NoSQL
Columnar Databases

Problem Set #5 is due tonight
Problem Set #6 will be online tonight, or you all get 100.

Agenda

- Data Model
  - Column families, super columns, two points of view
- Column-family Stores
  - Google BigTable, Cassandra, HBase
- Cassandra as an Example
  - Cassandra data model 1.0 vs. 2.0
  - Cassandra Query Language (CQL)
  - Data partitioning, replication
  - Local Data Persistence
  - Query processing, Indexes, Lightweight Transactions

Column-family Stores: Basics

- AKA: wide-column, columnar
  - not to confuse with column-oriented RDBMS
- Data model: rows that have many columns associated with a row key
- Column families are groups of related data (columns) that are often accessed together
  - e.g., for a customer we typically access all profile information at the same time, but not customer's orders

Data Model

- *Columns* within Rows = the basic data item
  - a 3-tuple consisting of
    - column **name**
    - **value**
    - **timestamp**

  - Can be modeled as follows


```
{ name: "firstName",
  value: "Martin",
  timestamp: 12345667890 }
```

- In the following, we will ignore the timestamp

Data Model

- *Row*: a collection of columns with a common row key
  - Columns can be added to any row at any time
    - without having to add it to other rows

```
// row
"martin-fowler" : {
    firstName: "Martin",
    lastName: "Fowler",
    location: "Boston"
}
```

Row key

Column keys

Row key1	Column Key1	Column Key2	Column Key3	...
	Column Value1	Column Value2	Column Value3	
⋮				

Data Model: Column Family

- **CF =** Set of columns containing "related" data

user_id (*row key*)	*column key*	*column key*	...
	column value	*column value*	...
1	login	first_name	...
	gonzo	Sam	...
4	login	age	...
	david	35	...
5	first_name	last_name	...
	Kathy	Wright	...
...			

Data Model: Column Family (2)

- Column family - example as JSON

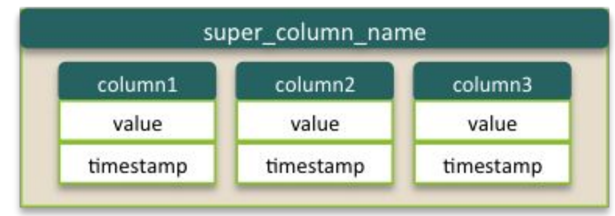
```
{ // row (columns from a CF)
  "pramod-sadalage" : {
    firstName: "Pramod",
    lastName: "Sadalage",
    lastVisit: "2012/12/12"
}
```

```
// row (cols from the same CF)
  "martin-fowler" : {
    firstName: "Martin",
    lastName: "Fowler",
    location: "Boston",
    activ: "true" }
}
```


source: Sadalage & Fowler: NoSQL Distilled, 2012

Data Model: Super Column Family

- **Super column**
  - A column whose value is composed of a map of columns
  - Used in some column-family stores (Cassandra 1.0)



- **Super column family**
  - A column family consisting of super columns

	Super Column key1			Super Column key2			
Row key1	Subcolumn Key1	Subcolumn Key2	…	Subcolumn Key3	Subcolumn Key4	…	…
	Column Value1	Column Value2		Column Value3	Column Value4		
⋮							

Super Column Family: Example

user_id (*row key*)	*super column key*			*super column key*			…
	subcolumn key	*subcolumn key*	...	*subcolumn key*	*subcolumn key*	...	...
	subcolumn value	*subcolumn value*	...	*subcolumn value*	*subcolumn value*	...	
1	home_address			work_address			
	city	street	...	city	street	...	
	Raleigh	Hillsborough St	...	Chapel Hill	Raleigh St	...	
4	home_address			temporary_address			
	city	street	...	city	street		
	Durham	Chapel Hill St	...	Chapel Hill	Raleigh St		
...							

Super Column Family in JSON

```
{ // row
  ″Cathy″: {
    ″username″: { ″firstname″: ″Cathy″, ″lastname″: ″Qi″ },
    ″address″: { ″city″: ″New York″, ″zip″: ″10001″ }
    }
  // row
  ″Terry″: {
    ″username″: { ″firstname″: ″Terry″, ″lastname″: ″Martin″ },
    ″account″: { ″bank″: ″Citi″, ″account″: 12346789 },
    ″background″: { ″birthdate″: ″1990-03-04″ }
   }
 }
```

Column Family Stores: Features

- Data model: Column families
- System architecture
  - Data partitioning
- Local persistence
  - update log, memory, disk...
- Data replication
  - balancing of the data
- Query processing
  - query language
- Indexes

