Evolution of Web Application Architecture

PHP Usergroup Düsseldorf

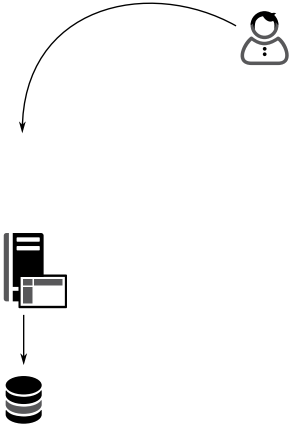
Kore Nordmann / @koredn / <kore@qafoo.com>

Spetember 24th, 2015

Qafoo
passion for software quality

Hi, I'm Kore

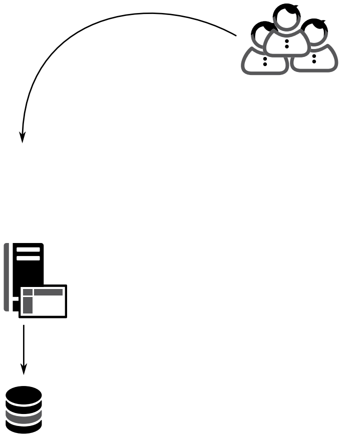
Evolution



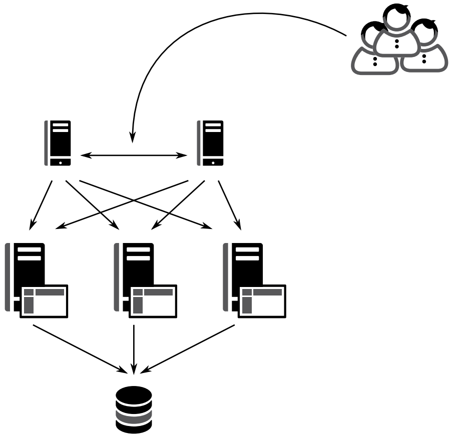
Too many visitors

CC BY 2.0 Scott Cresswell – `https://flic.kr/p/knkntv`

Evolution



Evolution

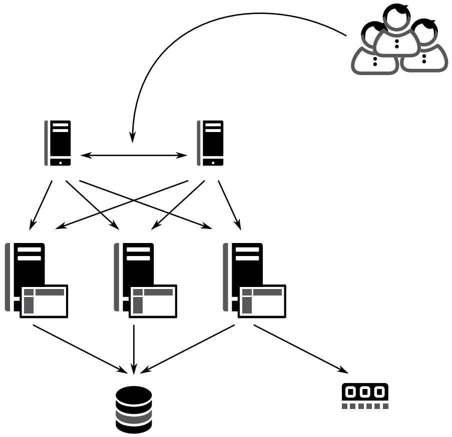


Lessons Learned: Load Balancing

- Works because of HTTP & PHP
  - HTTP is LCoDC$SS
  - PHP is build for shared-nothing
- Round Robin works best
  - Sticky sessions will overload certain servers

Non sticky session – how?

Evolution



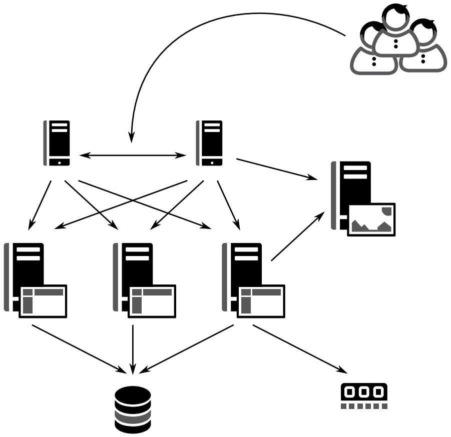
Lessons Learned: Non-Sticky Session

- Put session on memcached / Redis
  - Mostly trivial because of existing extensions

Qafoo
passion for software quality

Where to put the static data?

