Department of Computer Science
University of Cyprus

EPL646 – Advanced Topics in Databases

Lecture 17

Cloud Data Management VII (Column Stores and Intro to NewSQL)

Demetris Zeinalipour
http://www.cs.ucy.ac.cy/~dzeina/courses/epl646

Lecture Overview

- **(2003) Google GFS Paper (SOSP'03)**
  - **Objective:** Create a Google-scale Filesystem
  - Apache HDFS is GFS open-source implementation.
- **(2004) Google's Map-Reduce Paper (OSDI'04)**
  - **Objective:** Enable big-data analytics over non-tabular data (e.g., XML or text) … with the assistance of GFS.
  - Apache's MapReduce: An open-source implementation of the paper
- **(2006) Google BigTable Paper (OSDI'06)**
  - **Objective:** Enable big-data analytics over tabular data (i.e., tables)
  - (2008) Apache's Hbase: An open-source implementation of the paper
  - (2010): Facebook Messaging moves from Cassandra to HBase
- (2012) Google's F1 RDBMS (SIGMOD'12) & Spanner Storage Papers **(OSDI'12)**

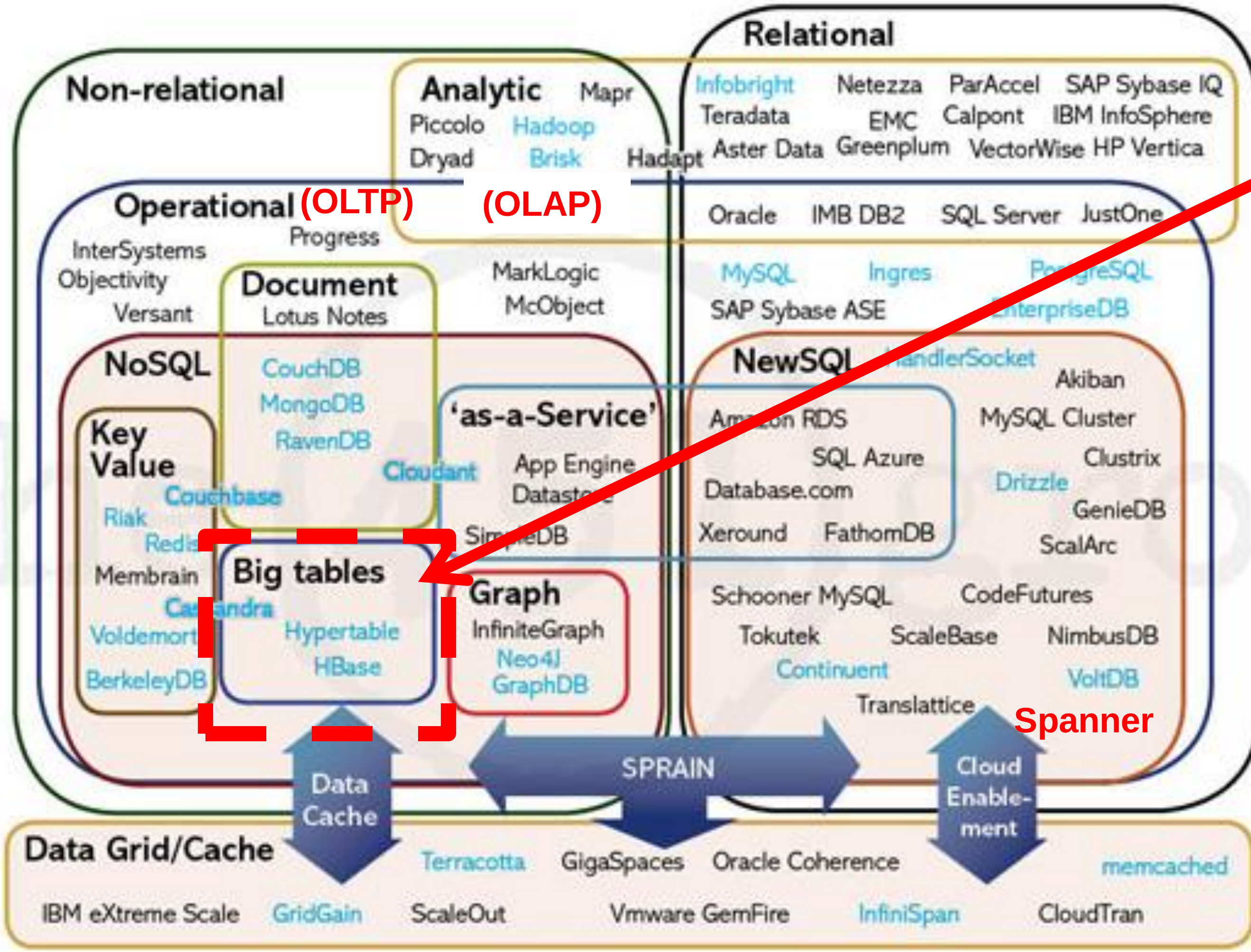
Today's Focus

HYPERTABLE INC APACHE HBASE Cassandra

EPL646: Part B

Distributed/Web/Cloud DBs/Dstores

Lecture Focus



Venn Diagram by 451 group

http://xeround.com/blog/2011/04/newsql-cloud-database-as-a-service

Column-oriented Databases