Representatives

HYPERTABLE

BigTable

- Google's paper:
  - Chang, F. et al. (2008). Bigtable: A Distributed Storage System for Structured Data. ACM TOCS, 26(2), pp 1–26.
- Proprietary, not distributed outside Google
  - used in Google Cloud Platform
- Data model: column families as defined above
  - *"A table in Bigtable is a sparse, distributed, persistent multidimensional sorted map. "*

```
(row:string, column:string, time:int64) →
string
```

BigTable: Example

- "BigTable = sparse, distributed, persistent, multi-dimensional sorted map indexed by *(row_key, column_key, timestamp)*"

column names

"contents:html" "param:lang" "param:enc" "a:cnnsi.com" "a:ihned.cz"

t_2 t_6 t_8

"com.ccn.www"

<html>...
<html>...
<html>...

t_2 EN t_2 UTF-8

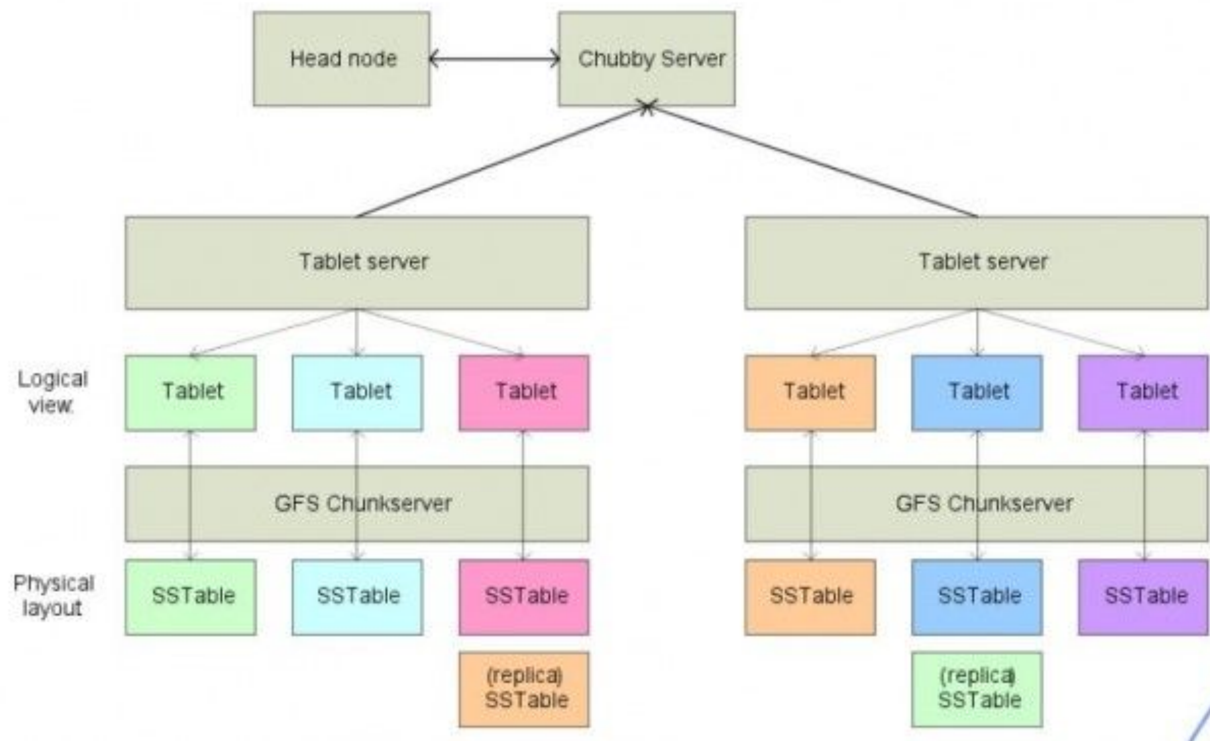
t_3 CNN.com t_7 CNN

column *column* *column* *column* *column*

column family

row key *row*

BigTable: Architecture



Cassandra

- Developed at Facebook
  - now, Apache Software License 2.0
- Initial release: 2008
- Written in: Java
- OS: cross-platform
- Operations:
  - CQL (Cassandra Query Language)
  - MapReduce support (can cooperate with Hadoop)

http://cassandra.apache.org/

Cassandra: Data Model

- Column families, super column families
  - Can define metadata about columns
  - Now denoted as: Thrift API
- **Static** – similar to a relational database table
  - Rows have the same limited set of columns
  - However, rows are not required to have all columns
- **Dynamic** – takes advantage of Cassandra's ability to use arbitrary *new* column names

Cassandra: Data Model

row key	columns...								
Alex Smith	2010	2011	2012	2013	2014	2015	2016	2017	2018
	49ers	49ers	chiefs	chiefs	chiefs	chiefs	chiefs	chiefs	redskins
Blaine Gabbert	2011	2012	2015	2016					
	jaguars	jaguars	49ers	49ers					
Colin Kaepernick	2012	2013	2014	2015	2016				
	49ers	49ers	49ers	49ers	49ers				

row key	columns...						
24911	first	last	college	dob	height	weight	2016
	Alex	Smith	Utah	1984-05-07	6-4	212	chiefs
27073	first	last	college	dob	high school		
	Blaine	Gabbert	Missouri	1989-10-15	Parkway West		
27154	first	last	college	height	weight		
	Colin	Kaepernick	Nevada	6-4	225		

