CCIndex: a Complemental Clustering Index on Distributed Ordered Tables for Multi-dimensional Range Queries

Presented by Jia Liu

liujia09@software.ict.ac.cn

2010-9-4

Outlines

- Introduction
- Why CCIndex?
- Data layout and management
- Query processing and optimization
- Fault tolerance
- Implementation and Evaluations
- Conclusion
- Related work

Multi-dimensional Range Queries

❖ Wide usage in Net Computing

- In Grid, find cluster
  - VO = ‘HPCVO’ and nodes > 32 and queueLength < 4
- In Nagios™, find CPU monitoring info
  - select host, service, time, status from MonitoringData where host=’node 216’ and service=’CPU Load’ and (time > 1260610511 and time < 1260610521)
- In Internet Apps, find nearby restaurants
  - “latitude > 48.5 and latitude < 48.6 and longitude > 112.5 and longitude < 112.8 and type = restaurants”
- In Flickr™, find latest hot photos
  - timestamp > 1267660008 and rank > 1000
- Index queries in Databases, Queries in P2P

Multi-dimensional Range Queries

- More and more data
  - In Nagios, with 1000 nodes, 10 services，1 minutes period →14.4 million records one day
  - Flickr has more than 4 billion photos at Oct. 2009
- New challenges for large scale data
  - high performance, low space overhead, and high reliability
- By Database? By P2P?
- By DOTs like BigTable [Cooper08] or PNUTS [Cooper08]!

Distributed Ordered Tables (DOTs)

- Introduced by Yahoo! [Silberstein08]
  - partitions continuous keys to regions, replicates regions, distributes regions to shared-nothing servers
  - serves as tables and columns, supports range queries on primary keys.
- Our way: Add Index to DOTs to support Multi-dimensional Range Queries

Problem formulation

- How to build index in DOTs to support Multi-dimensional Range Queries with high performance, low space overhead, and high reliability?

- none of existing schemes work well!
  - Secondary Index
  - Clustering Index

CCIndex main idea

- Important facts
  - usually 3 to 5 replicas in DOTs
  - indexes number is usually less than 5
  - random reads is significantly slower than scan: 12.7X
- CCIndex main idea
  - Re-organizes replica as several **Complemental Clustering Index Tables** containing full row data, to convert the slow random reads to fast range scan
  - leverages the region-to-server mapping information to estimate the result size
  - Introduces **Complemental Check Table** for record level replica to support incremental data recovery
- CCIndex is short for Complemental Clustering Index

Data layout and management

- Complemental Clustering Index Tables (CCIT)
  - Convert random read to sequential
- Complemental Check Table (CCT)
  - for fast data recovery

Complemental Clustering Index Table (CCIT0)

id	idx1	idx2	info
001	cpu	n1	info1
002	mem	n1	info2
003	net	n3	info3
004	cpu	n2	info4

id	idx1	idx2

Complemental Check Table, CCT0, replicated

CCIT2, key2=idx2+id+idx2Length

key2	idx1	info
n100102	cpu	info1
n100202	mem	info2
n200402	cpu	info4
n300302	net	info3

key2	idx1

CCT2, replicated

CCIT1, key1=idx1+id+idx1Length

key1	idx2	info
cpu00103	n1	info1
cpu00403	n2	info4
mem00203	n1	info2
net00303	n3	info3

