

NewSQL Databases and Time Series Databases

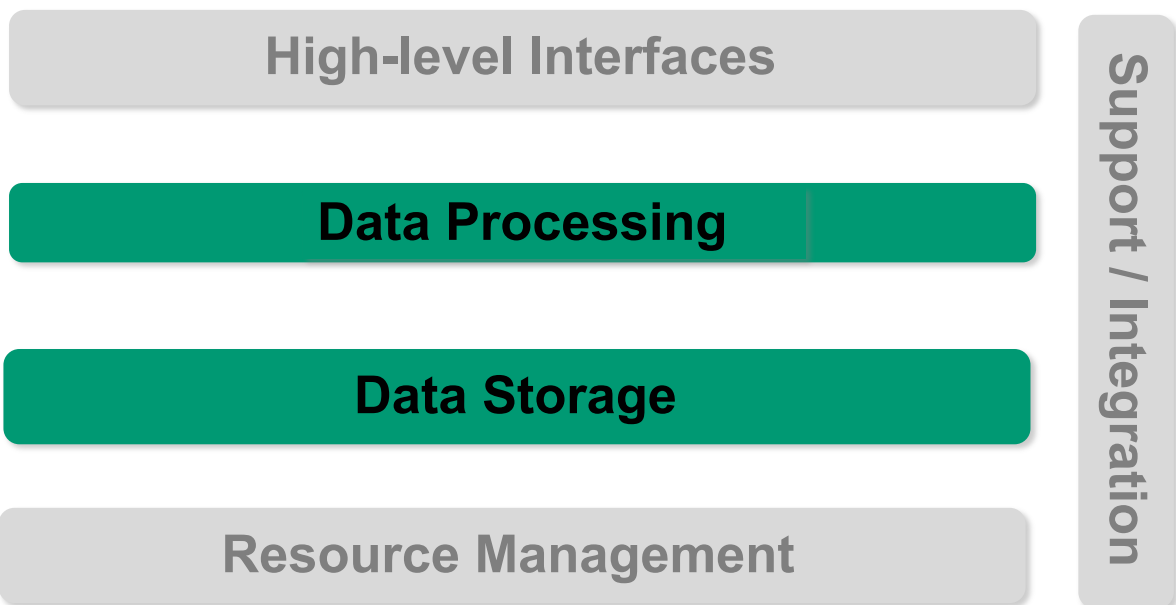
Corso di Sistemi e Architetture per Big Data

A.A. 2023/24

Valeria Cardellini

Laurea Magistrale in Ingegneria Informatica

The reference Big Data stack



Relational database systems

- RDBMS pros:
 - ACID transactions
 - Relational schemas (and schema changes without downtime)
 - SQL queries
 - Strong consistency
- RDBMS cons:
 - Lack of horizontal scalability (to 100s or 1000s of servers)

NewSQL databases

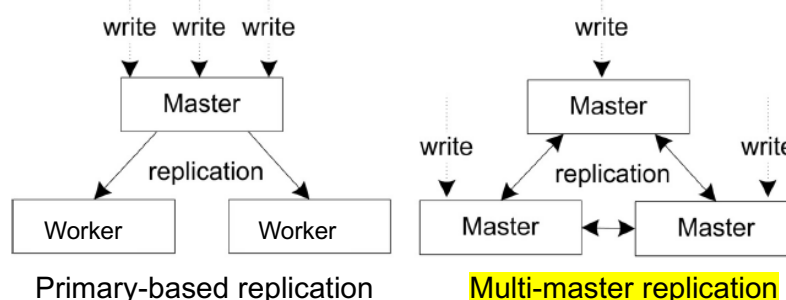
- How to build a **relational database system** that is both ACID compliant and horizontally scalable?
 - i.e., how to make ACID scale?
- **NewSQL**: a class of modern RDBMS
- Goals
 - Provide scalability of NoSQL systems for OLTP workloads, while maintaining ACID support of traditional RDBMS
 - Support SQL

NewSQL examples

- **Google's Spanner**
 - Also as cloud service: [Cloud Spanner](#)
- **CockroachDB**
 - Open-source, born as Spanner clone, then evolved differently
 - See hands-on lesson
- **VoltDB**
- MariaDB Xpand
- NuoDB
- Note: most of them closed source

Replication in NewSQL

- Hot to scale? Multi-master (or master-less) schemes
 - Any node can receive data update statements



- **Google Spanner**
 - Uses Paxos state machine replication to guarantee that a sequence of commands is executed in the same order by all the replicas
- **VoltDB**
 - A transaction manager receives the updates, which are forwarded to all replicas and executed in parallel