Evolution



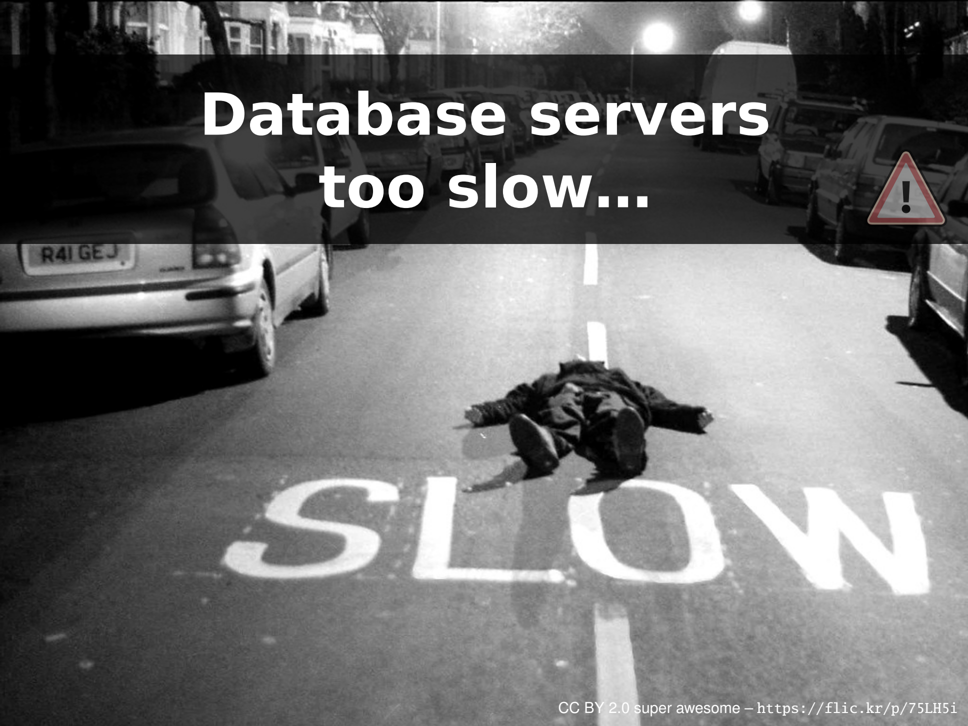
passion for software quality

Lessons Learned: Static Files

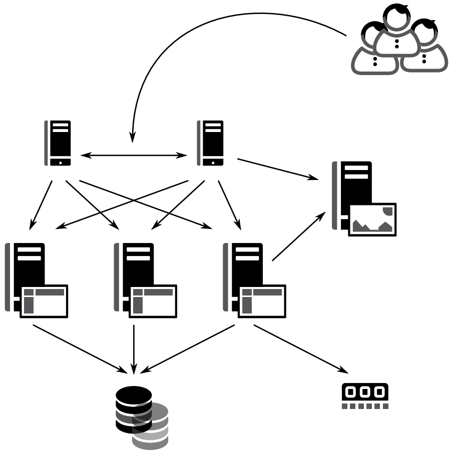
- NFS will eventually lead to dead locks
  - . . . still seems the most popular solution around.
- Multiple domains can hurt performance (TCP slow start)
- Using dedicated CDN providers can help
  - Content locality

passion for software quality

Database servers too slow…



Evolution

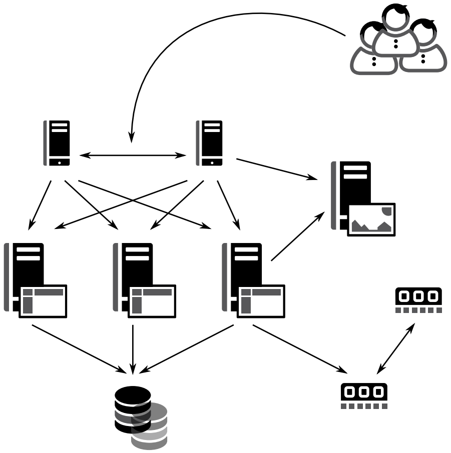


Lessons Learned: Replicate Database

- Master Slave Replication is fairly easy to set up
  - Obviously only scales READs
  - WRITEs are usually not your first problem

