDCR-tree insertion time is speedup since the distributed data structure allows multiple workers toprocess insertions concurrently. Figure 4.5 illustrates that the time for queries is decreasingby increasing the number of workers (m). Figure 4.6 demonstrates that both the PDCR-treeand the 1D-Index achieve close to a linear speedup by increasing the number of workers.However, the PDCR-tree takes a much smaller absolute time to run queries for all cases ofquery coverage.



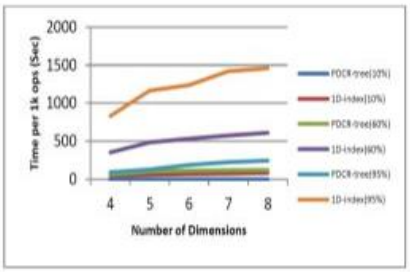


Figure 4.4: Time for 1000 insertions as a different query coverage's as a function of the number of workers.(N = 10M;k = 16; d = 8)

Figure 4.5: Time for 1000 queries with function of the number of workers.(N =10M;k = 16;d = 8)

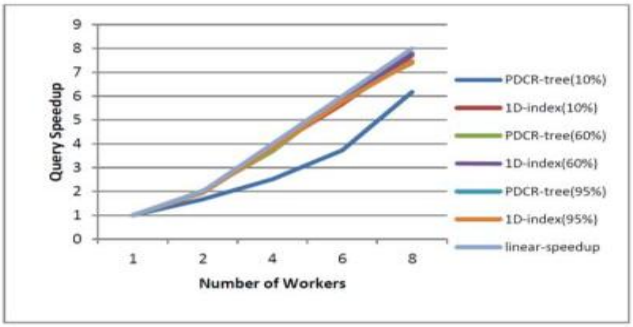


Figure 4.6: Speedup for 1000 queries with different query coverage as a function of thenumber of workers.(N = 10M;k = 16;d = 8)

Test 3: Increasing data size

The third test is used to evaluate the performance by scaling up the size of systems . Thenumber of worker nodes is increased from 1 to 8 ($1 \leq m \leq 8$) to handle the increasing datasize from 10 million to 80 million tuples ($10M \leq N \leq 80M$) respectively. The master nodeis using a m2.4xlarge instance and all the worker nodes are using the m3.2xlarge instances.The data set processed has 8 dimensions (d=8) and three sets of 1000 queries (q = 1000)are executed after data insertions.Figure 4.7: Time for 1000 insertions as a function of the data size (number of items currently stored).(k = 16,d = 8)Figure 4.8: Time for 1000 queries with different query coverage as a function of the data size (number of items currently stored).(k=16,d=8).From figure 4.7and 4.8, we observe that by increasing the number of workers to process the increasing size of data the insertion time is decreased and the query performanceof the system stays stable. The execution of insertions in the CR-OLAP is slower thanthe execution of insertions in the Stream-OLAP. However when the number of workers isincreased to 8, the total time of insertions in both systems is close. Conversely, the querytime of the PDCR-tree is significantly faster than the query time of the 1D-index is. Thistest shows the system can be scaled up and process much larger data size with more numberof workers without slowing down overall performance.

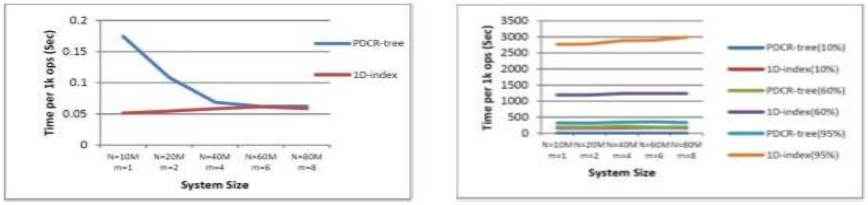


Figure 4.7: Time for 1000 insertions as a function of the data size (number of items currently stored).(k = 16;d = 8)

