Managing Data

Introduction

- Relational database
  - Default data storage and retrieval mechanism since 80s
  - Efficient in: transaction processing
  - Example: System R, Ingres, etc.
  - Replaced hierarchical and network databases
- For scalable web search service:
  - Google File System (GFS)
    - Massively parallel and fault tolerant distributed file system
  - BigTable
    - Organizes data
    - Similar to column-oriented databases (For e.g. Vertica)
  - MapReduce
    - Parallel programming paradigm

Introduction

- Suitable for:
  - Large volume massively parallel text processing
  - Enterprise analytics
- Similar to BigTable data model are:
  - Google App Engine's **Datastore**
  - Amazon's **SimpleDB**

Relational Databases

- Users/application programs interact with an RDBMS through SQL
- RDBM parser:
  - Transforms queries into memory and disk-level operations
  - Optimizes execution time
- Disk-space management layer:
  - Stores data records on pages of contiguous memory blocks
  - Pages are fetched from disk into memory as requested using pre-fetching and page replacement policies

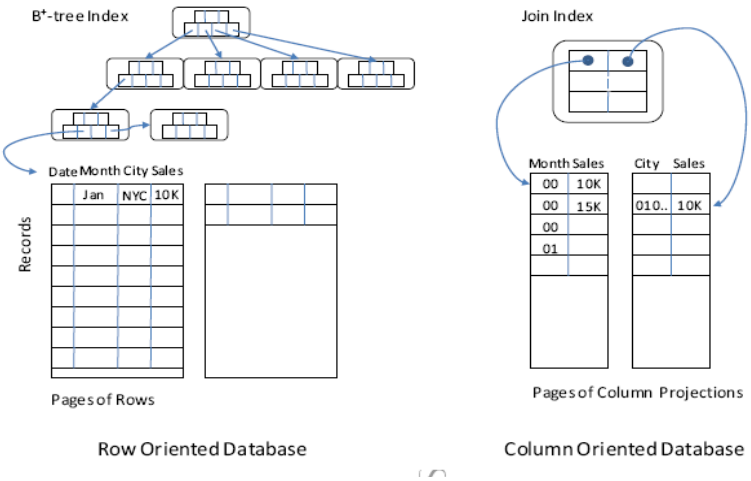
Relational Databases Contd...

- Database file system layer:
  - Independent of OS file system
  - Reason:
    - To have full control on retaining or releasing a page in memory
    - Files used by the DB may span multiple disks to handle large storage
  - Uses parallel I/O systems, viz. RAID disk arrays or multi-processor clusters

Data Storage Techniques

- Row-oriented storage
  - Optimal for write-oriented operations viz. transaction processing applications
  - Relational records: stored on contiguous disk pages
  - Accessed through indexes (primary index) on specified columns
  - Example: B^+- tree like storage
- Column-oriented storage
  - Efficient for data-warehouse workloads
    - Aggregation of **measure** columns need to be performed based on values from **dimension** columns
    - Projection of a table is stored as sorted by dimension values
    - Require multiple "join indexes"
      - If different projections are to be indexed in sorted order

Data Storage Techniques Contd...

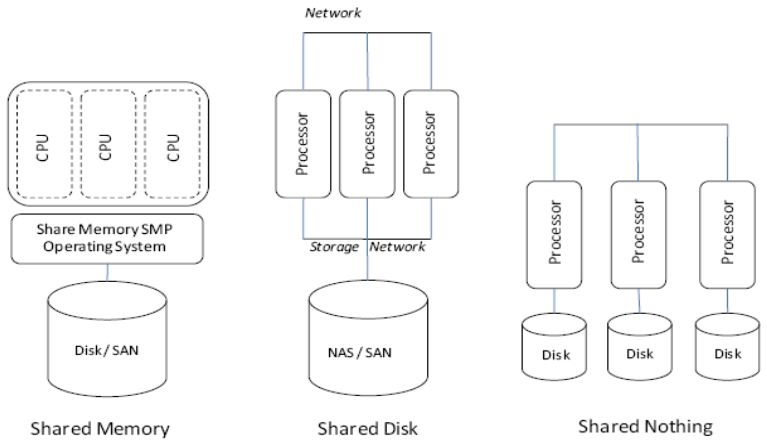


Source: "Enterprise Cloud Computing" by Gautam Shroff

Parallel Database Architectures

- Shared memory
  - Suitable for servers with multiple CPUs
  - Memory address space is shared and managed by a symmetric multi-processing (SMP) operating system
  - SMP:
    - Schedules processes in parallel exploiting all the processors
- Shared nothing
  - Cluster of independent servers each with its own disk space
  - Connected by a network
- Shared disk
  - Hybrid architecture
  - Independent server clusters share storage through high-speed network storage viz. NAS (network attached storage) or SAN (storage area network)
  - Clusters are connected to storage via: standard Ethernet, or faster Fiber Channel or Infiniband connections

Parallel Database Architectures contd...