- A **column-oriented DBMS** is a database management system (DBMS) that **stores data tables** as sections of **columns** of data **rather than as rows** of data, like most relational DBMSs
- This has **advantages for** data warehouses, customer relationship management (CRM) systems, and library card catalogs, and other ad-hoc inquiry systems **where aggregates or scans are carried out over large numbers** of **similar data items**
- **MonetDB (CWI) pioneered this model but not for Cloud-scale scenarios (where Google did…)**

Row-Store **OLTP-workloads!**

1,Smith,Joe,40000;
2,Jones,Mary,50000;
3,Johnson,Cathy,44000;

Column-Store **OLAP-workloads!**

1,2,3;
Smith,Jones,Johnson;
Joe,Mary,Cathy;
40000,50000,44000;

Big-Tables
How Big are Big-Tables?

Bigtable: A Distributed Storage System for Structured Data
Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber

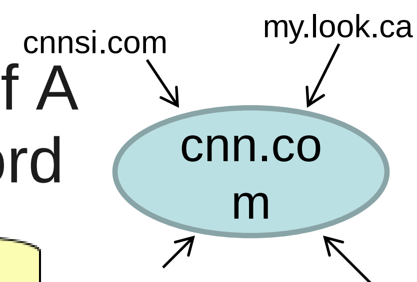
OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006.

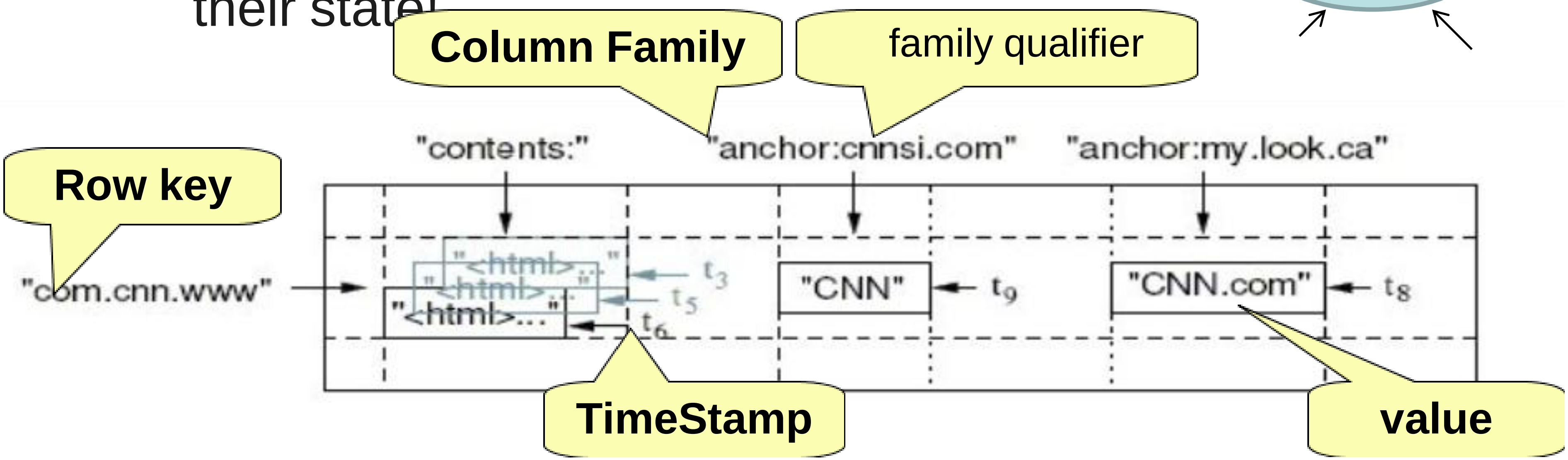
Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
Crawl	800	11%	1000	16	8	0%	No
Crawl	50	33%	200	2	2	0%	No
Google Analytics	20	29%	10	1	1	0%	Yes
Google Analytics	200	14%	80	1	1	0%	Yes
Google Base	2	31%	10	29	3	15%	Yes
Google Earth	0.5	64%	8	7	2	33%	Yes
Google Earth	70	–	9	8	3	0%	No
Orkut	9	–	0.9	8	5	1%	Yes
Personalized Search	4	47%	6	93	11	5%	Yes