Figure 4.8: Time for 1000 queries with different query coverage as a function of the data size (number of items currently stored).(k = 16;d = 8)

Test 4: Increasing query coverage

This test is used to evaluate the impact of every individual dimension on queries performance with different coverage. It was performed on 40 million tuples from the fact table(N=40M) with 8 dimensions (d=8) as an initial data set. The number of workers is 8 (m=8).We executed 9 sets of queries with the query coverage from 10% to 90% and from 91%to 99%. Among the 9 sets, 8 sets have a "*" in one of the dimensions in each and one query set contains no-star regular queries.Figure 4.9: Time for 1000 queries as a function of query coverages in PDCR- tree Impact of queries having value "*" for different dimensions.(N = 40M,m = 8,k = 16,d = 8,10% <= Coverage <= 90%)Figure 4.10: Time for 1000 queries as the query coverage in PDCR-tree Impact of queries having value "*" for different dimensions.(N = 40M,m = 8,k = 16,d = 8,91% <=Coverage <= 99%)Figure 4.9 and 4.10 show the CR-OLAP is efficient with either very low or very high query coverage. When the query with a low coverage like 10%, 20% or 30%, there are nottoo many directory nodes whose MDSRs intersect the MDSs of a query, so the PDCR-treeonly traverses a small amount of subtrees; When the query has a very large coverage like95% to 99%, the results should be containing a large portion of the data in the database.When traversing the PDCR-tree, the MDS of a query covers the MDSRs of many directorynodes in the top part of tree and the aggregation value stored in those directory nodes canFigure 4.11: Time for 1000 queries as a function of query cover ages in 1D- Index Impact of queries having value "*" for different dimensions.(N = 40M,m = 8,k = 16,d = 8,10% <= Coverage <= 90%)Figure 4.12: Time for 1000 queries as the query coverage in 1D-Index Impact of queries having value "*" for different dimensions.(N = 40M,m = 8,k = 16,d = 8,91% <=Coverage <= 99%)be reported as a part of results. The PDCR-tree doesn't need tracing down very deep inthe tree. However, the 1D-index does not have such advantage. The higher coverage thequery result has, the more data nodes in the arrays need to be scanned. As a result, theperformance is constantly decreased by increasing the query coverage. (see figure 4.11,and 4.12)Comparing the PDCR-index to the 1D-tree, the performance of a PDCR-tree is at least 5to 20 times faster than the performance of a 1D-index is in most cases. The Figure 4.13 and 4.14 show the ration of the query time of 1D-index over the query time of the PDCR-tree. Itproves that the performance of the CR-OLAP beats the performance of the Stream-OLAP.Figure 4.13: Ratio of 1D-index/PDCR- tree taken for 1000 queries as the query coverage increasesFigure 4.14: Ratio of 1D-index/PDCR- tree taken for 1000 queries as the query coverage increases

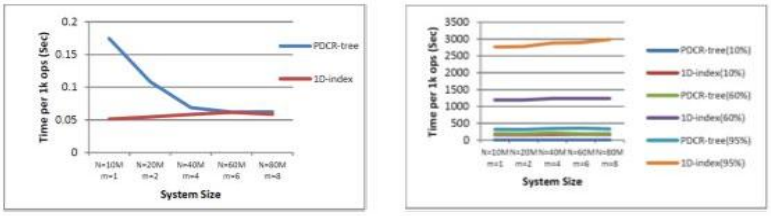


Figure 4.9: Time for 1000 queries as a function of query coverages in PDCR-tree impact of queries having value "*" for different dimensions (N = 40M;m =8;k = 16;d = 8;10% <= Coverage <=90%)

Figure 4.10: Time for 1000 queries as the query coverage in PDCR-tree Impact of queries having value "*" for different dimensions. (N = 40M;m = 8; k =16;d = 8;91% <=Coverage <= 99%)