Database servers too expensive…

Evolution



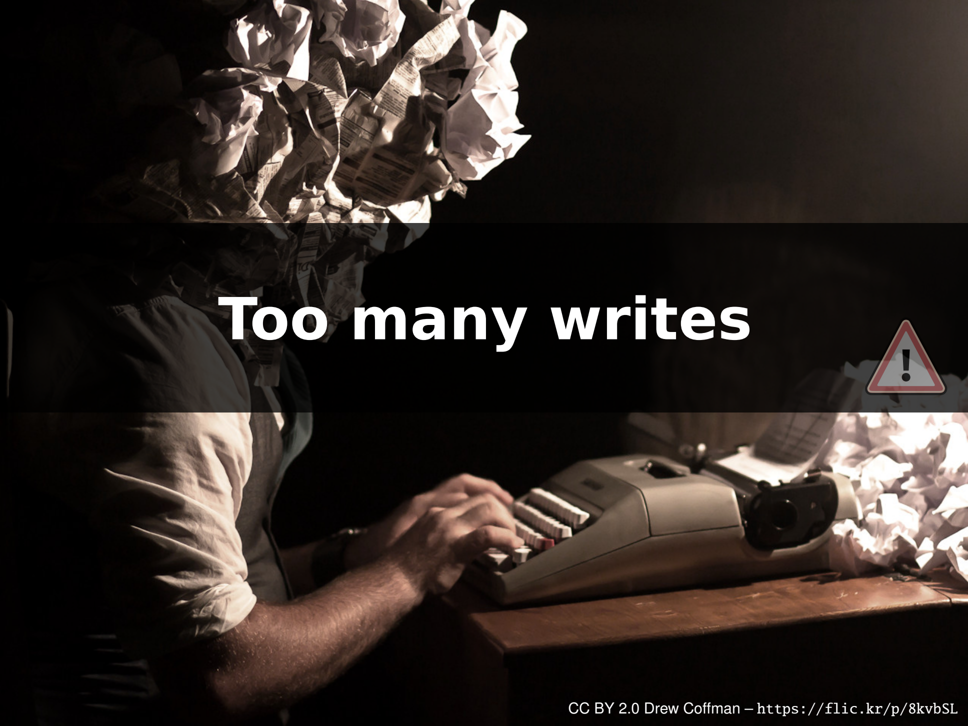
Qafoo
passion for software quality

Lessons Learned: Cache With Memcache

- Cache all the things in *memory*
  - Cache entities
  - Cache collections
  - Full page cache
- Cache invalidation

  There are three hard things in Computer Science:
  Cache invalidation and off by one errors.

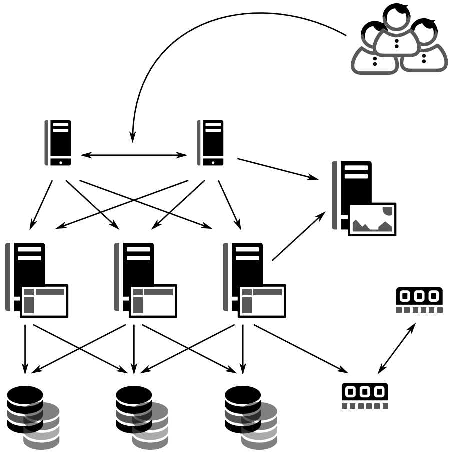
  - Cache dependency calculation
  - The paging problem



Too many writes

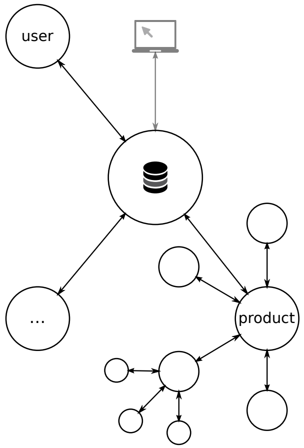
⚠

Evolution



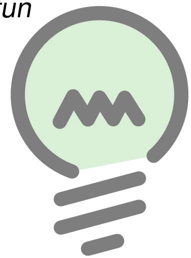
Sharding

- Split tables across multiple nodes
  - Vertical sharding
- Shard by consistent hashing
  - Horizontal sharding



Lessons Learned: Sharding

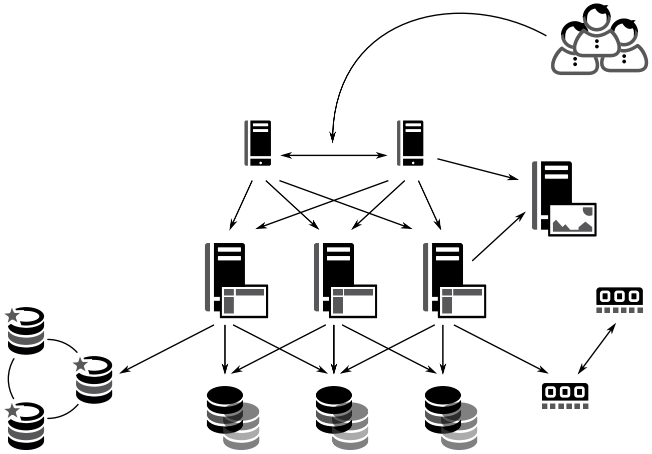
- Shard by table
  - . . . or even shard by consistent hash per entity
- No referential integrity checking
- Queries are limited to sharding solution
- Schema updates across multiple shards are *fun*