Key Spaces

- Databases are called "KEYSPACES" in Cassandra
- They are created as follows:

```
CREATE KEYSPACE db
      WITH replication = {
             'class' :  'SimpleStrategy',
             'replication_factor': '2' }
       AND durable_writes = 'true';

DESCRIBE KEYSPACES;
```

Cassandra Tables

KEYSPACES *can* have TABLES defined as follows

```
CREATE TABLE Player (
    pid int PRIMARY KEY,
    first TEXT,
    last TEXT,
    college TEXT);

DESCRIBE TABLES;
```

Working with Tables

```
CREATE TABLE users (
    user_id int PRIMARY KEY,
    login text,
    name text,
    email text );

INSERT INTO users (user_id, login, name)
VALUES (3, 'cathyqi', 'Cathy Qi');

SELECT * FROM users;
 user_id | email |  login   | name
---------+-------+---------+-----------
       3 | null  | cathyqi | Cathy Qi
```

Cassandra: Column Families

- An alternative to tables are *COLUMN FAMILIES*
- Requires a Name and Comparators
  - A key *must* be specified
  - Data types for columns *can* be specified
  - Options *can* be specified

```
CREATE COLUMNFAMILY Fish (key blob PRIMARY KEY);
CREATE COLUMNFAMILY FastFoodPlaces (name text PRIMARY KEY)
    WITH comparator=timestamp AND default_validation=int;
CREATE COLUMNFAMILY MonkeyTypes (
    key uuid PRIMARY KEY,
    species text,
    alias text,
    population varint
) WITH comment='Important biological records'
  AND read_repair_chance = 1.0;
```

Cassandra: Column Families (2)

- **Comparator** = data type for a column name
- **Validator** = data type of a column value
  - or content of a row key
- Data types do not need to be defined
  - Default: BytesType, i.e. arbitrary hexadecimal bytes
- Basic operations: GET, SET, DEL

Cassandra: Data Manipulation

```
create column family users
    with key_validation_class = Int32Type
    and comparator = UTF8Type
    and default_validation_class = UTF8Type;


// set column values in row with key 7
set users[7]['login'] = utf8('cathyqi');
set users[7]['name'] = utf8('Cathy Qi');
set users[7]['email'] = utf8('qi@best.com');

set users[13]['login'] = utf8('fantom');
set users[13]['name'] = utf8('Un Known');
```

I. Holubová, J. Kosek, K. Minařík, D. Novák. Big Data a NoSQL databáze. Praha: Grada Publishing, 2015. 288 p.

Cassandra: Data Manipulation (2)

```
get users[7]['login'];
=> (name=login, value=cathyqi, timestamp=1429268223462000)

get users[13];
=> (name=login, value=fantom, timestamp=1429268224554000)
=> (name=name, value=Un Known, timestamp=1429268224555000)


list users;
RowKey: 7
=> (name=email, value=qi@best.com, timestamp=14292682...)
=> (name=login, value=cathyqi, timestamp=1429268223462000)
=> (name=name, value=Cathy Qi, timestamp=1429268223471000)
-------------------
RowKey: 13
=> (name=login, value=fantom, timestamp=1429268224554000)
=> (name=name, value=Un Known, timestamp=1429268225231000)
```

Cassandra: Sparse Tables

- CQL: Cassandra Query Language
  - SQL-like commands
    - CREATE, ALTER, UPDATE, DROP, DELETE, TRUNCATE, INSERT, …
  - Simpler than SQL
- Since CQL 3 (Cassandra 1.2)
  - Column -> cell
  - Column family -> table
- Dynamic columns (wide rows) still supported
  - CQL supports everything that was possible before
  - “Old” approach (Thrift API) can be used as well

Tables: Dynamic Columns

- Values can use "collection" types:
  - **set** – unordered unique values
  - **list** – ordered list of elements
  - **map** – name + value pairs
    - a way to realize super-columns

- Realization of the original idea of free columns
  - Internally, all values in collections as individual columns
  - Cassandra can well handle "unlimited" number of columns

Tables: Dynamic Columns (2)

```
CREATE TABLE users (
    login text PRIMARY KEY,
    name text,
    emails set<text>, // column of type “set”
    profile map<text, text> // column of type “map”
)

INSERT INTO users (login, name, emails, profile)
VALUES ( 'honza', 'Jan Novák', { 'honza@novak.cz' },
     { 'colorschema': 'green', 'design': 'simple' }
);

UPDATE users
SET emails = emails + { 'jn@firma.cz' }
WHERE login = 'honza';
```

