Monday, March 08, 2010
10:17 AM

Services

So far, we've learned to compute with Hadoop, but the output of a computation is read on the command line.

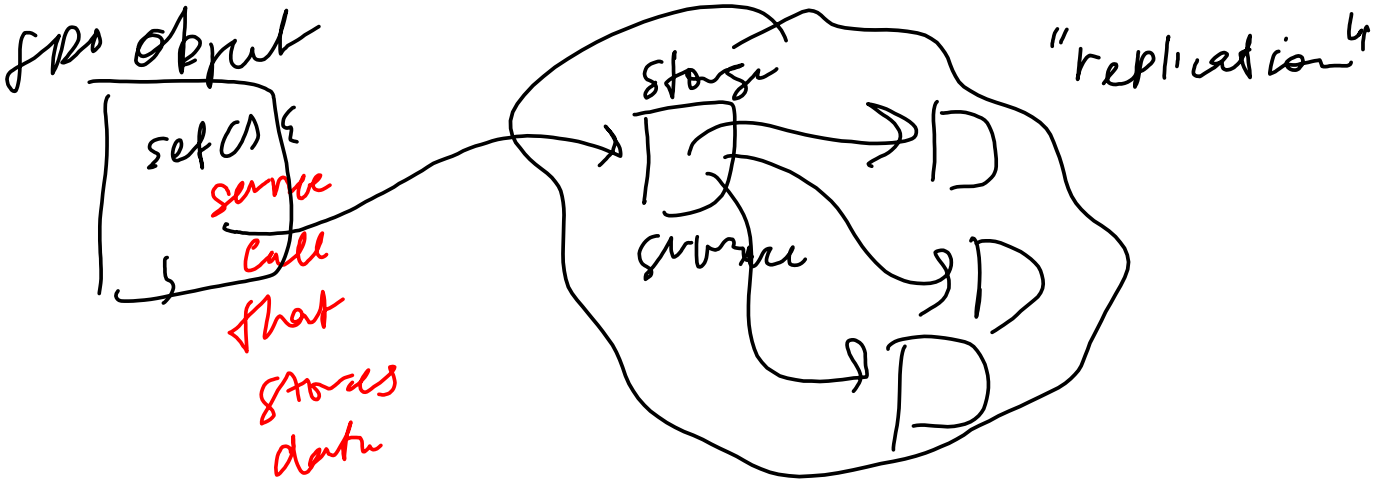
Next steps: transform a hadoop computation into a **service.**

How does JDO accomplish its magic?

Monday, March 08, 2010
10:30 AM

How does JDO accomplish its magic?

@Persistent annotations are interpreted by a factory to make CRUD calls to a persistent datastore service using Map/Reduce (but not using Hadoop, alas!).
Calls to the service actually manipulate the datastore.



Monday, March 08, 2010
10:18 AM

Properties of a service

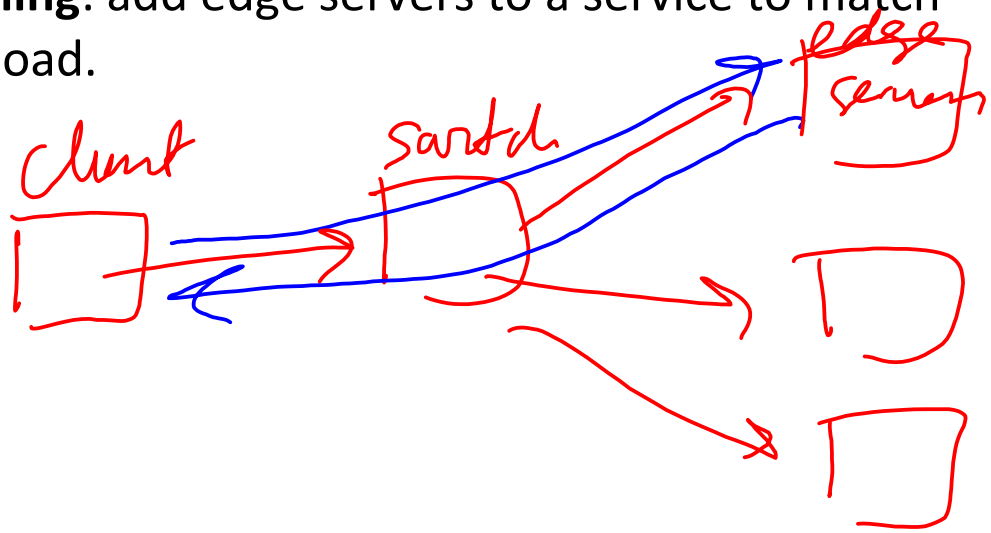
Multiple concurrent requests.

