Announcements

- Recursive art contest entries due Monday 12/1 @ 11:59pm (new submission instructions)
- Homework 10 due Wednesday 12/3 @ 11:59pm
- Homework Party Monday 6pm-8pm in 2050 VLSB
- Ask homework questions in lab; both lab and homework are about SQL
- Quiz 3 released Wednesday, due Thursday 12/4 @ 11:59pm
- No videos for Lecture 38 on Friday 12/5
- Come to class and take the final survey
- There will be a screencast of live lecture (as always)
- Screencasts: http://goo.gl/hyUTca
- Final exam held on Thursday 12/18 3pm-6pm (review info later this week)

Computer Systems

Systems research enables the development of applications by defining and implementing abstractions:

- Operating systems provide a stable, consistent interface to unreliable, inconsistent hardware
- Networks provide a robust data transfer interface to constantly evolving communications infrastructure
- Databases provide a declarative interface to software that stores and retrieves information efficiently
- Distributed systems provide a unified interface to a cluster of multiple machines

A unifying property of effective systems:

Hide complexity, but retain flexibility

The Unix Operating System

Essential features of the Unix operating system (and variants):
- Portability: The same operating system on different hardware.
- Multi-Tasking: Many processes run concurrently on a machine.
- Plain Text: Data is stored and shared in text format.
- Modularity: Small tools are composed flexibly via pipes.

"We should have some ways of coupling programs like a garden hose – screw in another segment when it becomes necessary to massage data in another way," Doug McIlroy in 1964.

The standard streams in a Unix-like operating system are similar to Python iterators.

Python Programs in a Unix Environment

The built-in input function reads a line from standard input

The built-in print function writes a line to standard output

The sys.stdin and sys.stdout values provide access to the Unix standard streams as files

A Python file has an interface that supports iteration, read, and write methods

Using these "files" takes advantage of the operating system text processing abstraction

Big Data Processing

MapReduce is a framework for batch processing of big data.

- Framework: A system used by programmers to build applications
- Batch processing: All the data is available at the outset, and results aren't used until processing completes
- Big data: Used to describe data sets so large and comprehensive that they can reveal facts about a whole population, usually from statistical analysis

The MapReduce idea:

- Data sets are too big to be analyzed by one machine
- Using multiple machines has the same complications, regardless of the application/analysis
- Pure functions enable an abstraction barrier between data processing logic and coordinating a distributed application

MapReduce
MapReduce Evaluation Model

**Map phase:** Apply a mapper function to all inputs, emitting intermediate key-value pairs
- The mapper takes an iterable value containing inputs, such as lines of text
- The mapper yields zero or more key-value pairs for each input

Google MapReduce
Is a Big Data framework
For batch processing

**Reduce phase:** For each intermediate key, apply a reducer function to accumulate all values associated with that key
- The reducer takes an iterable value containing intermediate key-value pairs
- All pairs with the same key appear consecutively
- The reducer yields zero or more values, each associated with that intermediate key

Google MapReduce
Is a Big Data framework
For batch processing

Reduce phase: For each intermediate key, apply a reducer function to accumulate all values associated with that key
- The reducer takes an iterable value containing intermediate key-value pairs
- All pairs with the same key appear consecutively
- The reducer yields zero or more values, each associated with that intermediate key

**Mapper inputs are Lines of text provided to standard input**

**Python Example of a MapReduce Application**

The mapper and reducer are both self-contained Python programs
- They read from standard input and write to standard output

```
#!/usr/bin/env python
import sys
from mr import emit

for line in sys.stdin:
    emit_vowels(line)

def emit_vowels(line):
    for vowel in 'aeiou'
        count = line.count(vowel)
    if count > 0:
        emit(vowel, count)
```

**Execution Model**

1. **Input**
2. **Mapper**
3. **Shuffle**
4. **Reducer**
5. **Output**

**MapReduce Assumptions**

Constraints on the mapper and reducer:
- The mapper must be equivalent to applying a deterministic pure function to each input independently
- The reducer must be equivalent to applying a deterministic pure function to the sequence of values for each key

Benefits of functional programming:
- When a program contains only pure functions, call expressions can be evaluated in any order, lazily, and in parallel
- Referential transparency: a call expression can be replaced by its value (or vice versa) without changing the program

In MapReduce, these functional programming ideas allow:
- Consistent results, however computation is partitioned
- Re-computation and caching of results, as needed

**MapReduce Applications**

MapReduce Execution Model
Python Example of a MapReduce Application

The mapper and reducer are both self-contained Python programs
• They read from standard input and write to standard output

```python
#!/usr/bin/env python3

import sys
from mr import emit, values_by_key

Reducer
for key, value_iterator in values_by_key(sys.stdin):
    emit(key, sum(value_iterator))
```

Takes and returns iterators

Input: lines of text representing key-value pairs, grouped by key
Output: Iterator over (key, value_iterator) pairs that give all values for each key

```python
for key, value_iterator in values_by_key(sys.stdin):
    emit(key, sum(value_iterator))
```

MapReduce Benefits

What Does the MapReduce Framework Provide

Fault tolerance: A machine or hard drive might crash
• The MapReduce framework automatically re-runs failed tasks

Speed: Some machine might be slow because it's overloaded
• The framework can run multiple copies of a task and keep the result of the one that finishes first

Network locality: Data transfer is expensive
• The framework tries to schedule map tasks on the machines that hold the data to be processed

Monitoring: Will my job finish before dinner?!?
• The framework provides a web-based interface describing jobs

(Demo)