passion for software quality

Setup too complex

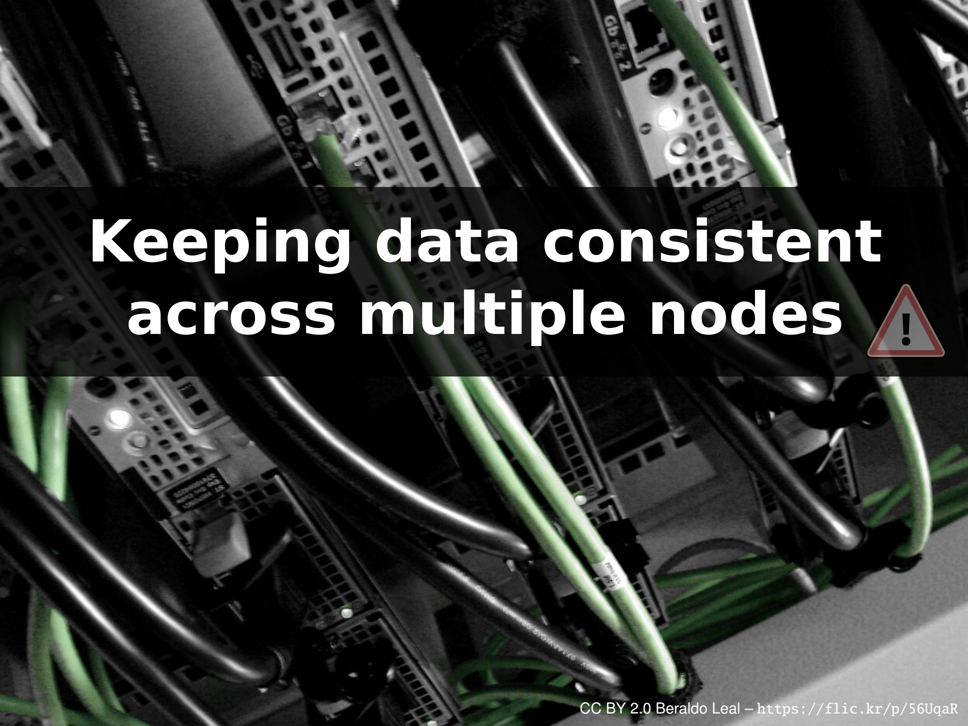
Evolution



Lessons Learned: NoSQL

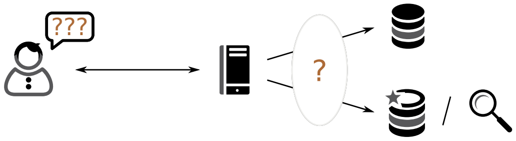
- Usually solves one problem really well:
  - Sharding
  - Multi-Master-Replication
  - Cross-shard queries
- Usually omits:
  - SQL
  - Referential Integrity
- . . . we lost all relevant features from Relational Database Management Systems anyways. . .