Spanner: why

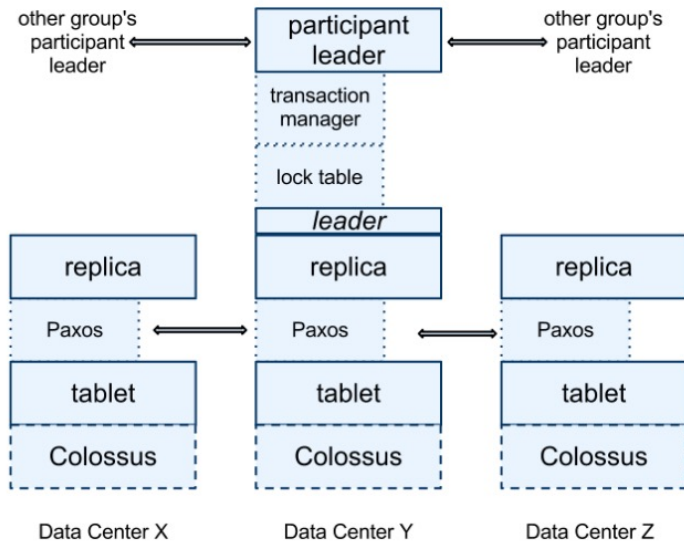
- Google's motivations:
 - “Even though many projects happily use Bigtable, we have also consistently received complaints from users that Bigtable can be difficult to use for some kinds of applications: those that have complex, evolving schemas, or those that want strong consistency in the presence of wide-area replication”
 - We provide a “temporal multi-version database instead of a Bigtable-like versioned key-value store” to make it easier for programmers to write their applications

Spanner

- **Wide-area distributed multi-version database**
 - Support for ACID transactions
 - Strong consistency and high availability
 - SQL-based query language
 - Multi-version data
 - Each version of data is automatically timestamped with its commit time
 - “At the highest level of abstraction, it is a database that shards data across many sets of Paxos state machines in data centers spread all over the world”
- **Runs in production**
 - E.g., storage layer for Google's ads data

Spanner: software stack

- Based on Paxos and Colossus (GFS successor)



Spanner: overview

- Feature: **lock-free distributed read-only transactions**
 - Lock-free: no need of locking to read any data item
 - But of course lock on read/write transactions!
- Property: **external consistency** of distributed transactions
 - External consistency: strictest concurrency-control guarantee for transactions (more than linearizability)
 - System behaves as if all transactions were executed sequentially
 - In a **globally distributed system**
- Implementation: integration of concurrency control, replication, and 2PC
 - Correctness and performance
- Enabling technology: a new API Time called **TrueTime**
 - Used to generate monotonically increasing timestamps and assign them to transactions

Spanner: Google's TrueTime (TT)

- Distributed synchronized clock with bounded non-zero error
 - Returns a time interval that is guaranteed to contain clock's actual time for some time during call's execution
 - Relies on a well engineered tight clock synchronization available at all servers thanks to **GPS clocks and atomic clocks**
- ✗ Requires special hardware and custom-build tight clock synchronization protocol: infeasible for many systems!
 - Spanner runs over Google's private global network (not over public Internet), which is a very high throughput, global fiber optic network that links its data centers

Spanner: concurrency control

- Hybrid approach
 - Read-write transactions are implemented through read-write locks, but read-only transactions are lock-free
- Why is it possible?
 - To read without blocking writes, Spanner (and other DB systems) keep multiple **immutable versions of data**: this concurrency control mechanism is called **multi-version concurrency control (MVCC)**
 - Each write creates a new immutable version of data whose timestamp is that of the write's transaction, such that concurrent readers can still see the old version while the update transaction proceeds concurrently
 - A read at a timestamp returns the value of the most recent version prior to that timestamp, and does not need to block writes
 - Spanner stores multiple versions of data, and a read transaction is basically a read at a "safe" timestamp
 - Proper timestamping is achieved by using TrueTime

Google's Cloud Spanner

- Spanner as Cloud service on Google Cloud Platform
- Globally distributed, ACID-compliant database that automatically handles replicas, sharding, and transaction processing
- High availability: up to 99.999%

- Note: Spanner (and Cloud Spanner) are both strongly consistent and high available on a wide-area scale: CAP theorem?
 - Short answer: technically, Spanner is CP
 - If you are curious: [Spanner, TrueTime and the CAP Theorem](#)