Each request receives **one response.**

Requests are **independent** of one another.

Flowless switching: no continuity in choice of server.

Edge scaling: add edge servers to a service to match request load.



But

Wednesday, March 09, 2011
4:33 PM

But

- Hadoop does *one* job.
- There is no concept of query independence, simultaneity

Service-Oriented Architecture

Monday, March 08, 2010
10:47 AM

Service-Oriented Architecture (SOA)

Refers to the practice of breaking up an application into components that are implemented as **independent services.**

Kinds of services include:

Datastore services: store and retrieve kinds of data/objects.

Filestore services: store and retrieve files.

Lookup services: (quickly) query into (potentially huge) databases/corpora.

Compute services: perform some compute-intensive operation, e.g., analyze an image, convert voice to text, mine corpora for patterns, look for evidence of extraterrestrial life, etc.

The dream of SOA

"A supercomputer in the palm of your hand"
Your phone contains **applications**,
that make requests of **services**,
that invoke potentially vast **computing infrastructure**.
Push complexity into the services; keep the applications relatively simple.

Note: not just for phones
Business process and workflow.
Desktop applications
Scientific applications (e.g., gene matching)

SOA example: google voice search

Monday, March 08, 2010
11:03 AM

SOA example: google voice search

You speak into your phone.

Google responds by searching for what you said.

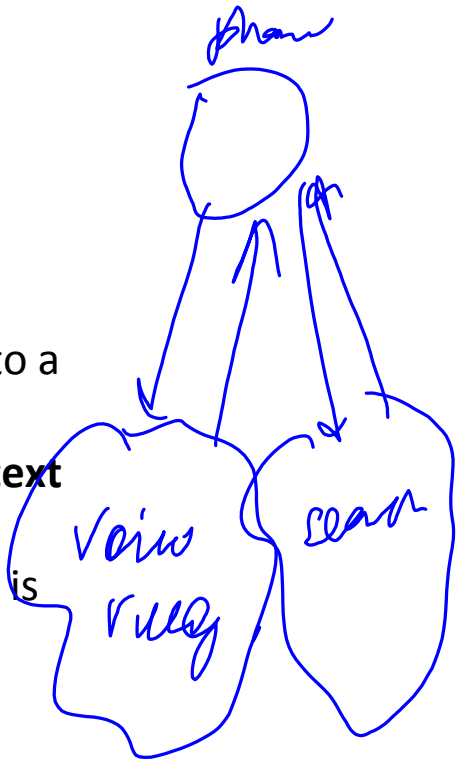
How does it work?

Your phone compresses your speech and ships it to a **google voice-recognition service**.

The service uses MapReduce to **convert voice to text** and returns the text to your application.

Your application then searches for the text, which is another service call.

Result is everything related to what you said.