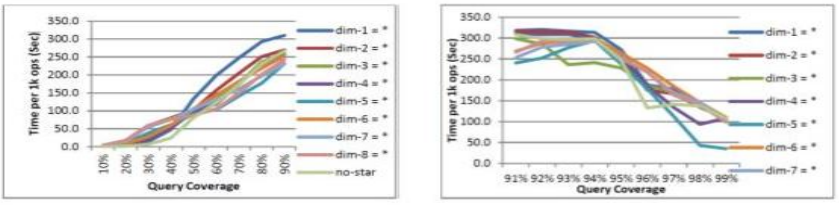


Figure 4.11: Time for 1000 queries as a function of query coverages in 1D index impact of queries having value "*" for different dimensions.(N = 40M;m =8;k = 16;d = 8;10% <= Coverage <= 90%)

Figure 4.12: Time for 1000 queries as the query coverage in 1D-Index Impact of queries having value "*" for Different dimensions. (N=40M;m=8,k=16;d=8;91% <=<=Coverage <= 99%)

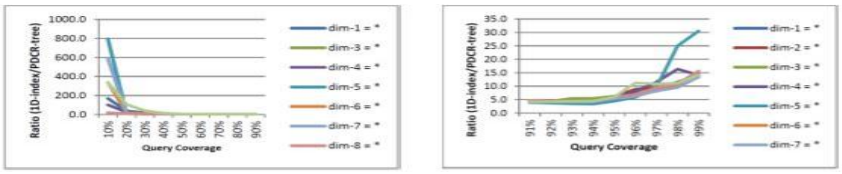


Figure 4.13: Ratio of 1D-index/PDCR Tree taken for 1000 queries as the query coverage increases

Figure 4.14: Ratio of 1D-index/PDCR- tree taken for 1000 queries as the query coverage increases

Test 5: Varying query pattern with star at different hierarchy levels

This test is used to evaluate the performance of selected query patterns. It is performedon 40 million tuples from the fact table (N = 40M) with 8 dimensions (d=8) as initialinsertions. The number of workers is 8 (m=8). The query sets include queries havingone or more "*" at different hierarchy levels in the "Date Dim" dimension. We selected7 combinations of "*" and values in the hierarchy levels "Year", "Month" and "Day" asthe query patterns for our test. They are *-*-*, year-*-*, year-month-*, year-month-day, *-month-*, *-month-day and *-*-day. Those query patterns cover many OLAP queries suchas "Total sales in stores located in Ontario and Alberta from February to May of all years"or "Total sales in all stores in Ottawa in May 2012" etc. We generated 3 sets of queries foreach pattern with the coverage 10%, 60% and 95% respectively.Figure 4.15: Time for 1000 queries as a function of query coverage for queries with multiple "*" values for PDC-tree.(N = 40M,m = 8,k = 16,d = 8)Figure 4.16: Time for 1000 queries as a function of query coverage's for queries with multiple "*" values for both PDCR- tree and 1D-index.(N = 40M,m = 8,k = 16,d = 8)From figure 4.15, we observe that when the query has value in a higher hierarchy level,the CR-OLAP performs better. Figure 4.16 also demonstrates thatthe PDCR-tree consistently outperforms the 1D-index no matter what query coverage andquery patterns are. Therefore, the CR-OLAP system has the advantage to support all kindsof OLAP queries and operations.

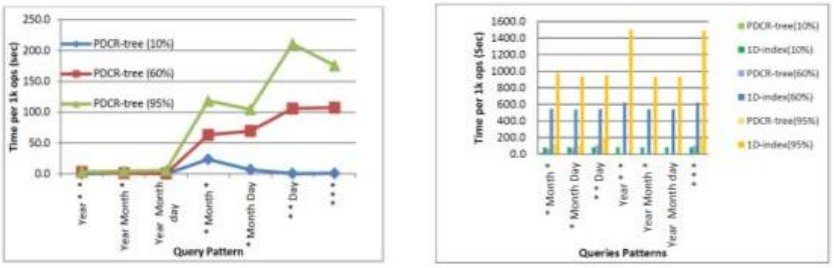


Figure 4.15: Time for 1000 queries as a function of query coverage for queries with multiple "*" values for PDC-tree.(N = 40M;m = 8;k = 16;d =16;d = 8)