VoltDB

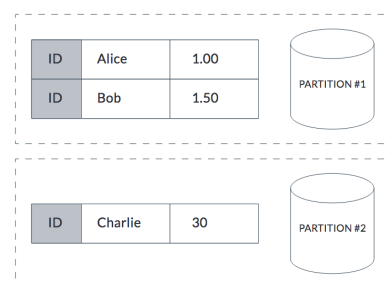
- **In-memory, partitioned, single-threaded, distributed, ACID-compliant database**
 - In-memory: data is held in RAM rather than on disk
 - Partitioned and distributed: database tables are partitioned across multiple servers so to achieve high concurrency and high throughput
 - Single-threaded: serialized processing on data within a single partition thus avoiding locking overhead
 - ACID-compliant: to ensure data consistency, integrity and accurate query results
- **Other features**
 - Based on shared nothing architecture at per-core level
 - Horizontal scale-out on commodity hardware
 - Durability and high availability through async and sync command logging, database snapshot, replication
 - Open-source community edition

VoltDB

- How it began
 - Open source RDBMs ran on memory-based file system
 - Over 80% of time spent on page buffer management, index management, and concurrency management
 - Index management: indexing schemes (e.g., B-tree, hashing) require significant CPU and I/O
 - Locking operations are overhead-intensive
 - Only 12% of time spent doing the real work
 - Lead to [H-Store](#)
 - Developed by M. Stonebraker (2015 ACM Turing award)

VoltDB: partitioning

- Tables are automatically partitioned over multiple servers, and clients can call any server
 - Transparent distribution, but user can choose how to partition the table by specifying the [partitioning column](#)
 - If table is partitioned, each time a new row is inserted into that table, VoltDB decides which partition the row goes into based on the value of the partitioning column
- Selected tables can be replicated over servers, e.g. for fast access to read-mostly data



VoltDB: concurrency control

- Alternative design with respect to Spanner, not using clock-based scheme but based on 2 assumptions
 1. Total available memory is large enough to store entire data
 2. All transactions are short-lived and can be very efficiently executed over in-memory data
- Transactions that involve a single partition are executed sequentially (from beginning to end) in a single-threaded, lock-free environment
- Transactions that span multiple partitions are sent to a global controller, which is responsible for deciding a serial order

Time series data base (TSDB)

- How to analyze DevOps monitoring, application metrics, sensor data from smart factories, smart cities, or smart vehicles?

Time series database (TSDB)

- A possible solution, not the only one!
- Optimized for handling **high-volume time series data**
 - *Time series*: sequence of data points (arrays of numbers) indexed by time (a date time or a date time range), e.g.:
 - Stock prices (price curve)
 - Energy consumption (load profile)
 - Temperature values (temperature trace)
- Optimized for providing complex logic to analyze time series data
 - Queries for historical data, replete with time ranges and roll ups and arbitrary time zone conversions are difficult in DBMS

TSDDB: overview

- Create, enumerate, update and destroy various time series and organize them in some fashion
 - Series may be organized hierarchically and have companion metadata
 - Provide basic calculations on a series as a whole (e.g., multiplying, adding, or combining various time series into a new time series)
 - Filter on arbitrary patterns (e.g., day of the week, low value, high value)
 - Provide statistical functions that are targeted to time series data
 - mean, mode, stddev, percentile, exponential moving average, ...

TSDDB: some open-source products

- [CrateDB](#)
- [Graphite](#)
 - Stores numeric time-series data and renders graphs on demand
- [InfluxDB](#)
 - See hands-on lesson
- [KairosDB](#)
 - Stores time series on Cassandra
- [Prometheus](#)
 - Popular open-source monitoring and alerting system, which collects and stores metrics as time series data
- [Riak TS](#)
 - NoSQL key-value store optimized for time series data with masterless architecture (similar to Riak KV)

InfluxDB

- Written in Go
- Supports high write loads and large dataset storage
- Conserves space through downsampling
 - By automatically expiring and deleting unwanted data as well as backup and restore
- Provides easy-to-use SQL-like query language for interacting with data
- Provides simple, high performing write and query HTTP(S) APIs, e.g.:
 - To create a database

```
curl -i -XPOST http://localhost:8086/query --data-urlencode "q=CREATE DATABASE mydb"
```
 - To write data

```
curl -i -XPOST 'http://localhost:8086/write?db=mydb' --data-binary 'cpu_load_short,host=server01,region=us-west value=0.64 1434055562000000000'
```