SOA example: google goggles

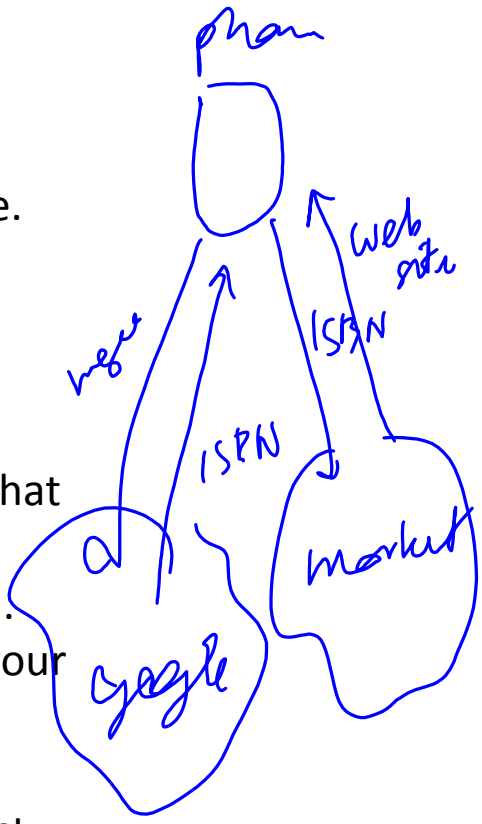
Monday, March 08, 2010
11:06 AM

SOA example: google goggles

You take a picture of something with your phone.
Google responds with how to buy it.

How does this work?

You tell goggles a search domain, e.g., CD album covers, novels, etc.
Your phone invokes an **image search service** in that domain, based upon images of albums, for the album's ISBN. This is a compute-intensive search.
Search service returns the album's ISBN, which your phone then looks up on Amazon Marketplace (another service).
Result is that you're taken to the Amazon product page.



SOA example: opinion profiling

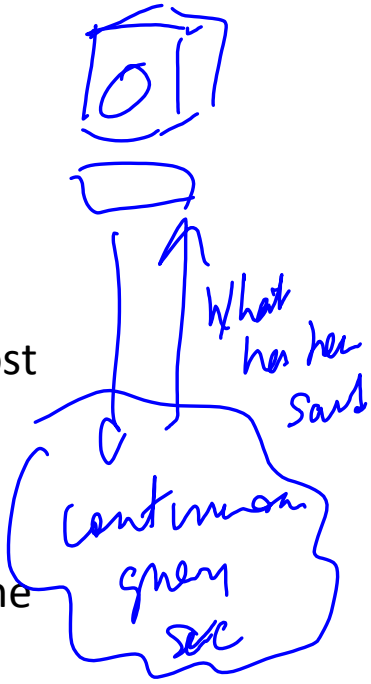
Monday, March 08, 2010
11:22 AM

SOA example: public opinion profiling

A corporation subscribes to a public-opinion data mining service.

Service calls return those posts to social networks that comment on the company and received the most responses/retweets/comments in the last day/week/month/etc. This is a compute-intensive service.

The company decides upon strategic responses to the posts.



Examples of advanced services

Continuous queries: track some specific data as it arrives, and be able to summarize it instantly upon request, e.g., stock market behavior, posts to social networks. Application: business decision support.

Demographic profiling: based upon one identity indicator (e.g., a web cookie), return the potential demographic profile of the user based upon past interactions with "bugged" sites. Application: targeted advertising.

Location-aware search: based upon some concept of the location of a user, sort search matches into order of distance from that location. Application: end-user support.

Services and Hadoop

- Hadoop is a **parallel computing environment**, not a service environment.
- So far, we've written a driver that supports **one computation at a time.**