key1	idx2

CCT1, replicated

Primary key

Index column

data

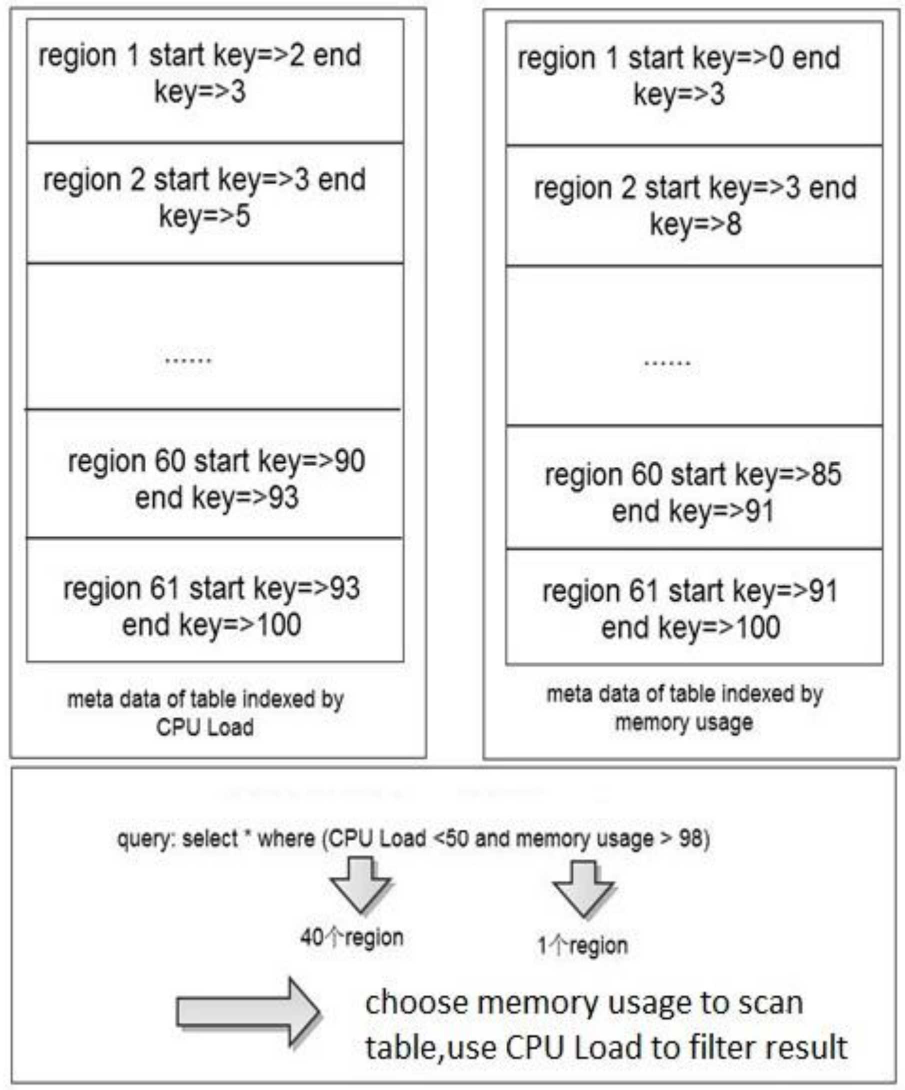
Query processing and optimization

Query processing

- Support select and where, SQL-like
- Build query plan tree and do simple optimization to eliminate redundant range queries

