CIS4301 Guest Speakers

Speakers: Dave Stanton (@gotoplanb), Sean Goldberg(sean@cise.ufl.edu) Pierre St. Ju
Notes: Ryan Roden-Corrent

March 10, 2014

1 App Development (Dave Stanton)

1.1 Users Goals Metrics

- Who is the user? What are their goals?
- Come up with a "persona" to represent the people who will be using your app.
- Have general "metrics of success"
- One-size-fits-all software doesn't really appeal to anyone in particular

1.1.1 example

Developer

IT Manager

CIO

1.1.2 5 person useability studies

Often just testing with 5 people can give great insight into what works and what doesn't in terms of the user interface. Track the errors they make (thinking they were clicking on something when they really weren't) and tweak the software. Need to make sure you have the appropriate testers for the software.

1.2 Strategy

Exit Strategy What will you do if it takes off? Is it a full time project or just a project done out of curiosity?

Defense What makes your app different from what already exists?

Funding source dictates how you can build features

1.2.1 Funding

bootstrap Your own money, you dicate how to build the software

grant grants tend not to care as much how it is build, just whether it achieves the end goal

With grants, need to know when to stop. Don't undervalue your time.

1.3 Flow

Flow Diagram Create before writing code, can save time in the long run.

Wire Frame Representation of user interfaces. Each wireframe should have an associated persona and goal. Appearance not important, very high level

Composite Mock-up, generally looks better than wireframe

Paper Prototype Walk someone through using a paper version of the app

Define MVP Minimally Viable Product: what is the minimum required for app to "work"

Prioritizing Features what are the most important features? What are blocking features? In what order should features be completed so everyone can work in parallel?

Tickets use goal tracker (e.g. Trello) and issue tracker (e.g. github issues.)

Sprints define a time period for work, set backlog during planning meeting, typically have once-a-day $\approx$ 15min "standup" meeting

Quality Assurance mutliple environments (work, staging/integration, production). Test in an environment exactly like the "real-world". Run test suites.

Release

Retrospective

1.4 Common Trouble

Scope creep Keep adding new features, which delays release and bloats product

Nebulous Hierarchy can't have a "flat" team. Assign specialties or lead roles to members.

Too many stakeholders Who "says yes" for a particular kind of decision?

Automating too soon don't over-engineer too soon. Just make it happen as quick as possible

1.5 Stack

Make sure you are using the appropriate technology for the project

2 Crowd Sourcing (Sean Goldberg)

2.1 Big Data

Volume Existing data waiting to process

Velocity Data in motion (Streaming)

Variety structured, unstructured, text, multimedia

Veracity data in doubt (inconsistency, incompleteness, ambiguity, latency)

Want to organize this data. Example: Wikipedia info box.

2.2 Machine Learning

training set → learning algorithm → heuristic

2.2.1 Examples of use:

- Self-driving car
- Machine translation (like Babel Fish from Hitchhiker's Guide)
- Amazon reccommendations
- Apple's Siri (natural language question → database query)
- license plate processing
- Watson (jeopardy playing robot)

2.2.2 Current Machine Learning Limitations

- Natural Language Processing still limited

2.3 Crowd Sourcing

Human processing power can make up for areas that machines lack. Try to combine human and machine computation to complete a goal.
Bringing in a single human is not efficient. Need an effective way to source the collective knowledge of many humans.

2.3.1 Amazon Mechanical Turk

Provides a marketplace for sourcing human intelligence.

- Make an open call for jobs
- People sign up for jobs
- People complete task with human intelligence
- People get payed
- Allows parallelization of tasks

Example use: Collecting Training data for an intelligent system. If you are training a system to identify emotions by looking at a picture, you need people to look at the pictures and classify them so the machine can learn from them.

2.3.2 TurKit

A programming toolkit that integrates human intelligence as subroutines in your code.

2.3.3 Soylent

A MS Word plugin using people to perform word processing tasks that are difficult fr a computer (shortening, proofreading, macro)

2.3.4 CrowdDb

Use human intelligence to correct and optimize a database in ways that are difficult for a machine.

2.3.5 DbPedia

Make wikipedia queryable like a database by using human intelligence.

2.4 CASTLE (Crowd Assisted System for Text Labeling and Extraction)

Converting unstructured text into a structured form (database) that can be queried.
Combine probablilistic database with crowd knowledge.
Run a machine learning algorithm over text to tag tokens. The algorithm generates a set of possible tags, each with a given correctness probablility. Identify uncertain tags and source them to humans by automatically posting HITs.