InfluxDB: time series

- Data organized by **time series**, which contain a measured value, like “cpu_load” or “temperature”
- Time series have zero to many points, one for each discrete sample of the metric
- Points consist of:
 - **time** (a timestamp)
 - **measurement** (e.g., “cpu_load”)
 - at least one **key-value field** (the measured value itself, e.g. “value=0.64”, or “temperature=21.2”)
 - and 0 to many **key-value tags** containing any **metadata** about the value (e.g. “host=server01”, “region=EMEA”, “dc=Frankfurt”)

InfluxDB: time series

- General format of points:

```
<measurement>[,<tag-key>=<tag-value>...] <field-  
key>=<field-value>[,<field2-key>=<field2-value>...]  
[unix-nano-timestamp]
```

- Timestamp is optional: InfluxDB uses the server's local nanosecond timestamp in UTC if the timestamp is not included with the point

- Examples of points:

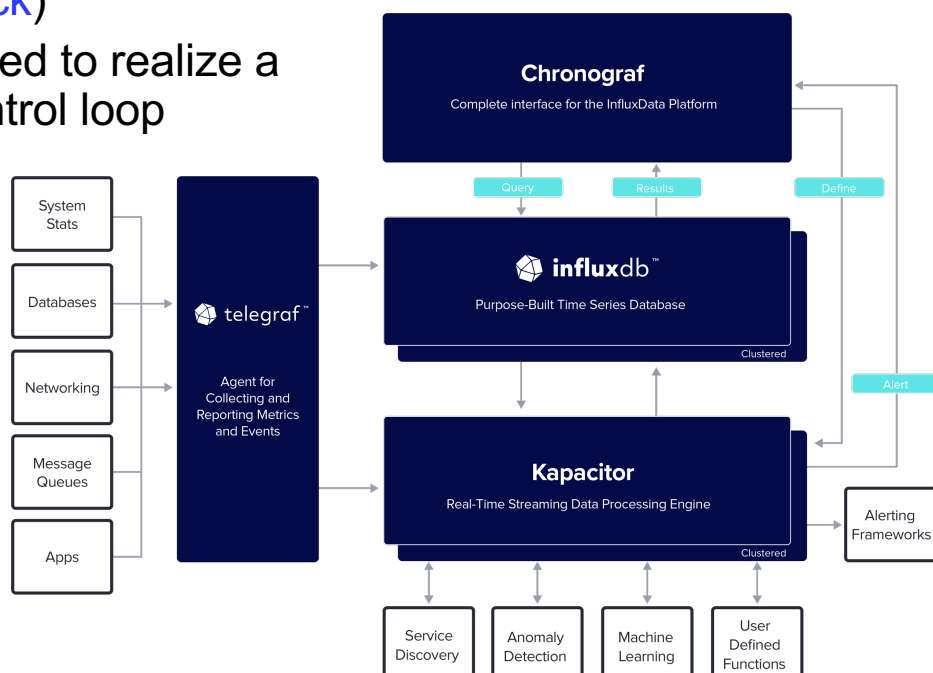
```
cpu,host=serverA,region=us_west value=0.64  
payment,device=mobile,product=Notepad,method=credit billed=33,licenses=3i 1434067467100293230  
stock,symbol=AAPL bid=127.46,ask=127.48  
temperature,machine=unit42,type=assembly external=25,internal=37 1434067467000000000
```

InfluxDB: data store

- A measurement is like a SQL table, where the primary index is **time**
- With respect to DBMS:
 - No need to define schema up-front
 - Null values are not stored
- InfluxDB limitation
 - Horizontal scalability: clustered installation available only in commercial product

InfluxDB: TICK stack

- Integrated with Telegraf, Chronograf and Kapacitor (**TICK stack**)
- Can be used to realize a MAPE control loop



InfluxDB: TICK stack

- **Telegraf**: server-based agent to collect metrics and events from cloud services, applications, IoT sensors
 - Supports a variety of metrics through input plugins
 - Supports output plugins to send metrics to other data stores, services, and message queues (InfluxDB, Graphite, OpenTSDB, Kafka, MQTT, ...)
- **Chronograf**: administrative user interface and visualization engine
 - To build dashboards with real-time visualizations of data and create alerting and automation rules
- **Kapacitor**: native data processing engine
 - To process both stream and batch data from InfluxDB
 - E.g., to perform specific actions (e.g., dynamic load balancing) based on alerts (e.g., above load threshold)

References

- Grolinger et al., [Data management in cloud environments: NoSQL and NewSQL data stores](#), *J. Cloud Comp.*, 2013
- Corbett et al., [Spanner: Google's globally distributed database](#), *OSDI 2012*
- Stonebraker and Weisberg, [The VoltDB main memory DBMS](#), 2013
- Dunning and Friedman, [Time Series Databases](#), O'Reilly, 2015