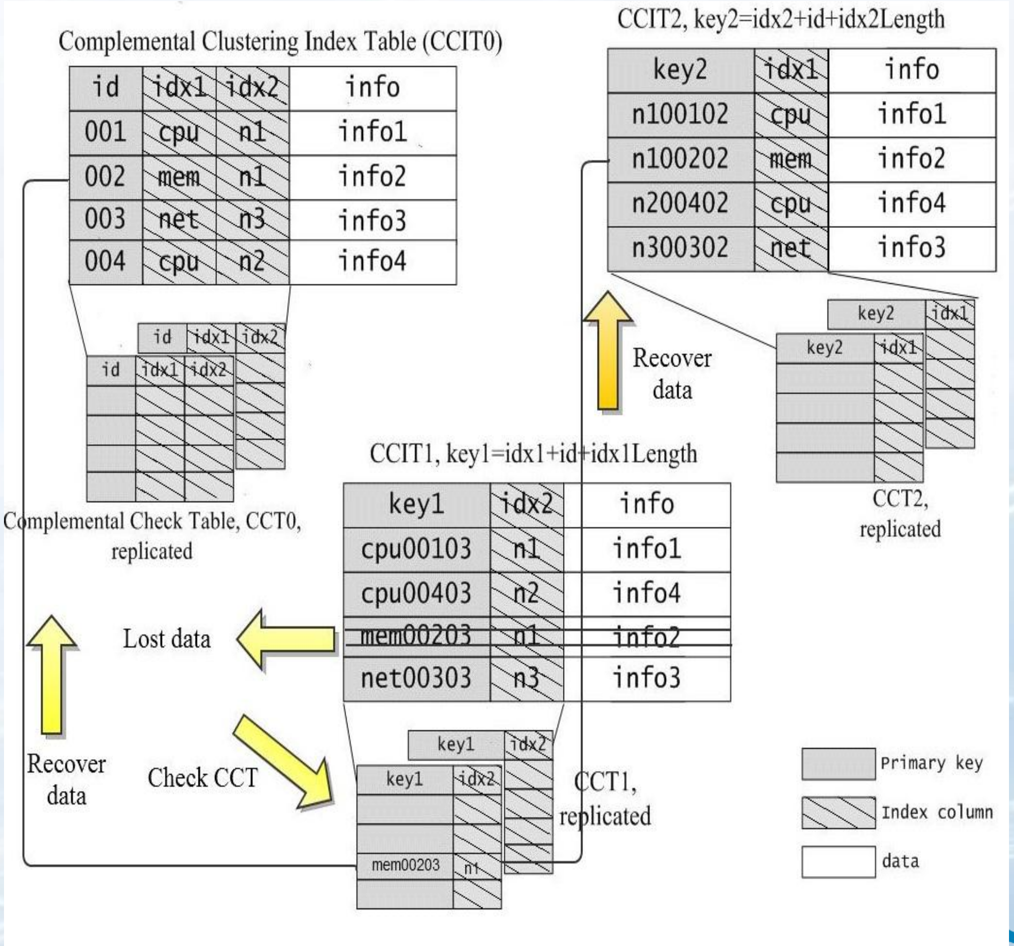
Query optimization

- To choose best sub-queries without statistics
- statistics are very difficult to gather and maintain in massive scale tables
- To estimate result size by the covered region numbers for the query range
  - Single region size: $S_{max}/2 < S < S_{max}$



Fault tolerance

- Rrecord level data recovery mechanism by Complemental Check Table (CCTs)
  - Get other index value from CCTs
  - Query the CCITs to recovery data
  - Replica CCTs themselves
- CCTs Maintaince
  - Asynchronous updating CCTs

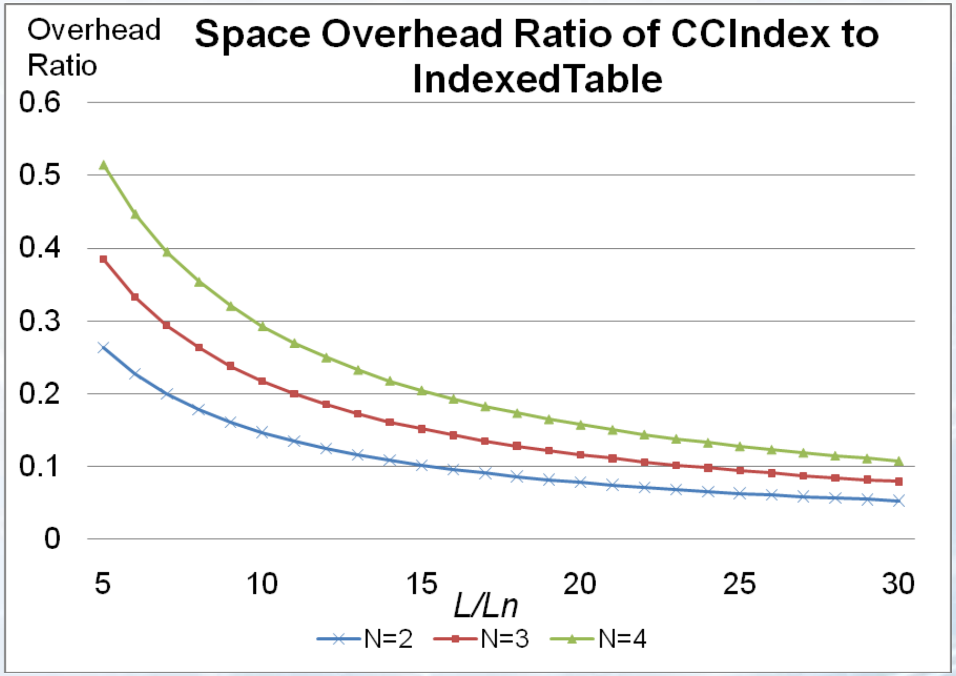


CCIndex Implementation

- Based on Apache Hbase 0.20.1, Java
  - Disables HDFS replica
  - Stores CCITs and CCTs in HBase
  - Data in CCITs and CCTs are distributed physically
- The comparing system: IndexedTable in Hbase, a secondary index scheme
- MySQL Cluster

Evaluations: Theoretical analysis

- The space overhead ratio of CCIndex to IndexedTable is $(N*N+1)/(2*N+(N+1)*L/L_n$



If N changes from 2 to 4 and the L/L_n changes from 10 to 30, then the overhead changes from 5.3% to 29.3%