Before and after Map-reduce processing of the given HTables !!!

Big Table Example 2 (Google Crawling)

- **Web crawlers** download the web by following links found inside **web pages** (starting from a seed).
- Google wants to know for each **URL A**, which other **URLs are linking A**. **Why?**
- To calculate the importance (Pagerank) of A
- Google's Crawlers use a Big-table to record their state!





BigTable Data Model (Conceptual View)

Table 5.1. Table `webtable`

Row Key	**Time Stamp**	**ColumnFamily `contents`**	**ColumnFamily `anchor`**
"com.cnn.www"	t9		`anchor:cnnsi.com` = "CNN"
"com.cnn.www"	t8		`anchor:my.look.ca` = "CNN.com"
"com.cnn.www"	t6	`contents:html` = "<html>..."	
"com.cnn.www"	t5	`contents:html` = "<html>..."	
"com.cnn.www"	t3	`contents:html` = "<html>..."	

All **column family** members are **stored together** on the filesystem. (see next slide)

It is **advised** that all **column family** members have the same general **access pattern** and **size** characteristics.

BigTable Data Model (Physical View)

Hbase stores column families physically close on disk

Table 5.2. ColumnFamily anchor

Row Key	Time Stamp	Column Family anchor
"com.cnn.www"	t9	anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8	anchor:my.look.ca = "CNN.com"

Table 5.3. ColumnFamily contents

Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	contents:html = "<html>..."
"com.cnn.www"	t5	contents:html = "<html>..."
"com.cnn.www"	t3	contents:html = "<html>..."

Empty Cells are not stored in hbase!

Apache HBase

Apache HBase™ is the **Hadoop database**, a **distributed**, **scalable**, **big data store**.

This project's goal is the hosting of very large tables -- **billions of rows** X **millions of columns** -- atop clusters of **commodity hardware**.

Just as **Bigtable** leverages the distributed data storage provided by the **Google File System**, **Apache HBase** provides **Bigtable-like** capabilities on top of **Hadoop** and **HDFS**.

Standalone / Distributed modes available like all other **Hadoop** projects we've seen so far.