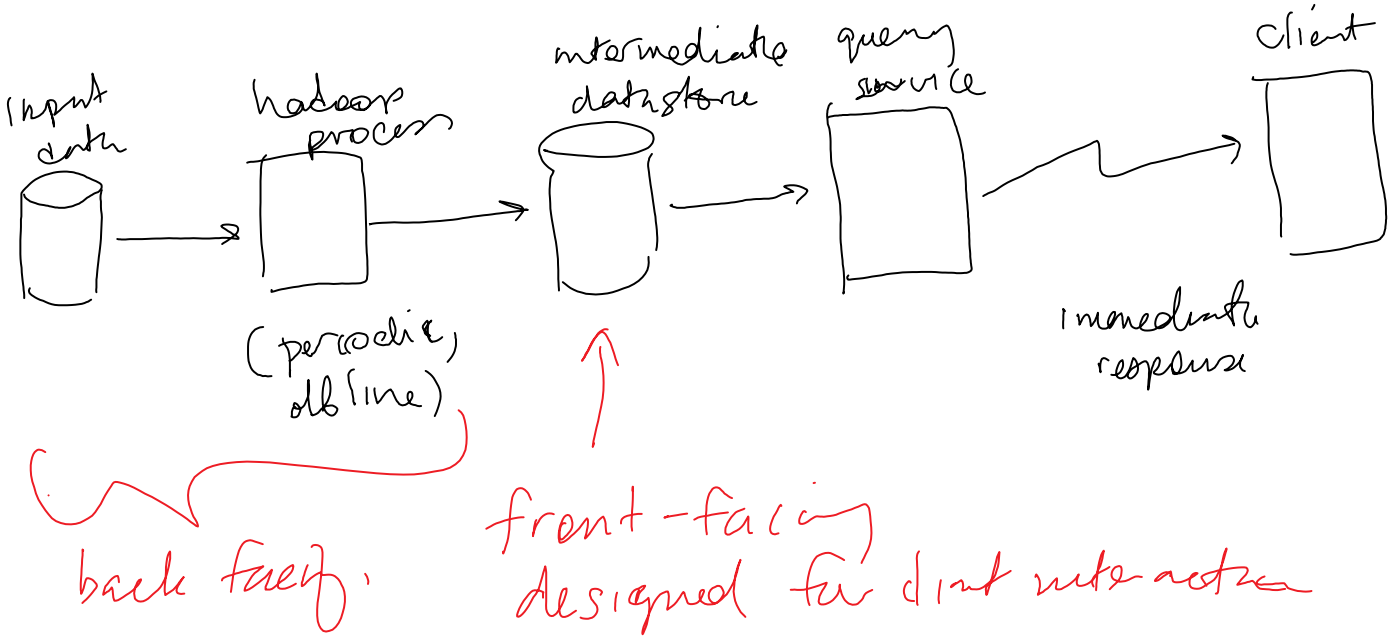
To make a hadoop computation into a service, one must provide for:

- Multiple concurrent queries
- Query independence.

Basically **no one does this in real time!**

Structure of a typical Hadoop service

Wednesday, March 09, 2011
12:50 PM



Two main issues for a Hadoop service

Wednesday, March 09, 2011
12:56 PM

Two main issues for a service

- What is the appropriate intermediate datastore to define a service? (this lecture)
- What is the appropriate way to define a service based upon the datastore? (next lecture)

A simple service

Monday, March 08, 2010
10:40 AM

A simple service: location awareness

Input: your IP address.

Output: your probable physical location in <lat,long> notation.

Mechanism: a database of IP addresses to <lat,long> pairs.

Theoretically, one can code this information into domain name service.

In actuality, few people do this, and one has to infer location from context, which is a long-term process.

Solution: a distributed datastore that maps IP addresses to <lat,long> pairs.

Two choices for a service

Wednesday, March 09, 2011
1:05 PM

Two design choices for a service:

- What is the front-end?
  - I.e., what will clients talk with?
- What is the back-end?
  - I.e., what will store data?

Kinds of service back-ends

Wednesday, March 09, 2011
1:01 PM

What is the back-end datastore for the service?

- Two main choices
  - SQL: MySQL, Oracle, Postgres, ...
  - NoSQL: Dynamo, BigTable, Voldemort, …
- Why? Answer concerns both:
  - assumptions about how the service behaves.
  - assumptions about service robustness.

Distributed or single-server?

Wednesday, March 09, 2011
1:08 PM

Most traditional services are based upon a single-server model

- No issues of data distribution.
- Typically SQL-based.

Cloud services are based upon a distributed storage model.

- Main issue is propogation of changes.
- Typically NoSQL-based.