Figure 4.16: Time for 1000 queries as a function of query coverages for queries with multiple "*" values for both PDCR- tree and 1D-index.(N = 40M;m = 8;k =8)

Test 6: Changing the type of cloud instance

This test is used to evaluate the CR-OLAP performance on different types of cloud instances. Five types of instances are selected and they are the VCL HPC instance, the Amazon cc2.8xlarge, cr1.8xlarge, m3.2xlarge and m2.4xlarge. The experiments are performedwith initial 10 million data (N=10M) with 8 dimensions (d=8) as insertions followed bythree sets of 1000 queries (q=1000) with 10%, 60% and 95% coverage respectively. Webuilt the system on 5 different instances and each time created 5 nodes including one masterand 4 workers.Figure 4.17: Time for 1000 queries on different type cloud instances.(N = 10M,m = 4,k = 16)Figure 4.18: Time for 1000 insertions on different type cloud instances.(N = 10M,m = 4,k = 16)Figure 4.17 and 7.18 show that the CR-OLAP is running faster on the Amazon cloudthan it is on the VCL cloud. Amazon describes an EC2 computing unit (ECU) to measure itsCPU capacity. We expectwhen the number of threads is increased in the system or the data size is increased, thecluster instances can outperform the regular instances.

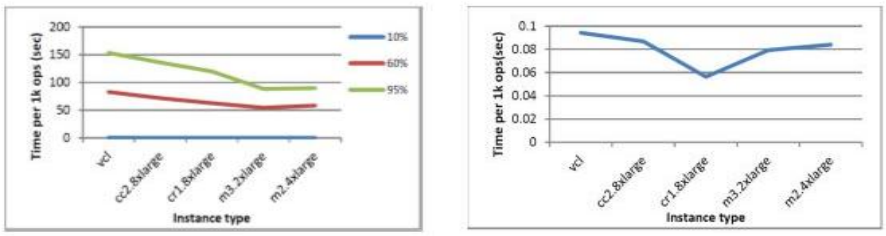


Figure 4.17: Time for 1000 queries on different type cloud instances.(N =10M;m = 4;k = 16)

Figure 4.18: Time for 1000 insertionson different type cloud instances.(N =10M;m = 4;k = 16)

Test 7: Increasing number of cut level

The test is used to evaluate the impact of the cut level. The experiments are performedwith an initial 10 million data (N=10M) with 8 dimensions (d=8) as insertions followedby 1000 queries (q=1000) with 10% coverage. The cut level is the depth of the hat in aPDCR-tree on a master node. We increase the cut level from 1 to 10.Figure 7.19: Time for 1000 queries as a function of the number of cut level in hat.(N = 10M,m = 4,k = 16)Figure 4.20: Time for 1000 insertions as a function of the number of cut level in hat.(N = 10M,m = 4,k = 16)The Figure 4.19 and 4.20 show that the better performance can be achieved when thecut level is smaller. The system is designed to use multiple parallelisms including parallelcomputing by multiple workers and parallel multi-threads on each worker. The CR-OLAPdistributes the data to multiple workers, therefore, insertions and queries can be dispatchedto multiple workers to be executed concurrently not only within a worker but also on multiple workers in order to improve computing speed.