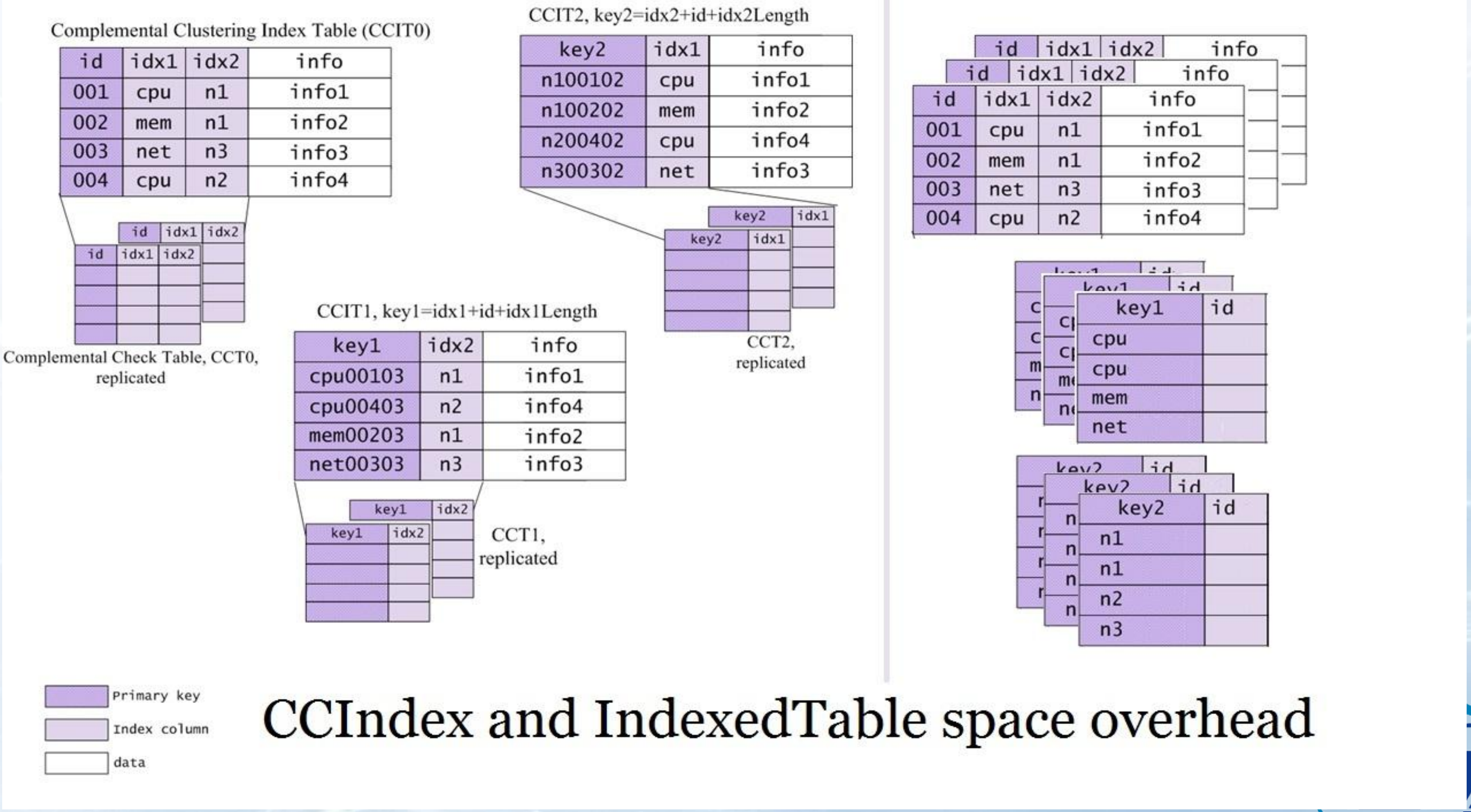
N is number of index

Ln is the length of indexed columns

L is the length of row

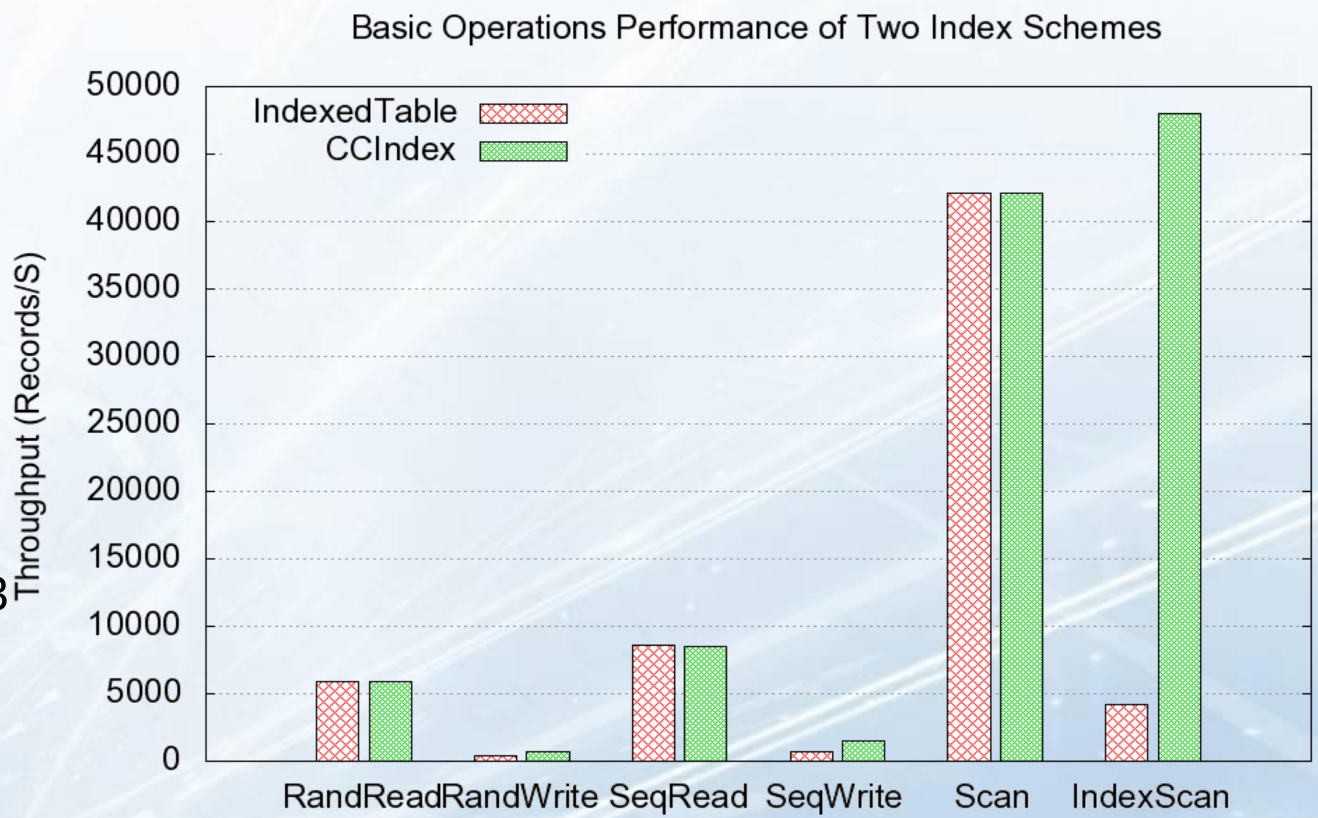
Evaluations: Theoretical analysis

The space overhead ratio of CCIndex compared to IndexedTable



Evaluations: Micro benchmarks

- Experimental Setup: 3 nodes
  - Dual CPU, 4 cores, 1.8 GHz AMD Opteron，6 GB Memory，321 GB RAID5 SCSI DISK，Gigabits Ethernet
  - Red Hat CentOS (kernel 2.6.18), ext3 FS, Sun JDK1.6.0_14, Hadoop 0.20.1, HBase 0.20.1
- Workload
  - 1million 1KB records, 3 indexes