The NoSQL controversy

NoSQL = "**N**ot **o**nly **S**tructured **Q**uery **L**anguage"

- Often misunderstood: some more radical authors interpret NoSQL as "Prevent Structured Query Language":)

In actuality, the NoSQL movement:

- Assumes that the majority of queries are simple key fetches that don't require SQL structures.
- Optimizes simple key fetches for quick response.
- Often contains a fallback to interpreting full SQL, with performance disadvantages.

A typical NoSQL service

Operations are:

ValueType get(KeyType key)

Assert the binding of a key to a value.
Subsequent puts update data.

int put(KeyType key, ValueType value)

Gets the last value associated with a key
(most of the time).

Roots of NoSQL

The most common use-case is **retrieval by key.**

Values are just bags of bits; the user can given them structure as needed.

The **CAP theorem**: one cannot build a distributed database system that exhibits all of **Consistency**, high **Availability**, and robustness in the presence of **Partitioning** (loss of messages).

Wednesday, March 09, 2011
1:15 PM

The CAP Theorem

- The CAP conjecture (Eric Brewer, 2000): any distributed datastore can only exhibit two of the following three properties
  - **Consistency**: all nodes have the same view of data.
  - (High) **Availability**: data is instantly available from the system.
  - **Partitionability**: the system continues to respond if messages are lost (due to system or network failure).
- Proved to be true by Gilbert and Lynch (2002).

In the cloud context:

- Very similar to my initial claim about the tension between consistency and concurrency.
- Main contribution: cloud datastores can be categorized in terms of the two (of three) properties C,A,P that they exhibit.
- AppEngine is in **class CP** = Consistency + Partitionability
- Only other reasonable cloud class is **class AP** = Availability + Partionability
- **We don't want a cloud datastore that loses data (e.g., ¬P)!**

Impact of the CAP theorem

Wednesday, March 09, 2011
1:56 PM

In the cloud context:

- CAP theorem claim is related to my initial claim about the tension between consistency and concurrency.
- Main contribution: cloud datastores can be categorized in terms of the two (of three) properties C,A,P that they exhibit.
- AppEngine JDO is in class **CP** = Consistency + Partitionability
- Only other reasonable class is **AP** = Availability + Partionability
- **We don't want a cloud datastore that loses data! (e.g., ¬P)**

The class AP

Wednesday, March 09, 2011
1:58 PM

The class AP contains datastores like:

- Amazon Dynamo
- Facebook's Cassandra
- LinkedIn's Project Voldemort

Why so many?

- Everybody needed one.
- Google didn't publish theirs.
- And it was CP, not AP.

How LinkedIn's colleague search actually works

Wednesday, March 09, 2011
2:02 PM

How LinkedIn's colleague search actually works

- Periodically, a Hadoop job is run to identify potential friends.
- (This is what you are writing for assignment 4)
- Output is stored to a Voldemort datastore.
- A web service accesses the datastore in read-only mode.

Some notes:

- Data changes slowly, so the Hadoop job only has to be done once/day or less.
- The Voldemort datastore is read-only once the job is done.
- So we don't need the C in CAP, because there are no consistency issues!
- So Voldemort, in class AP, suffices.

A fun challenge for you

Wednesday, March 09, 2011
2:05 PM

A fun challenge for you

LinkedIn's Hadoop job doesn't just find likely acquaintances ("2nd-order" acquaintances who share friends).

It also finds people you don't know who your friends might introduce you to ("3rd-order" acquaintances).

These are listed in the order of the number of contacts they have (so you can get introduced to someone who can introduce you to others).

See if you can code this up as a separate Pig script… extra credit for Assignment 4!

Inside Dynamo

An arbitrary choice…

Because it's very well documented…

What is dynamo?

Wednesday, March 09, 2011
2:17 PM

What is dynamo?

- The back-end cloud datastore for amazon.com itself.
- Serves requests for shopping carts, purchases.
- On an **eventually consistent** datastore…!
- How?