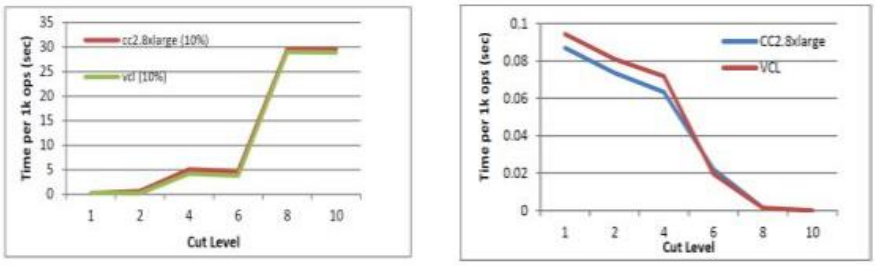


Figure 4.19: Time for 1000 queries as a function of the number of cut level in hat.(N = 10M;m = 4;k = 16)

Figure 4.20: Time for 1000 insertions as a function of the number of cut level in hat.(N = 10M;m = 4;k = 16)

Test 8: Increasing directory node capacity

This test is used to evaluate the impact of the directory node capacity. The experiments are performed with an initial 10 million data (N=10M) with 8 dimensions (d=8) asinsertions followed by four sets of 1000 queries (q=1000) with 10%, 20%, 60% and 95%coverage respectively. We performed two experiments by varying the node capacity from10 to 35 in the hat only and in the workers only as well. When changing the capacity in thehat, the directory node capacity in the workers is fixed to 15, while changing the capacityin the workers, the node capacity in the hat is fixed to 10.Figure 4.21: Time for 1000 queries as a function of the number of directory node capacity in hat.(N = 10M,m = 4,k = 16)Figure 4.22: Time for 1000 insertions as a function of the number of directory node capacity in that. (N = 10M,m = 4,k = 16)We notice that when increasing the node capacity of directory nodes in the hat, thequery time is decreased in high data coverage but increased in a low coverage. Whenthe directory node capacity is increased, the tree increases its flatness and contains moredirectory/data nodes in the hat. Figure 4.21 demonstrates that for a low coverage, the querytraverses more nodes in the hat so it slows down the search process; for a high coverage, thequery can get results from certain directory nodes in the hat if the MDSRs is fully coveredwithin query's MDSs so it speeds up the performance. Figure 4.22 shows the insertiontime is improved when the node capacity is increased since less node splits happen duringinsertions.Figure 4.23: Time for 1000 queries as a function of the number of directory node capacity in workers.(N =10M,m= 4,k = 16)Figure 4.24: Time for 1000 insertions as a function of the number of directory node capacity in workers.(N =10M,m= 4,k = 16)From figure 4.23, for a fixed directory node capacity 10 in the hat, we observe that thePDCR-tree reaches its optimal output when the directory node capacity in the workers isnear 10 for queries with a high coverage and is around 40 for queries with a low coverage.For insertions, the experiment shows the result reach the best at the capacity of 15 (Figure 4.24).Furthermore, Figure 4.26: Time for 1000 insertions as a function of the number of di- rectory node capacity in both hat and workers.(N = 10M,m = 4,k = 16)With the above tests, we know that when changing the value of parameters like a cutlevel and a directory node capacity, the CR-OLAP can have different behaviors. With afixed 10 million data set, the system can reach the best output with the directory nodecapacity around 10-15 at the cut level 1 or 2.

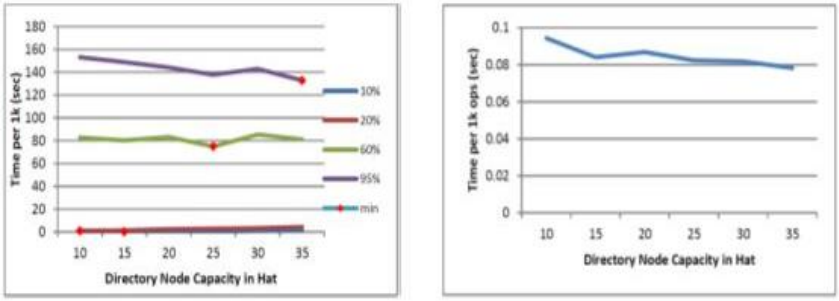


Figure 4.21: Time for 1000 queries as a function of the number of directory node (N =10M;m=4;k =16)

Figure 4.22: Time for 1000 insertions as a function of the number of directorycapacity in hat. node capacity in hat.(N = 10M;m = 4;k = 16)