Qafoo
passion for software quality

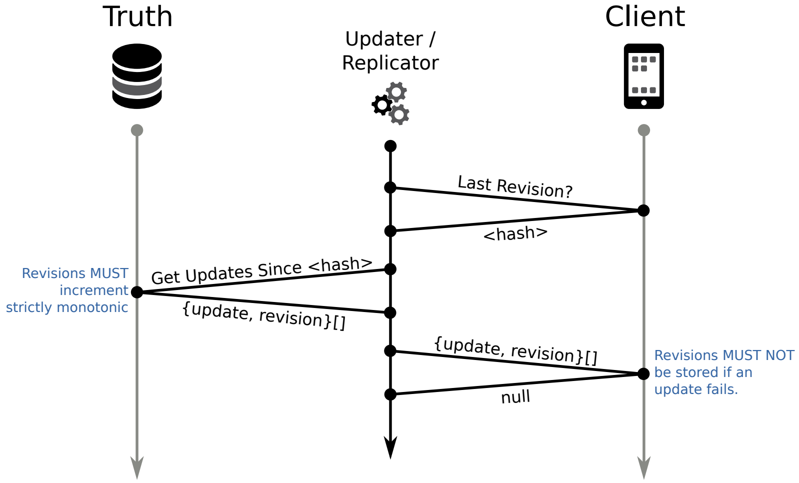


Keeping data consistent across multiple nodes ⚠

Data Consistency Across Nodes



Eventual Consistency



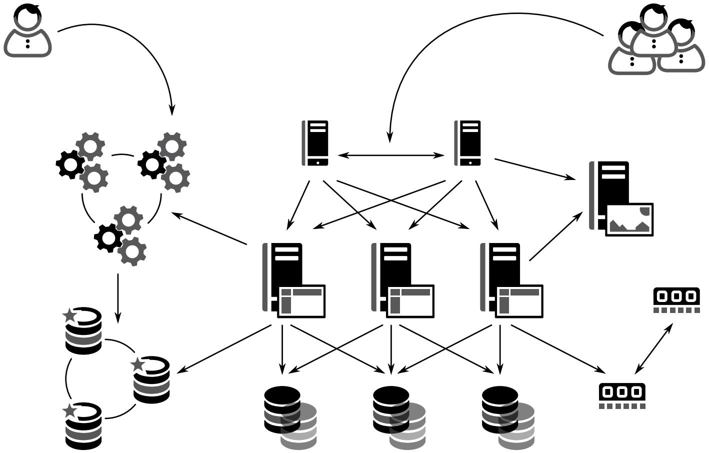
Lessons Learned: Data Consistency

- Embrace Eventual Consistency
  - Compaction is hard
  - Data migrations are hard

Qafoo
passion for software quality

Business wants to query data

Evolution



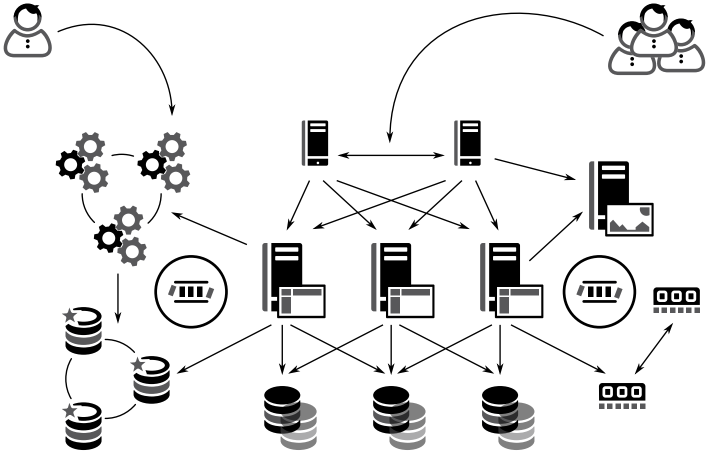
Lessons Learned: Map-Reduce

- Execute queries on distributed databases
- New query language to learn
  - Your developers write analysis scripts, instead of the business analysts writing slow SQL queries

How to orchestrate?

⚠

Evolution

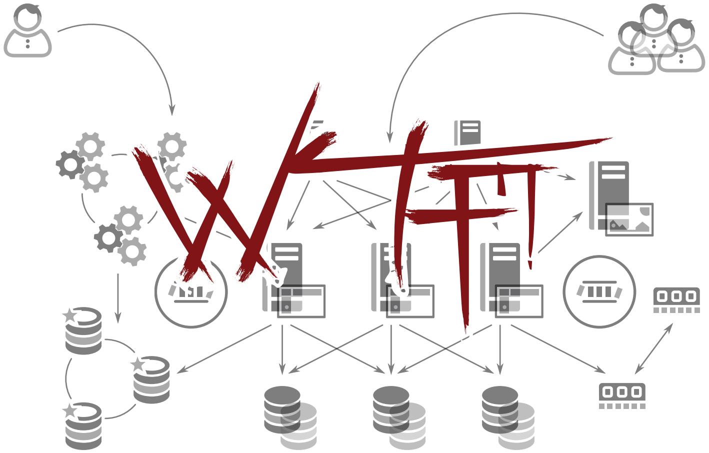


Qafoo
passion for software quality

Lessons Learned: Queues

- Queues can ensure data is processed asynchronously
  - Data consistency must be ensured even when pushing into queues
  - Following the data flow of an action can be “tricky”
- Used to distribute data between systems

Evolution



Microservices

*Apply **Seperation of Concerns** on service level to allow for seperate teams & technologies per concern.*

- Microservices **can** simplify things:
  - Choose optimal technology stack per team & concern
- Microservices **will** also complicate things:
  - Automated deployment is a must
  - Service orchestration is still a problem
  - Service downtimes and latency must be handled gracefully (Eventual Consistency)
- Big Data^{TM} will stay a problem
- Sensible services are often not *micro* any more. . .

Qafoo
passion for software quality

Lessons Learned (subjective)

- Boring technology choices will often work best
  - Just start & stay with LAMP?
- Only bring in shiny new technologies with care
  - There are enough reasons to eventually do that, though

The Hipster Says:

DO NOT USE HIPSTER TECH!*

* Except you evaluated
it as the correct solution
for your case

Conclusion

There are many developers, documentation & experience for boring technologies

Evaluate before adding new technologies (ATAM)

Do not jump on every bandwagon – this includes microservices

Qafoo
passion for software quality

THANK YOU

Rent a quality expert
qafoo.com