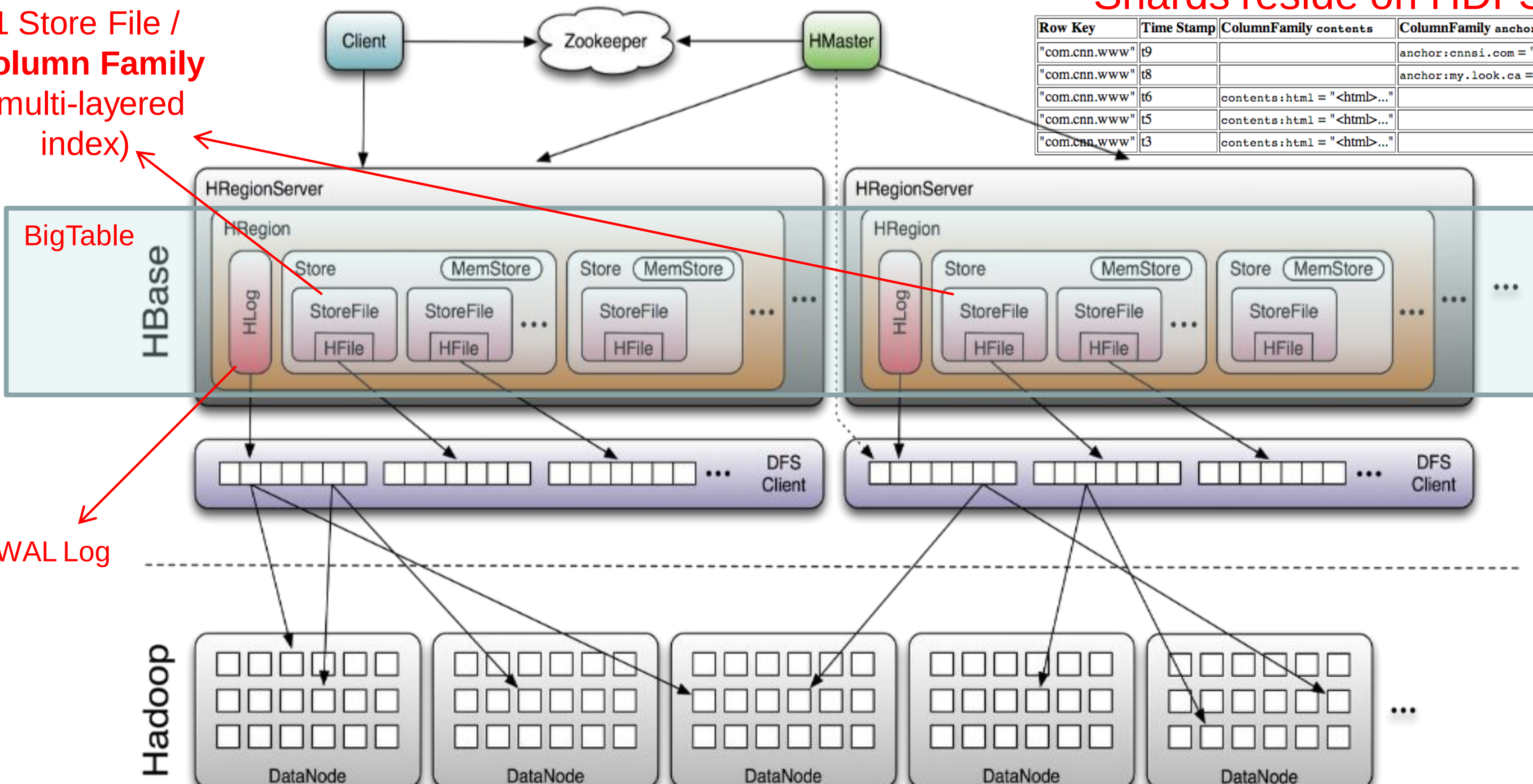
Apache HBase (Architecture)

Automated Sharding!
Shards reside on HDFS

1 Store File / **Column Family** (multi-layered index)

Table 5.1. Table webtable

Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor
"com.cnn.www"	t9		anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8		anchor:my.look.ca = "CNN.com"
"com.cnn.www"	t6	contents:html = "<html>..."	
"com.cnn.www"	t5	contents:html = "<html>..."	
"com.cnn.www"	t3	contents:html = "<html>..."	



HDFS View: Disk use: hadoop fs -du /hbase/myTable

```
/hbase
     /<Table>                (Tables in the cluster)
          /<Region>               (Regions for the table)
               /<ColumnFamiy>          (ColumnFamilies for the Region for the table)
                    /<StoreFile>            (StoreFiles for the ColumnFamily for the Regions for the table)
```

Apache HBase (Shell Interface)