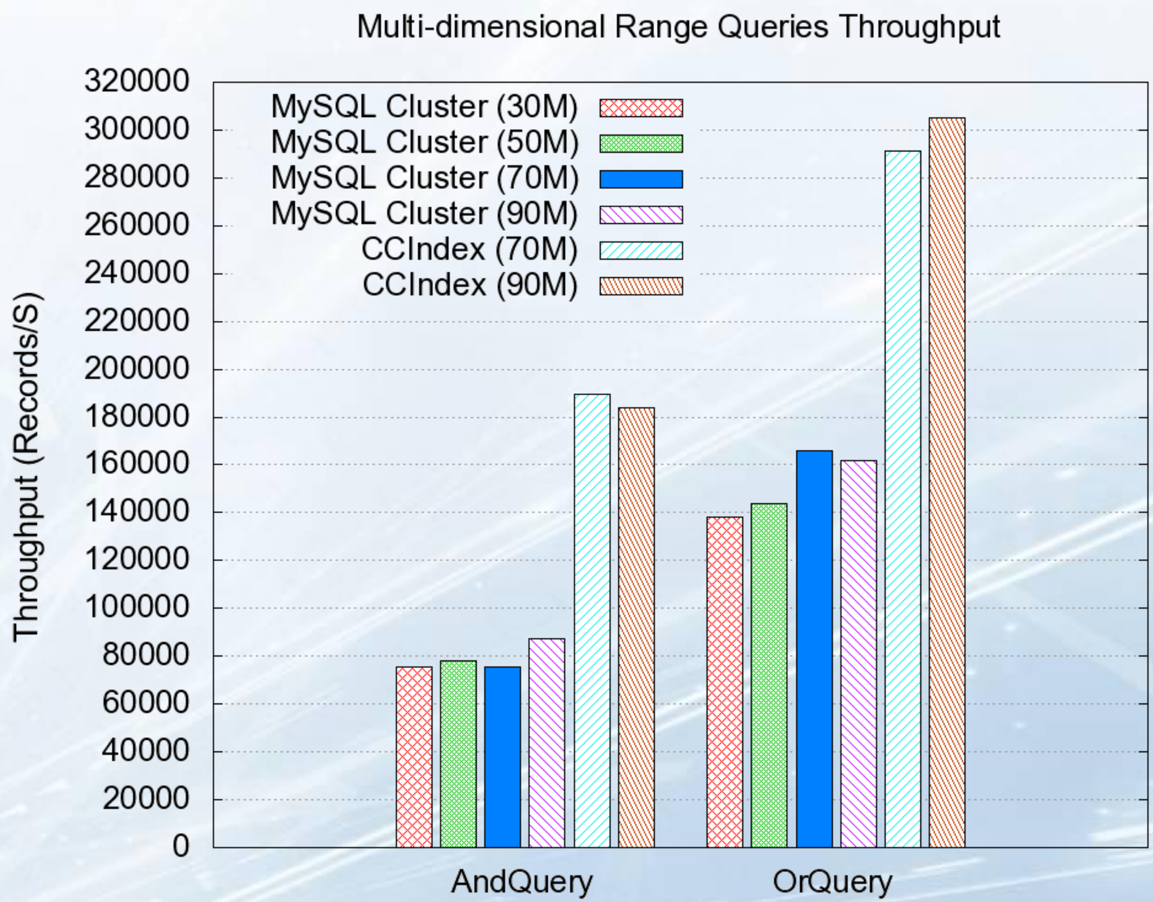


•Index scan is 11.4 times of IndexedTable

•Random read and write is 54.9% and 121.4% faster than IndexedTable

Evaluations: Synthetic Application benchmarks

- Env: 16 nodes
  - Dual CPU, 4 cores, 1.8 GHz AMD Opteron，6 GB memory, 186GB RAID1 SCSI disk, Gigabits Ethernet
- Workload
  - 120 million Nagios monitoring records, average length 118 bytes, 2 indexes



With the 90 million records, CCIndex AndQuery and OrQuery throughput is 2.1 and 1.9 times of the memory-based parallel database MySQL Cluster.

Conclusions

- We proposes a scheme CCIndex to support Multi-dimensional Range Queries in DOTs
- CCIndex consumes 5.3 ~ 29.3% more storage, and has same data recovery ability
- CCIndex throughput is about 11.4 times of secondary index in Apache HBase.
- The synthetic benchmarks in a 16-node cluster shows that CCIndex AndQuery and OrQuery throughput is 2.1 and 1.9 times of MySQL Cluster with 90 million records dataset.

History of HiC

- HiC is NOT an pure academic conference
- HiC is a opportunity to exhibit your Great idea on Hadoop
- HiC is a active stage for Hadoop grass-roots
  - Founded by Hadoop engineers on 11/23/2008
    - 1st Hadoop Salon (2008-11-23), 60 attendees
    - 2nd Hadoop Salon (2009-05-09), 120 attendees
    - Hadoop in China 2009 (2009-11-15), 300 attendees
    - Hadoop in China 2010 (2010-9-4), 600 attendees
  - Covers Development, Application, Research and Tutorial around Hadoop technologies
  - Goals: Communicate, Understanding and Practice

HiC Highlights

HiC 2009

HiC 2010

- Thanks!