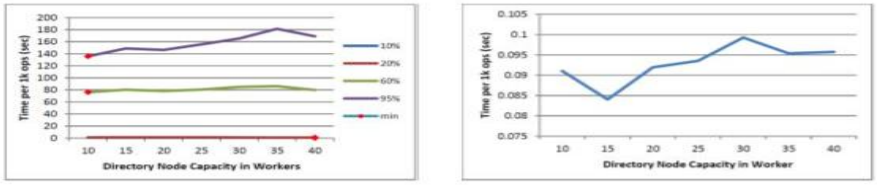


Figure 4.23: Time for 1000 queries asa function of the number of directory node capacity in workers.(N =10M;m=4;k = 16)

Figure 4.24: Time for 1000 insertions as a function of the number of directory node capacity in workers.(N =10M;m=4;k = 16)

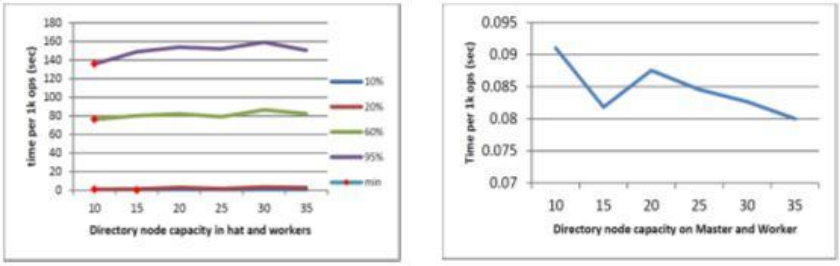


Figure 4.25: Time for 1000 queries as afunction of the number of directory node capacity in both hat and workers.(N = 10M;m = 4;k = 16)

Figure 4.26: Time for 1000 insertions as a function of the number of directory node capacity in both hat and workers.(N = 10M;m = 4;k = 16)

5. CONCLUSION

Business Intelligence and its tools such as data warehouses, OLAP servers and report systems have their important roles in organizations to assist in making decision precisely andreacting to market changing quickly. With the increasing globalization of market and thegrowing complexity of businesses, there is an increasing demand on BI and OLAP systems. The current OLAP systems have been facing challenges such as storage spaces foran increasing data size, query responses for an efficient performance, or data freshness foranalysis in real-time. In many common practices, an OLAP server loads data periodicallyto its data cubes. The loading procedures usually can take hours to be completed sincemany extra data and calculations are added during loading. The cube contains mostly staticand pre-aggregated data but provides multi-dimensional views to users. Such a fashioncannot meet the needs of modern businesses. Many decision makers are seeking real-timeOLAP systems which can provide the analysis with integrated data source in real-time suchthat the results are not based on outdated information. However, it is a challenge to providesuch a system which involves large portion data aggregation and responses user requestswith low latency and high throughput.

6. REFERENCES

[1] M. C. Kurt and G. Agrawal, “A fault-tolerant environment for large- scale query processing,” in High Performance Computing (HiPC), 2012 19th International Conference on, 2012, pp. 1–10.

[2] H. Al-Aqrabi, L. Liu, R. Hill, and N. Antonopoulos. Taking the business intelligence to the clouds. In High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on, pages 953–958. IEEE, 2012.

[3] D. Jin and T. Tsuji. Parallel data cube construction based on an extendible multidimensional array. In Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on, pages 1139–1145. IEEE, 2011.

[4] K. Doka, D. Tsoumakos, and N. Koziris, “Brown dwarf: A fully- distributed, fault-tolerant data warehousing system,” J. Parallel Distrib. Comput., vol. 71, no. 11, pp. 1434–1446, Nov. 2011.

[5] Y. Zhai, M. Liu, J. Zhai, X. Ma, and W. Chen. Cloud versus in-house cluster: evalu- ating amazon cluster compute instances for running mpi applications. In State of the Practice Reports, page 11. ACM, 2011.