Simply install HBase over Hadoop / HDFS

table — Column family

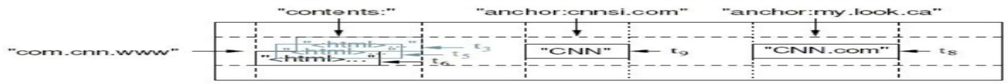
```
$ hbase shell
> create 'test', 'data'
0 row(s) in 4.3066 seconds
> list
test
1 row(s) in 0.1485 seconds
> put 'test', 'row1', 'data:1', 'value1'
0 row(s) in 0.0454 seconds
> put 'test', 'row2', 'data:2', 'value2'
0 row(s) in 0.0035 seconds
> put 'test', 'row3', 'data:3', 'value3'
0 row(s) in 0.0090 seconds
```

```
> scan 'test'
ROW COLUMN+CELL
row1 column=data:1, timestamp=1240148026198, value=value1
row2 column=data:2, timestamp=1240148040035, value=value2
row3 column=data:3, timestamp=1240148047497, value=value3
3 row(s) in 0.0825 seconds
> disable 'test'
09/04/19 06:40:13 INFO client.HBaseAdmin: Disabled test
0 row(s) in 6.0426 seconds
> drop 'test'
09/04/19 06:40:17 INFO client.HBaseAdmin: Deleted test
0 row(s) in 0.0210 seconds
> list
0 row(s) in 2.0645 seconds
```

Type: "help" to see all commands!

Apache HBase (Overview of Features)

- **Column families**: declared at schema definition time.
  - Additional Columns can be added on the fly while the table is up an running.



- **Get/Put/Delete** and **Scan Operations** only
  - Can be combined with Region-based Filters.
  - No built-in Joins (can be implemented with MR jobs)!

5.8.1.3. Versioned Get Example

The following Get will return the last 3 versions of the row.

```
Get get = new Get(Bytes.toBytes("row1"));
get.setMaxVersions(3);  // will return last 3 versions of row
Result r = htable.get(get);
byte[] b = r.getValue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
List<KeyValue> kv = r.getColumn(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns all versions of this column
```

- Supports **constraints** (e.g., range).
- **Row Locks** supported but deprecated
  - might lock whole Regionserver.
- Catalog tables **-ROOT-** and **.META.** exist as HBase tables (i.e., not on Master but on RegionServers!)