3 Distributed Databases and their applications: From DHTs to Memcache to Twister

By Pierre St. Juste

- Distributed Hash Tables (Memcache, Dynamo)
- Distributed Datastores (Bigtable, MongoDB)
- Real Applications (Bitweav, Twister, Litter)

3.1 Centralized Data Storage

$\boxed{\text{Presentation tier}} \leftrightarrow \boxed{\text{Application Tier}} \leftrightarrow \boxed{\text{Data Tier}}$
Infeasible for large amounts of data with frequent access.

3.2 Distributed Data Storage

Increase performance through parallelism

3.2.1 Amazon Dynamo

- NoSQL key-value
- Accesible via HTTP REST API
- Data Auto-partitioned

Different macines responsible for storing different ranges of keys. Want to ensure key distribution is random to achieve good load balancing (need a strong hash function). Good load balancing can avoid performance bottleneck and network bottleneck.

3.2.2 Questions about key-value stores

handling replication? Generate multiple key-val pairs with different keys but same value

load balancing? Strong (sha1) hash of key

map table to KV store? map each sell to a key (e.g. key = tablename_primarykey_column)

3.3 Memcache

Designed to work alongside a database.

- all information stored on disk
- key-value pair has no data-type restrictions
- read-only cache for information stored in DB
- Memcache server failure is not critical

A number of memcache servers store information that is requested from DB. A request first checks the Memcache, and only has to check the DB if the requested value isn't present in the Memcache.

3.3.1 Limitations of key-value stores

- SQL like queries on DHT can be ineficcient

3.4 Google Bigtable

- Read a single row or range
- can request timestamp range
- implemented as multi-map
- sections of table stored as "tablets" across servers
- meta data servers track location of tablets
- key-value lookup performed to retrieve table contents
- **More efficient SQL-like lookups**

3.5 MongoDB

- Schema-free
- Document oriented
- Scaling and load-balancing through sharding
- Store arbitrary objects directly in database

3.5.1 Sharding

Break a key space evenly across servers.

3.6 Distributed Microblogging

3.6.1 Twister

- P2P anonymous microblogging service
- use Bitcoin for user registration
  - everyone is a witness to other's transactions
- use Bittorrent DHT for data storage
  - download torrent file to get keys
  - values are IP addresses of people who have needed data
  - in this case, can be used to store tweets in a decentralized manner

3.6.2 Litter

- each node runs a simple python webserver
- each node uses SQLite database
- Updates are pushed through IP Multicast
- `https://github.com/ptony82/litter`

4 Grooveshark Architecture, Managing DB at Scale, NoSQL

- multi-tiered stack
- CACHE EVERYTHING
- ansynchronous code is your friend
- different workloads → different DBMSes (don't make sql do everything)

4.1 The Website

- several front-end servers with load balancer
- requests are routed to a random front-end server
- only load some of the data synchronously
- load most of the data asynchronously
- get the user on the site as fast as possible!

4.2 Gearman / Memcached

- Most frontend GS calls are asynchronous
- gearman/memcached serve as buffer to data stores and a common 'language' for other services
- gearman performs actions at behest of user

4.2.1 Gearman

- Asynchronous task manager
- job submitted to a queue
- worker processes listen on queues, process jobs as they come in
- stateless, workers just know what queues to listen on and what to do upon job arrival

4.2.2 Memcached

- distributed hash table
- key-value store
- each memcached process is a bucket
- all library calls go to memcached first
- eliminate unnecessary calls to data stores, reduce cost of data retrieval

4.3 Data Stores

- highly heterogenous environment (MariaDB, Percona, MongoDB ...)
- not all data stores are made the same
- different R/W tolerance due to differing data storage
- each DBMS excels in certain areas
- only MySQL, Postgres, and Hive (kinda) use SQL
- do not maintain same guarantees as SQL (could be good thing)
- SQL: good for transactional, ACID complient storage, can be unwieldy
- NoSQL: no ACID guarantees but is more flexible (good for e.g. comment system)

- CAP Theorem

Commonalities among data stores:

- caching
- indices!!!
- backups

4.3.1 MySQL

- primary backend, go-to unless there is a use case for a different DB
- use open source forks instead of Oracle MySQL
  - MariaDB
  - Percona
- top-down topology
- production master/slave
- used by front-end servers exclusively

4.3.2 MongoDB

- distributed NoSQL system: node awareness, eventual consistency
- JSON-like syntax: type sensitive arrays and objects
- indices: more sensitive than SQL counterparts

```
use gs;
db.myColl.find({x:5})
```

4.3.3 Hive/HBase

- Part of apache Hadoop project
- Build on HDFS (Hadoop Filesystem)
- Hive: ad-hoc data querying (data warehouse)
  - transforms SQL to Map/Reduce job
- HBase: column store, easy aggregation, write-heavy workloads

4.3.4 Redis

- in memory
- like memcached, but more fully featured
- sorted sets, lists
- mostly used for internal services