Source: "Enterprise Cloud Computing" by Gautam Shroff

9

3/13/2023

Advantages of Parallel DB over Relational DB

- Efficient execution of SQL queries by exploiting multiple processors
- For **shared nothing** architecture:
  - Tables are partitioned and distributed across multiple processing nodes
  - SQL optimizer handles distributed joins
- Distributed **two-phase commit** locking for transaction isolation between processors
- Fault tolerant
  - System failures handled by transferring control to "stand-by" system [for transaction processing]
  - Restoring computations [for data warehousing applications]

10

3/13/2023

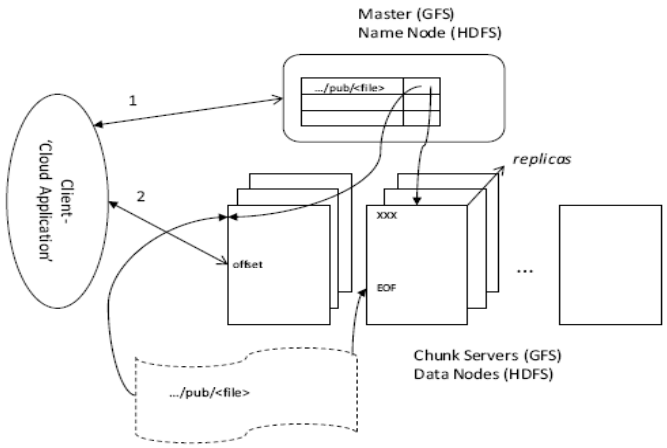
Advantages of Parallel DB over Relational DB

- Examples of databases capable of handling parallel processing:
  - Traditional transaction processing databases: **Oracle, DB2, SQL Server**
  - Data warehousing databases: **Netezza, Vertica, Teradata**

Cloud File Systems

- Google File System (GFS)
  - Designed to manage relatively large files using a very large distributed cluster of commodity servers connected by a high-speed network
  - Handles:
    - Failures even during reading or writing of individual files
    - Fault tolerant: a necessity
      - p(system failure) = 1-(1-p(component failure))^N → 1 (for large N)
    - Support parallel reads, writes and appends by multiple simultaneous client programs
- Hadoop Distributed File System (HDFS)
  - Open source implementation of GFS architecture
  - Available on Amazon EC2 cloud platform

GFS Architecture



Source: "Enterprise Cloud Computing" by Gautam Shroff

GFS Architecture Contd...

- Single master controls file namespace
- Large files are broken up into **chunks** (GFS) or **blocks** (HDFS)
- Typical size of each chunk: 64 MB
  - Stored on commodity (Linux) servers called **Chunk servers** (GFS) or **Data nodes** (HDFS)
  - Replicated **three** times on different:
    - Physical rack
    - Network segment

Read Operation in GFS

- Client program sends the full path and offset of a file to the **Master** (GFS) or **Name Node** (HDFS)
- Master replies with meta-data for **one** of replicas of the chunk where this data is found.
- Client caches the meta-data for faster access
- It reads data from the designated chunk server

Write/Append Operation in GFS

- Client program sends the full path of a file to the **Master** (GFS) or **Name Node** (HDFS)
- Master replies with meta-data for **all** of replicas of the chunk where this data is found.
- Client send data to be appended to all chunk servers
- Chunk server acknowledge the receipt of this data
- Master designates one of these chunk servers as **primary**
- Primary chunk server appends its copy of data into the chunk by choosing an offset
  - Appending can also be done beyond **EOF** to account for multiple simultaneous writers
- Sends the offset to each replica
- If all replicas do not succeed in writing at the designated offset, the primary retries

