VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

Cloud Data Models

Lecturer* : *Dr. Pavle Mogin

SWEN 432
Advanced Database Design and Implementation

Plan for Cloud Data Models

- NoSQL data models:
  - Key-Value data model,
  - Document data model, and
  - Column families data model
- Data storage techniques
  - Serializing Logical Structures on Disk

  - ***Readings:*** Have a look *at Useful Links at the Course Home Page*

Data Models

- A data model is a mathematical abstraction that is used to design a database of a real system
- The structure of the database depends on the underlying paradigm of the data model
- A data model provides languages:
  - To define the database structure, and
  - To query a database

Cloud Database Models

- Each DBMS relies on a data model
- Cloud DBMS fall into one of the following categories:
  - NoSQL, and
  - Relational
- NoSQL DBMSs themselves are built using one of the following data models:
  - Key-Value data model,
  - Document data model,
  - Column (families) data model, and
  - Graph data model
- Physical storage and data accessing techniques form an important part of NoSQL data models
  - Since these are important tools for achieving high throughput

Key-Value Data Model (Structure)

- The data structure of the key – value data model is based on a map or dictionary as found in many OO languages
- Each object of a database is represented by a unique key and a value, representing the relevant data
- There is no schema that stored objects have to comply with
  - So, value part of an object is stored without any semantic or structural information
  - The value data type is BLOB, or string and it may contain any kind of data as an image, or a file
- Also, there is no restriction on objects that can be stored together in a database

Key-Value Data Model (Operations)

- Standard operations are:
  - Read - get(key), and
  - Write – put(key, value)
- Additionally, there may also be a delete operation available
- Objects are retrieved from the database by solely using key values
- Key values are hashed to find the identifier of a network node where the object should be stored

Key-Value Data Model (Evaluation)

- Databases relying on the key-value data model
  - Have a very simple data structure
  - Use two (or three) simple basic operations, only and
  - Use hashing for retrieving data
- All that makes it possible to deploy databases that:
  - Have high throughput,
  - Are scalable,
  - Highly available, network partition tolerant, and eventually consistent
- Key-value database systems are best suited for simple applications needing only to look up objects
- Examples of popular key – value cloud DBMSs are:
  - Amazon's Dynamo,
  - Voldemort, and
  - Tokyo Cabinet

Document Data Model (Structure) (1)

- The document data model represents a logical progression from the key – value data model
- A document database is a (key, value) pair, where:
  - The key is a (unique) database name, and
  - The value is the database itself, represented as a set of collections
- A collection is a (key, value) pair, where
  - The key is a unique (user defined) collection name, and
  - The value is a set of documents
- A document is a (key, value) pair, where
  - The key is a unique (possibly system defined) document identifier, and
  - The value is a set of fields
- A field is a (key, value) pair, where
  - The key is a unique field name, and
  - The value is a scalar data type (including strings), or an array of them

Document Data Model (Structure) (2)

- Documents can be nested, or referenced from within documents by a defined identifier
- Documents of a collection do not adhere to any common predefined schema
  - Documents are self describing, since they contain both meta data and actual data
  - A collection may contain documents of completely different types
- Most often, the content of a document is represented in the JSON format, although it also can be an XML file

A Nested Document

{

type: “*CD*”

artist : “*Net King Cole*”

tracklist : [

{*track*: “*1*”, *title*: “*Fascination*” },

{*track*: “*2*”, *title*: “*Star Dust*” }

]

}

Document Data Model (Operations)

- The operations part of the data model offers a rich set of commands for:
  - Defining collections,
  - Inserting documents,
  - Updating fields,
  - Deleting documents,
  - Querying documents, and
  - Doing aggregation operations

Document Data Model (Evaluation) (1)

- The goal of developing the document data model was to fill gap between simple, but highly available, scalable, fast, partition tolerant, and eventually consistent key-value database systems and traditional, feature rich relational ones
- Since the document model is basically a key-value data model, the potential for availability, scalability, and partition tolerance has been retained

Document Data Model (Evaluation) (2)