Unique features of dynamo

Wednesday, March 09, 2011
2:27 PM

Unique features of Dynamo

- Vector clocks for conflict detection.
- Business logic for conflict resolution.

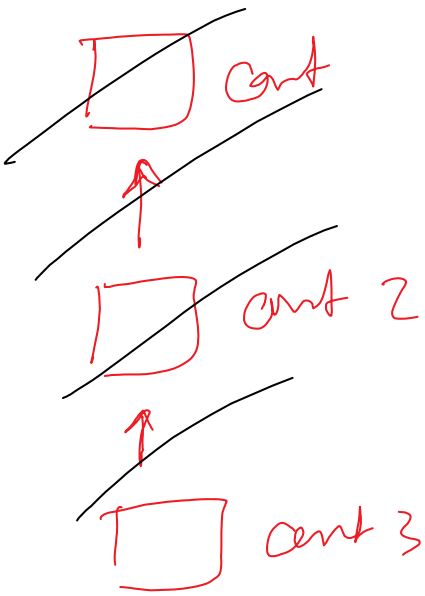
Wednesday, March 09, 2011
2:56 PM

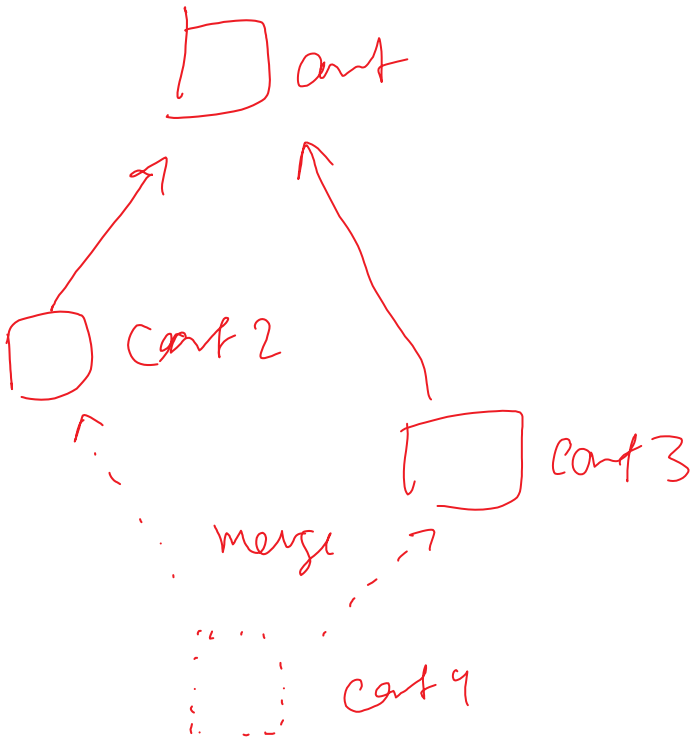
Vector clocks

- Every change transaction contains a timestamp and a pointer to the previous version of the object.
- As transactions flow through the system, they are accumulated in a local history on each node.
- Local history is
  - Pruned if there are no conflicts.
  - Resolved (in an application-specific manner) if conflicts occur.

Example: shopping cart

Wednesday, March 09, 2011
5:03 PM





Business-based recovery

- What happens when a conflict occurs is based upon business rules.
  - If object is a shopping cart, contents are merged.
  - If object is a purchase record, apparent duplicate purchases are eliminated.

Why?

Wednesday, March 09, 2011
3:16 PM

Why Amazon "gets away with" Dynamo

I told you in the second lecture that one would not want to store business data in an eventually consistent store.

Amazon "gets away with" doing that, by

- embedding business logic.
- using vector clock updates.
- for each kind of object, separately.

This is a complex game, which is why Dynamo isn't available to their customers.

Amazon SLA's

Wednesday, March 09, 2011
5:08 PM

Amusing fact: Amazon SLA's

We will respond in X seconds, 99.9% of the time
(and the .1% of the time when we don't do that,
is not specified)

Behind the scenes, that form of SLA has been shown
to save power!