Fault Tolerance in GFS

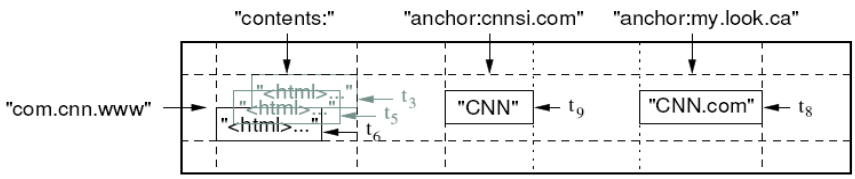
- Master maintains regular communication with chunk servers
  - Heartbeat messages
- In case of failures:
  - Chunk server's meta-data is updated to reflect failure
  - For failure of primary chunk server, the master assigns a new primary
  - Clients occasionally will try to this failed chunk server
    - Update their meta-data from master and retry

17

3/13/2023

BigTable

- Distributed structured storage system built on GFS
- Sparse, persistent, multi-dimensional sorted map (**key-value pairs**)
- Data is accessed by:
  - Row key
  - Column key
  - Timestamp



18

3/13/2023

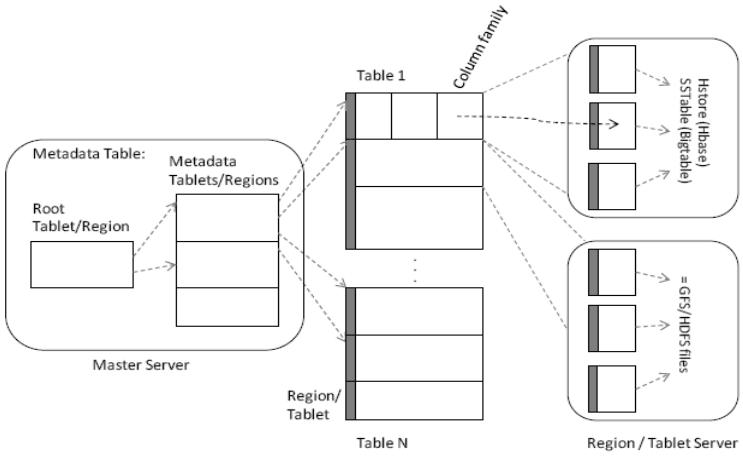
BigTable Contd...

- Each column can store arbitrary **name-value** pairs in the form: ***column-family : label***
- The set of possible column-families for a table is fixed when it is created
- Labels within a column family can be created dynamically and at any time
- Each BigTable cell (row, column) can store multiple versions of the data in decreasing order of timestamp
  - As data in each column is stored together, they can be accessed efficiently

19

3/13/2023

BigTable Storage



Source: "Enterprise Cloud Computing" by Gautam Shroff

20

3/13/2023

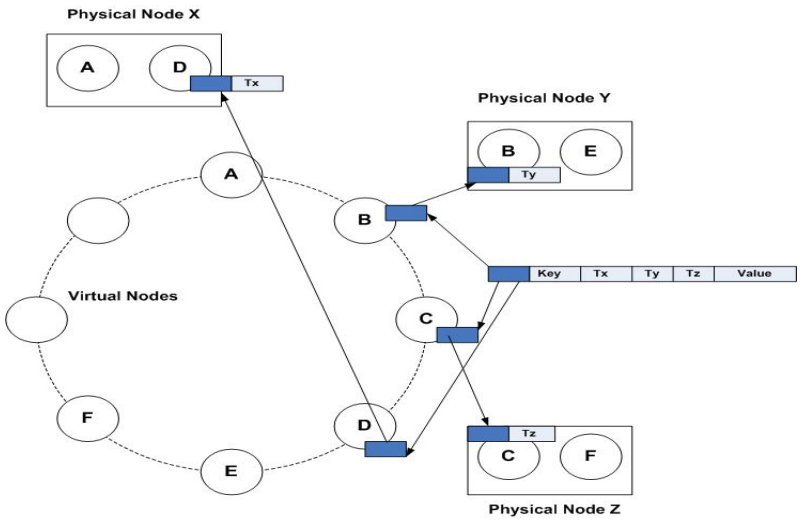
BigTable Storage Contd...