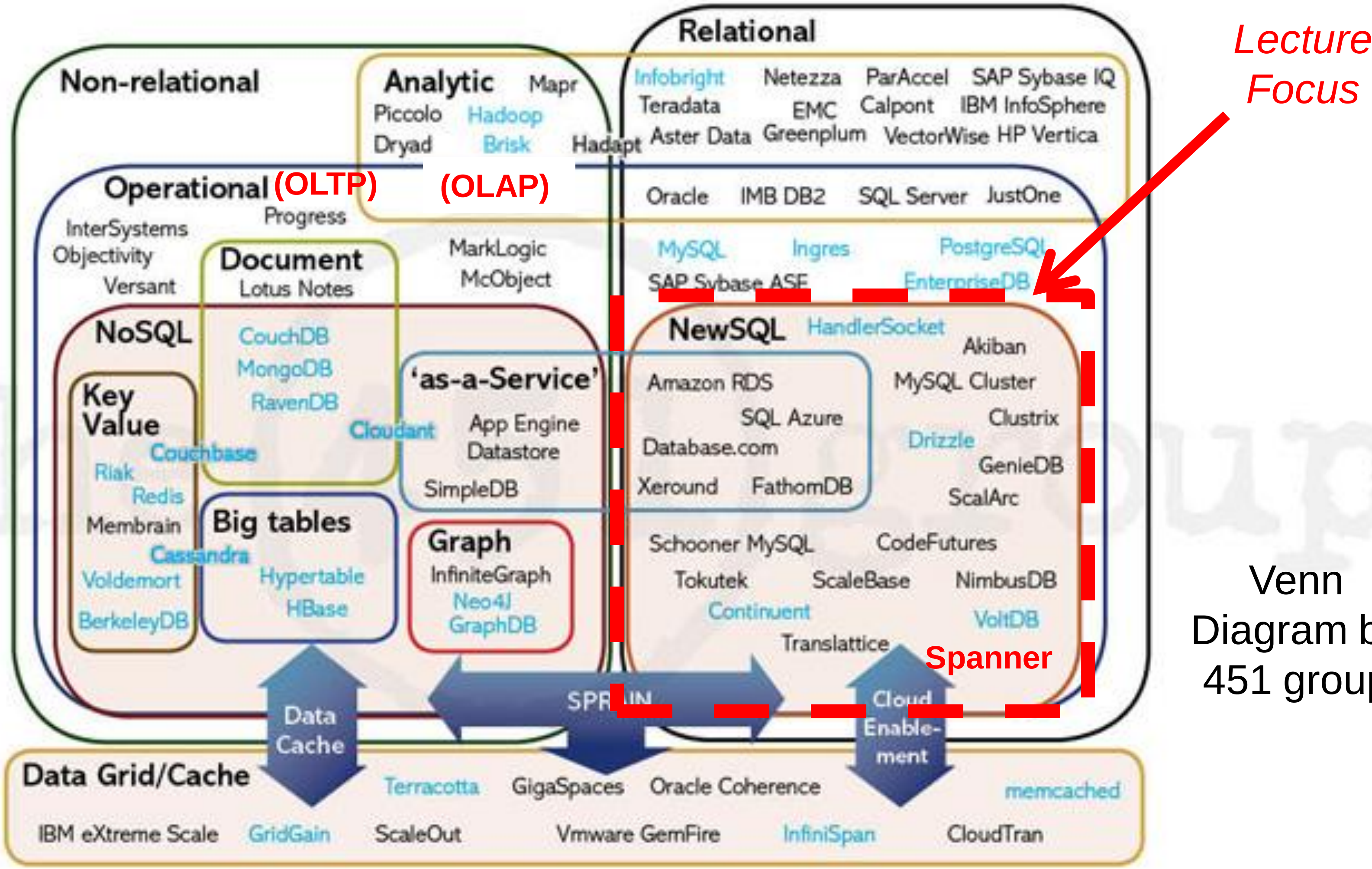
Apache HBase Features

- **Strongly consistent reads/writes:** HBase is NOT an "eventually consistent" DataStore. This makes it very suitable for tasks such as high-speed counter aggregation.
- **Automatic sharding:** HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows.
  - First replica is written to local node, Second to another node in same rack, Third replica is written to a node in another rack (if sufficient nodes)
- **Automatic RegionServer failover (basic element of availability)**
- **Hadoop/HDFS Integration:** HBase supports HDFS out of the box as its distributed file system.
- **MapReduce:** HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.
- **Java Client API:** HBase supports an easy to use Java API for programmatic access.
- **Thrift/REST API:** HBase also supports Thrift (Apache's RPC-like) and REST (W3C's HTTP-like) for non-Java front-ends.
- **Operational Management:** HBase provides build-in web-pages for operational insight as well as JMX metrics.

Apache HBase (When to use?)

- **Not suitable for every problem.**
  - You need **enough data**: a few thousand/million rows.
- Make sure you **can live without all the extra features** that an RDBMS provides (e.g., typed columns, secondary indexes, transactions, advanced query languages, etc.)
  - An application built against an **RDBMS cannot be "ported"** to HBase by simply **changing a JDBC** driver, for example.
  - Consider moving from an RDBMS **to HBase as a complete redesign as opposed to a port.**
- **Have enough hardware:** Even HDFS doesn't do well with **anything less than 5 DataNodes** (due to things such as HDFS block replication which has a **default of 3**), plus a **NameNode**.
  - **HBase** can run quite **well stand-alone** on a **laptop** - but this should be considered a **development configuration** only.

EPL646: Part B
Distributed/Web/Cloud DBs/Dstores



Venn Diagram by 451 group

http://xeround.com/blog/2011/04/newsql-cloud-database-as-a-service

NewSQL Summary

- **OLTP (Online Transaction Processing)**: facilitate & manage transaction-oriented applications (order something, withdraw money, cash a check, etc.)
- **New OLTP:** Consider new Web-based applications such as **multi-player games**, **social networking** sites, and **online gambling networks**.
  - The aggregate number of interactions per second is skyrocketing!.
- **New SQL:** An alternative to NoSQL or Old SQL for New OLTP applications.
- **Examples:** Clustrix, NimbusDB, and VoltDB but also "Spanner" / "F1" (Google's NewSQL DB)

Big Data!