[6] P. Brezany, Y. Zhang, I. Janciak, P. Chen, and S. Ye. An elastic olap cloud plat- form. In Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on, pages 356–363. IEEE, 2011.

[7] Y. Cao, C. Chen, F. Guo, D. Jiang, Y. Lin, B. C. Ooi, H. T. Vo, S. Wu, and Q. Xu. Es¡ sup¿ 2¡/sup¿: A cloud data storage system for supporting both oltp and olap. In Data Engineering (ICDE), 2011 IEEE 27th International Conference on, pages 291–302. IEEE, 2011.

[8] Asiki, D. Tsoumakos, and N. Koziris, “Distributing and searching concept hierarchies: an adaptive dht-based system,” Cluster Computing, vol. 13, no. 3, pp. 257–276, Sep. 2010.

[9] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, “Dremel: interactive analysis of web-scale datasets,” Proc. VLDB Endow., vol. 3, no. 1-2, pp. 330–339, Sep. 2010.

[10] Y. Zhang, S. Wang, and W. Huang. Paracube: A scalable olap model based on dis- tributed aggregate computing

with sibling cubes. In Web Conference (APWEB), 2010 12th International Asia-Pacific, pages 323–329. IEEE, 2010.

[11] Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy. Hive-a petabyte scale data warehouse using hadoop. In Data Engineering (ICDE), 2010 IEEE 26th International Conference on, pages 996–1005. IEEE, 2010.

[12] Z. Guo-Liang, C. Hong, L. Cui-Ping, W. Shan, and Z. Tao, "Parallel Data Cube Computation on Graphic Processing Units," Chines Journal of Computers, vol. 33, no. 10, pp. 1788–1798, 2010.

[13] X. Zhang, J. Ai, Z. Wang, J. Lu, and X. Meng. An efficient multi-dimensional index for cloud data management. In Proceedings of the first international workshop on Cloud data management, pages 17–24. ACM, 2009.

[14] R. J. Santos and J. Bernardino. Optimizing data warehouse loading procedures for enabling useful-time data warehousing. In Proceedings of the 2009 International Database Engineering & Applications Symposium, pages 292–299. ACM, 2009.

[15] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so- foreign language for data processing. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1099–1110. ACM, 2008.

[16] R. J. Santos and J. Bernardino. Real-time data warehouse loading methodology. pages 49–58, 2008.

[17] D. Power. A brief history of decision support systems, version 4.0, march 10, 2007. Series A Brief History of Decision Support Systems. Version, 4, 2007.

[18] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, et al. The landscape of paral- lel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[19] S. S. Conn. Oltp and olap data integration: a review of feasible implementation meth- ods and architectures for real time data analysis. In SoutheastCon, 2005. Proceedings. IEEE, pages 515–520. IEEE, 2005.

[20] Amazon elastic compute cloud (amazon ec2). http://aws.amazon.com/ec2/.

[21] Amazon ec2 instance details. http://aws.amazon.com/ec2/.

7. AUTHOR'S PROFILE

Mani Sarma Vittapu received a PhD in Computer Science and Engineering in 2013 from AcharyaNagarjuna University, Guntur, Andhra Pradesh, India. In February 2014,joined the ITSC department at Addis Ababa Institute of Technology (AAIT), Addis Ababa University, Ethiopia as an assistant professor. My research interests are in machine-learning, a subfield that lies at the intersection of statistics and computer science. I am interested in three aspects of machine-learning -- unsupervised learning, online learning and privacy-preserving machine learning. In unsupervised learning, the goal is to extract information from unlabeled data to assist various learning tasks. In online learning, data arrives one at a time, and the challenge is to make good predictions on the face of changing data and models. Privacy-preserving machine learning addresses the problem of learning a good predictor from the data, while ensuring the privacy of individuals in the training data set. Topics in machine-learning, in particular, Data Models Design and Implementation, clustering or unsupervised learning, online learning and privacy-preserving machine-learning, Cloud, Parallel and Distributed computing.