Designing for the Cloud

So far, I've given you the "solution" and asked you to "program a piece of it".
This is a short-term problem!
In fact, 90% of the real human work in creating a cloud application is appropriate architectural design. Most of what I have asked you to program so far has been automated.
How to create a cloud

Requirements/criteria
   What (to do?)
   When (how fast?)
   Persistence, consistency, concurrency requirements.

Building blocks:
   Distributed Hash Tables (DHTs)
   Map/Reduce
   Services (that talk to DHTs or Map/Reduce)
"Waterfall model"

- Systems engineering: decide what platform to use, and what hardware.
- Requirements analysis: what should the product do.
- Design: how to do it.
- Implementation: a specific way to do it.
- Testing: make sure it works.
- Maintenance: keep it working in the presence of change.

Implementation is a 3rd generation concept.

- Java
- C++
- Python
- ML
- ...

Requirements are a 4th generation concept: tell what you want, just get it.

- Prolog
- Ops5 (theorem proving language)

The real 5th generation

natural language processing
don't have to specify the requirements in a machine readable form.

My joke

5th generation does away with testing, which means
all comp11 programmers are 5th generation:)
Systems engineering after requirements!

Need to know requirements to determine building blocks.

Design replaces implementation:

after we've specified design in machine-readable form, application writes itself.
DHTs

Distributed Hash Tables

Two functions

put(key, value)
get(key)

Keys are hashed to machines, and within machines to bins
Strengths

Natural replication: hash to subset of machines.
Very fast access: much faster than M/R.
High reliability.
Good for quick retrieval of something already stored.
Weaknesses

- Global queries are awkward.
- Indexing is only possible on primary key.
- Selection other than by key is linear time.
Tricks

DHTs are polymorphic: anything can be stored under a key.

(but you have to remember what the type is)
Never embed a persistent object inside another; embed a reference.
Ex: a shopping cart's contents are not necessarily contained in the DHT for the shopping cart; they're filed in the DHT themselves.
Never use lists when you play tricks with keys!
DHT example: shopping cart

Wednesday, March 16, 2011
4:52 PM
Very different behavior than DHTs
A waste if DHT lookup is what you want.
Map/Reduce strengths

At-scale computations (do something to 1,000,000 objects)
Cross-products: large-scale pairing.
Typical strategy: do at-scale with M/R, access with DHTs.
Map/Reduce weaknesses

Wednesday, March 16, 2011
3:53 PM

Weaknesses

Slow no matter what you ask for.
Relatively complex to think about and program.
(though, when you think about it, there aren't many unique M/R programs/patterns)
In software engineering, a pattern is a well-documented approach to a problem that includes:
- a template program with blanks to fill in for a particular application.
- a list of constraints as to what can go in the blanks.

Example: single-instance pattern
When you make a factory,
- create a self-initializing instance.
- use the instance instead of calling the constructor.
- so you only create one factory.

The blank: the thing you want to create.
What can fill the blank: any static object that is not stateful once created.
Patterns in Pig

FILTER

COGROUP-BY

...

Why I call them patterns?

Because the java code is itself a template with blanks, that are filled in by the arguments.
Kinds of cloud design units

DHT objects
DHT files
Distributed files
Distributing and M/R and concentrating take time.
  Distributing: turning a regular file into a distributed file (HDFS or AppEngine objects).
  Concentrating: turning a distributed file into a regular file.
DHT querying is quick.
  No matter how big a thing you're looking for.
DHT general search is slow.
  Anytime you have to search for something other than a primary key.
So, a lot of what we will do is to move search from M/R to DHTs!
We've already talked a bit about the social mining problem. But why was my "solution" the only reasonable one?
Ebay prices in search aren't the same as prices in the item page.
Search prices are updated every 5 minutes (background computation).
Item prices are real-time.
Why? Too expensive to index your search with high granularity.
What requirement makes it too expensive?