** Michael Stonebraker, June 16, 2011, http://tinyurl.com/9fok4kt*

Google's Spanner

Based on Wilson Hsieh's slides at USENIX OSDI 2012

Spanner: Google's Globally-Distributed Database

James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford

Google's F1 RDBMS

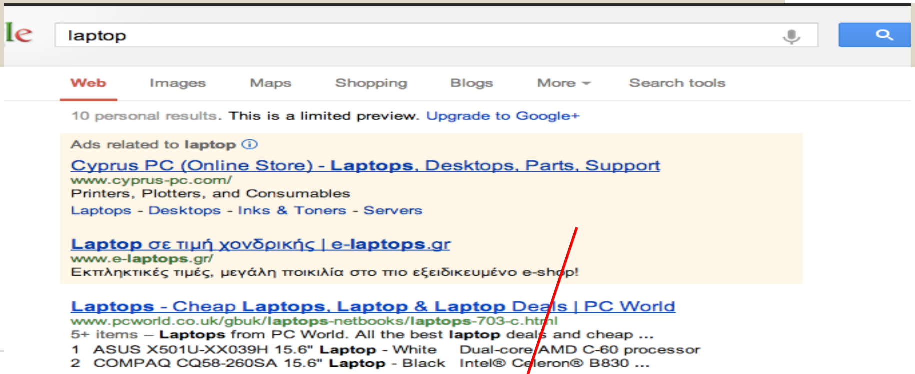
Also the SIGMOD'12 slides

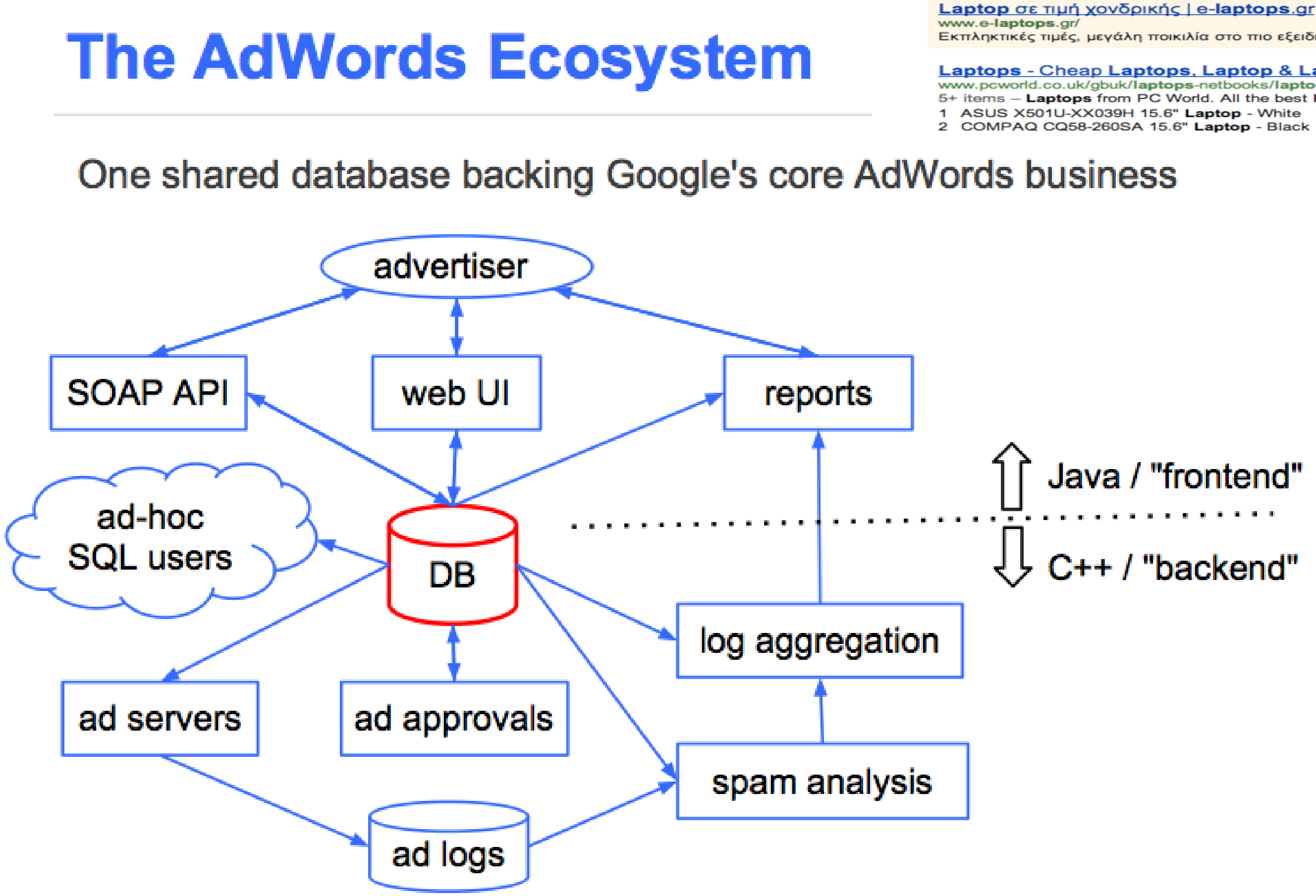
F1 - The Fault-Tolerant Distributed RDBMS Supporting Google's Ad Business

