hag·i·og·ra·phy
(noun)

ChristosTheGreekGodofDatabases.com

• Pinterest meets Causal Encounters meets Kickstarter meets Twitter
  – With Christos!
More reads than writes.
All media stored outside of DBMS.
How do we choose the right database architecture?

Outline
- Single-Node Databases
- NoSQL Systems
- NewSQL Systems

Late-1990s / Early-2000s
- All the big players were heavyweight and expensive.
  - Oracle, DB2, Sybase, SQL Server, Informix.
- Open-source databases were missing important features.
  - Postgres, mSQL, MySQL.
Mid-2000s

- MySQL + InnoDB is widely adopted by new web companies:
  - Supported transactions, replication, recovery.
  - Memcache for caching queries.

Let's go with MySQL.

We're getting a lot of traffic.

Our database server is saturated!

How do we increase the capacity of our database server?

Idea #1:

Buy a faster machine.
Scaling Up

- More disks.
- More RAM.
- Faster CPUs.
- Use SSDs.

(+) Requires no change to application.
(+) Improvements are immediate.
(-) Expensive! Diminishing Returns.
(-) Single Point of Failure.

Replicate database on multiple servers.

(+) Requires no change to application.
(+) Parallelize read operations.
(+) Improved fault tolerance.
(-) Expensive! Diminishing Returns.
(-) Writes limited to slowest node.
Idea #3: Cache query results.

- Reduce load on DBMS.
- Fast API.
- Extra roundtrip per query.
- Requires application changes.
- Doesn’t help write-heavy apps.

Idea #4: Push SQL into stored procedures.
**Stored Procedures**

Application Code

(+) Reduces network roundtrips.
(+) Less lock contention.
(+) Modularization.

(-) Application logic in two places.
(-) PL/SQL is not standardized.

**Idea #5:**
Shard database across multiple servers.

**Sharding / Partitioning**

(+) Parallelize all operations.
(+) Much easier to add more hardware.

(-) Most DBMSs don’t support this.
(-) Joins are expensive.
(-) Non-trivial to split database.
We want to scale out but writing a sharding layer is hard.

Some parts of our application don’t need a full-featured DBMS.

Idea #6: Give up ACID guarantees for scalability.

Eventual Consistency
Late-2000s (NoSQL)

- NoSQL systems are able to scale horizontally right out of the box by giving traditional database features.

ChristosTheGreekGodofDatabases.com

- We need to process payments.
- We don’t want to lose orders.
- We need joins and ACID transactions.

Strong Consistency

Use Two-Phase Commit

Nice Christos Pictures!

Send Money

$100

Use Two-Phase Commit

Thanks!

$100
Idea #7: Keep guarantees, optimize for workload type.

Early-2010s (NewSQL)

- New DBMSs that can scale across multiple machines natively and provide ACID guarantees.

Conclusion

- RDBMS (Single-Node):
  - MySQL, Postgres
- NoSQL (Multi-Node):
  - Key-Value, Documents, Graphs
- NewSQL (Multi-Node):
  - Transaction Processing, MySQL Sharding
What DBMS should my start-up use?
• Christos is teaching **15-826** this fall:
  – Multimedia Databases and Data Mining
• Send me an email if you’re interested in working on a database research project.