- Compared to the key/value data model:
  - The document as a basic unit of access control with its hierarchical internal schema gives raise to:
    - Fast fetching the amount of data needed to answer queries of a medium complexity
    - A richer semantics of stored data,
    - A much more powerful query language,
    - Avoiding need for some joins in generating query answers (thanks to document embedding), and
    - A potential for performing joins within a user application using referencing
- The two popular document CDBMS are:
  - CouchDB and
  - MongoDB

Column Family Data Model – General

- Column family data model:
  - Represents a logical progression of the key – value data model where the value part has a recognizable structure
  - Tables are used for the logical (visual) data representation
  - Table rows contain sparsely populated ***columns*** of data,
  - Each set of logically related columns forms a ***column family*** that is:
    - A unit of physical data access control,
    - Stored contiguously on a separate location on disk using large disk pages to promote for fast reads by amortizing disk head seeks, and thus promote efficient analytical (statistical) processing
  - Columns are nullable and may be ***run time defined***
  - ***Row keys*** (record ID's) are used for data partitioning,
  - Users are encouraged to reason about physical storage properties of their data (where and how their data has been stored)
  - Either timestamps or version vectors are used for versioning

An Example (A Relational Table)

- Consider a relational table called *person* having many hundreds of millions of rows:
  - Each row is stored contiguously as a whole,
  - It is very good for retrieving partial or whole rows based on *personId*,
  - It is very slow for answering queries : “How many men are born in 1950?”

person

personId	name	surname	address	year_of_birth	gender
1	James	Bond	London	1950	M
2	July	Andrews	New York	1970	F
...	...	...	...	...	...
999999	Tommy	Robredo	Madrid	1950	M
...	...	...	...	...	...

An Example (Column Families)

- Column family cloud databases group columns of data needed to satisfy a query into *Column Families*
  - The *person* table would be subdivided into, say *personal_data* and *demographic* column families
  - The *demographic* column family would be stored separately and retrieved in a few disk accesses as a whole

person

row key	personal _data			demographic	
	name	surname	address	year_of_birth	gender
1	James	Bond	London	1950	M
2	July	Andrews	New York	1970	F
...	...	...	...	...	...
999999	Tommy	Robredo	Madrid	1950	M
...	...	...	...	...	...

Column Family Operations

- The API's of column family CDBMS provide functions for creating and deleting:
  - Column families (tables) and
  - Indexes
- Client applications can:
  - Insert rows
  - Write column values in a table (updates and deletes are accomplished by writes),
  - Look up and select column values from individual rows,
  - Iterate over a subset of data in a table,
  - Select column data based on their timestamp
  - Perform aggregation

Serializing Logical Data Structures

- Serializing logical data structures on a disk determine how the disk is accessed and thus directly implicate performance
- There are two main serializing approaches:
  - Row based storage layout and
  - Colum based storage layout
- A running example:

StudentId	CourseId	Year	Trimester	Grade
7007	COMP102	2012	1	A+
3993	SWEN304	2013	2	B-
5555	SWEN432	2014	1	C+

Row Based Storage Layout

- A relational table gets serialized by appending its rows and flushing to disk

7007	COMP102	2012	1	A+	3993	SWEN304	2013
2	B-	5555	SWEN432	2014	1	C+	

- Advantage:
  - Fast retrieval of whole records
- Disadvantage:
  - Operations on columns are slow since it may require reading all of the data

Columnar Storage Layout

- Columnar storage layout serializes tables by
  - Appending whole columns and flushing them to disk

7007	3993	5555	COMP102	SWEN304	SWEN432	2012	2013
2014	1	2	1	A+	B-	C+	

  - Or even storing columns separately

7007	3993	5555
COMP102	SWEN304	SWEN432
2012	2013	2014
1	2	1
A+	B-	C+

Advantage:
- Fast retrieval of whole columns

Disadvantage:
- Retrieval of records is slow since it may require reading all of the data

- Column families combine row based and columnar storage layouts by storing column families separately

Summary

- Key-value data model:
  - Each data object is a (key, value) pair, where value is opaque
  - DDL and DML languages are of a limited expressive power
- Document data model:
  - A database is a set of collections
  - A collection is a set of fields
  - Embedding and referencing used to implement relationships between documents
  - DDL and DML of extensive expressive power
- Column family data model:
  - Logical view: tables
  - Each column family serves to satisfy one or a few predefined queries
  - Each updatable piece of data has either a timestamp or a version vector