Jeff Shute, Mircea Oancea, Stephan Ellner,
Ben Handy, Eric Rollins, Bart Samwel,
Radek Vingralek, Chad Whipkey, Xin Chen,
Beat Jegerlehner, Kyle Littlefield, Phoenix Tong

SIGMOD
May 22, 2012

The Problem?





Adwords: Instrumental part of Google's business revenue!!!

ACID guarantees on transactions are **fundamentally important**

Sharded MySQL was not scaling well ☹

The Problem?

Can we have:

The Scalability of Bigtable

+

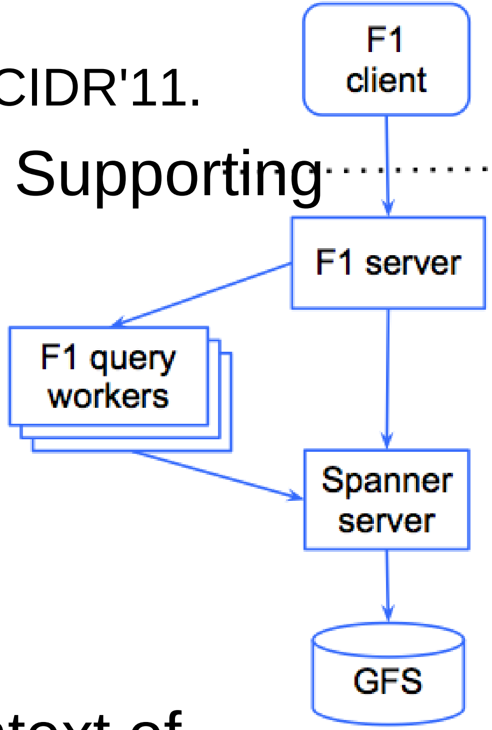
Usability and functionality of SQL databases

=

An ACID-compliant RDBMS system that scales to thousands of nodes (i.e., Google-scale scenarios)

What is Spanner & F1 ?

- **Spanner:** ACID-compliant transaction Storage Subsystem for Google's F1 RDBMS (using GFS)
  - Founded on BigTable
  - Replaces Google's Megastore system + paper, CIDR'11.
- **F1:** The Fault-Tolerant Distributed RDBMS Supporting Google's Ad Business
  - General-purpose transactions (ACID)
  - SQL query language, Schematized tables
  - Semi-relational data model (relational + other)



- Both Running in production
  - Replaced a sharded MySQL database

– Use many ideas known for years in the context of distributed databases (Three Phase Commit), but scale those ideas out to Google-scale scenarios!

Spanner Overview

- **Feature: Lock-free** distributed **read** transactions
- **Property:** External **consistency** of distributed transactions
  - **Commit order** respects **global wall-time order**
  - First system at global scale!
- **Implementation:** Integration of concurrency control, replication, and 2PC
  - Correctness and performance
- **Enabling technology:** TrueTime
  - Interval-based **Global wall-clock time**

Google's F1 RDBMS

How We Deploy

Google

- Five replicas needed for high availability
- Why not three?
  - Assume one datacenter down
  - Then one more machine crash => partial outage

Geography

- Replicas spread across the country to survive regional disasters
  - Up to 100ms apart

Performance

- Very high commit latency - 50-100ms
- Reads take 5-10ms - much slower than MySQL
- High throughput

Google's F1 RDBMS

SQL Query

- Parallel query engine implemented from scratch
- Fully functional SQL, joins to external sources
- Language extensions for protocol buffers

```
SELECT CustomerId
FROM Customer c PROTO JOIN c.Whitelist.feature f
WHERE f.feature_id = 302
  AND f.status = 'STATUS_ENABLED'
```

Making queries fast

- Hide RPC latency
- Parallel and batch execution
- Hierarchical joins