- Each table is split into different row ranges, called **tablets**
- Each tablet is managed by a **tablet server:**
  - Stores each column family for a given row range in a separate distributed file, called **SSTable**
- A single meta-data table is managed by a **Meta-data server**
  - Locates the tablets of any user table in response to a read/write request
- The meta-data itself can be very large:
  - Meta-data table can be similarly split into multiple tablets
  - A **root tablet** points to other meta-data tablets
- Supports large parallel reads and inserts even simultaneously on the same table
- Insertions done in sorted fashion, and requires more work can simple append

Dynamo

- Developed by Amazon
- Supports large volume of concurrent updates, each of which could be small in size
  - Different from BigTable: supports bulk reads and writes
- Data model for Dynamo:
  - Simple <key, value> pair
  - Well-suited for Web-based e-commerce applications
  - Not dependent on any underlying distributed file system (for e.g. GFS/HDFS) for:
    - Failure handling
      - Data replication
      - Forwarding write requests to other replicas if the intended one is down
    - Conflict resolution

Dynamo Architecture



Dynamo Architecture Contd...

- Objects: <Key, Value> pairs with arbitrary arrays of bytes
- MD5: generates a 128-bit hash value
- Range of this hash function is mapped to a **set of virtual nodes** arranged in a ring
  - Each key gets mapped to one virtual node
- The object is replicated at a **primary** virtual node as well as (N – 1) additional virtual nodes
  - N: number of physical nodes
- Each physical node (server) manages a number of virtual nodes at distributed positions on the ring

Dynamo Architecture Contd...

- Load balancing for:
  - Transient failures
  - Network partition
- Write request on an object:
  - Executed at one of its virtual nodes
  - Forwards the request to **all** modes which have the replicas of the object
  - **Quorum protocol**: maintains eventual consistency of the replicas when a large number of concurrent reads & writes take place

Dynamo Architecture Contd...

- Distributed object versioning
  - Write creates a new version of an object with its local timestamp incremented
  - Timestamp:
    - Captures history of updates
    - Versions that are superseded by later versions (having larger vector timestamp) are discarded
    - If multiple write operations on same object occurs at the same time, all versions will be maintained and returned to read requests
    - If conflict occurs:
      - Resolution done by application-independent logic

Dynamo Architecture Contd...

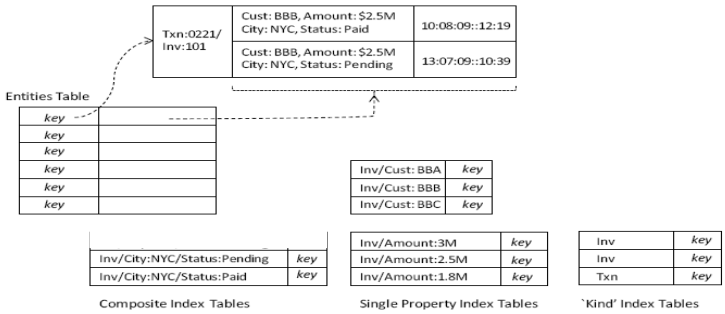
- **Quorum consistent**:
  - Read operation accesses **R** replicas
  - Write operation access **W** replicas
    - If **(R + W) > N** : system is said to be **quorum consistent**
  - Overheads:
    - For efficient write: larger number of replicas to be read
    - For efficient read: larger number of replicas to be written into
- Dynamo:
  - Implemented by different storage engines at node level: Berkley DB (used by Amazon), MySQL, etc.

27

3/13/2023

Datastore

- Offers simple transactional <Key, Value> pair database stores
- All entities (objects) in Datastore reside in one BigTable table
  - Does not exploit column-oriented storage
- **Entities table:** store data as **one** column family



Source: "Enterprise Cloud Computing" by Gautam Shroff

28

3/13/2023

Datastore

Contd...

- Multiple index tables are used to support efficient queries
- BigTable:
  - Horizontally partitioned (also called **sharded**) across disks
  - Sorted lexicographically by the key values
- Beside lexicographic sorting Datastore enables:
  - Efficient execution of **prefix** and **range** queries on key values
- Entities are 'grouped' for transaction purpose
  - Keys are lexicographic by group ancestry
    - Entities in the same group: stored close together on disk
- **Index tables:** support a variety of queries
  - Uses values of entity attributes as keys

29

3/13/2023

Datastore Contd...

- Automatically created indexes:
  - Single-Property indexes
    - Supports efficient lookup of the records with **WHERE** clause
  - 'Kind' indexes
    - Supports efficient lookup of queries of form **SELECT ALL**
- Configurable indexes
  - Composite index:
    - Retrieves more complex queries
- Query execution
  - Indexes with highest selectivity is chosen

30

3/13/2023

THANK YOU