Dynamic Columns: Another Way

- Compound primary key

```
CREATE TABLE mytable (
    row_id int, column_name text, column_value text,
    PRIMARY KEY (row_id, column_name)
);

INSERT INTO mytable (row_id, column_name, column_value)
VALUES ( 3, 'login', 'honza' );
INSERT INTO mytable (row_id, column_name, column_value)
VALUES ( 3, 'name', 'Jan Novák');
INSERT INTO mytable (row_id, column_name, column_value)
VALUES ( 3, 'email', 'honza@novak.cz');
```

Data Sharding in Columnar Systems

System	Terminology
BigTable	tablets
HBase	regions
Cassandra	partitions

	user_id (row key)	login	name
partition_1	1	cathyqi	Cathy Qi
	4	davidm	David Man
	...		
	1000	iamboo	Nota James
partition_2	1001	violet	Ziwei Chen
	1003	ernie	Bernard ...
	...		
	2000	peach	Joseph Ash
	...		

Data Sharding in Cassandra

- Entries in each table are split by partition key
  - Which is a selected column (or a set of columns)
  - Specifically, the first column (or columns) from the primary key is the partition key of the table

```
CREATE TABLE tab ( a int, b text, c text, d text,
    PRIMARY KEY ( a, b, c)
);
```

```
CREATE TABLE tab ( a int, b text, c text, d text,
    PRIMARY KEY ( (a, b), c)
);
```

Data Sharding in Cassandra (2)

- All entries with the same partition key
  - Will be stored on the same physical node
  - => efficient processing of queries on one partition key

```
CREATE TABLE mytable (
    row_id int, column_name text, column_value text,
    PRIMARY KEY (row_id, column_name) );
```

- The rest of the columns in the primary key
  Are so called clustering columns
  - Rows are locally sorted by values in the clustering columns
    - the order for physical storing rows

Data Replication

- Cassandra adopts peer-to-peer replication
  - The same principles like in key-value stores & document DB
  - Read/Write quora to balance between availability and consistency guarantees

- Google BigTable
  - Physical data distribution & replication is done by the underlying distributed file system
  - GFS

Cassandra Query Language (CQL)

- The syntax of CQL is similar to SQL
  - But search just in one table (no joins)

```
SELECT <selectExpr>
FROM [<keyspace>.]<table>
[WHERE <clause>]
[ORDER BY <clustering_colname> [DESC]]
[LIMIT m];
```

```
SELECT column_name, column_value
FROM mytable
WHERE row_id=3
ORDER BY column_value;
```

CQL: Limitations on "Where" Part

```
CREATE TABLE mytable (
   row_id int,
   column_name text,
   column_value text,
   PRIMARY KEY
       (row_id, column_name)
);
```

- The search condition can be:
  - on columns in the partition key
    - And only using operators == and IN

```
... WHERE row_id IN (3, 4, 5)
```

    - Therefore, the query hits only one or several physical nodes (not all)

  - on columns from the clustering key
    - Especially, if there is also condition on the partitioning key

```
... WHERE row_id=3 AND column_name='login'
```

    - If it is not, the system must filter all entries

```
SELECT * FROM mytable
WHERE column_name IN ('login', 'name') ALLOW FILTERING;
```

CQL: Limitations on "Where" Part (1)

- Other columns can be queried
  - If there is an index built on the column
- Indexes can be built also on collection columns (set, list, map)
  - And then queried by CONTAINS like this

```
SELECT login FROM users
    WHERE emails CONTAINS 'jn@firma.cz';
SELECT * FROM users
    WHERE profile CONTAINS KEY 'colorschema';
```

Indexes

- Secondary indexes on any column
  - B^+-Tree indexes
  - User-defined implementation of indexes

```
CREATE INDEX ON users (emails);
```

Transactions

- Cassandra 2.x supports "lightweight transactions"
  - compare and set operations
  - using Paxos consensus protocol
    - nodes agree on proposed data additions/modifications
    - faster than Two-phase commit protocol (P2C)

```
INSERT INTO users (login, name, emails)
VALUES ('cathyqi', 'Cathy Qi', { 'qi@best.com' })
IF NOT EXISTS;

UPDATE mytable SET column_value = 'qi@best.org'
WHERE row_id = 3 AND column_name = 'email'
IF column_value = 'qi@best.com';
```

Summary

- Column-family stores
  - are worth only for large data and large query throughput
  - two ways to see the data model:
    - large sparse tables or multidimensional (nested) maps
  - data distribution is via row key
    - analogue of document ID or key in document or key-value stores
  - efficient disk + memory local data storage
- Cassandra
  - CQL: structured after SQL, easy transition from RDBMS