Lecture 8
Google's Map/Reduce

EN 600.320/420

Instructor: Randal Burns

5 March 2014

Department of Computer Science, *Johns Hopkins University*

Overview

- Map/Reduce
  - A structured programming model for data parallelism
  - Geared toward data intensive applications
- The Google File System
  - Large scale parallel file system for low $$
  - Failures are the normal case
- Open-source re-implementation
  - HADOOP: Map/Reduce
  - HDFS: Google File System

Systems that Use Map/Reduce

- Compile to Apache Hadoop!
  - PIG, Hive
- That build on HDFS
  - Hbase
- Systems that run Hadoop! on their storage
  - Cassandra, MongoDB, Aster, Oracle NoSQL

Map/Reduce Model

- Map
  - Process key/value pairs
  - Produce intermediate key/value pairs
- Reduce
  - Merge intermediate pairs on key value
- Map and reduce were LISP primitives
  - Hence the functional style
  - Actually write procedural code under functional constraints

Example: Count all Words

- This is pseudocode
  - Need a tokenizer, etc.

```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");
```

Example: Count all Words

```
reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

What can you do?

- Seems like a limited model
- But, many string processing problems fit naturally
- Can be used iteratively
- Examples
  - Grep
  - Web-log processing (e.g. URL hit counts)
  - Reverse Web-link graph
  - Terms vector per host
  - Build an inverted index
  - Distributed sort (naturally)

Types and Domains

- Map is a transformation
  - Input domain to output domain
- Reduce is a collection
  - No domain change
- C++ implementation is based all on strings
  - User code must convert to structured types

map	(k1,v1)	→ list(k2,v2)
reduce	(k2,list(v2))	→ list(v2)

The Why of Map/Reduce

- *How does programming in Map/Reduce help parallelism?*
  - Consider the count all words pseudo-code example

The Why of Map/Reduce

- *How does programming in Map/Reduce help parallelism?*
  - Consider the count all words pseudo-code example
- Potential parallelism
  - Map: on all the sites that store data
    - Or on all the sites to which you distribute data after accessing it from storage
  - Reduce: as many computers as there are terms

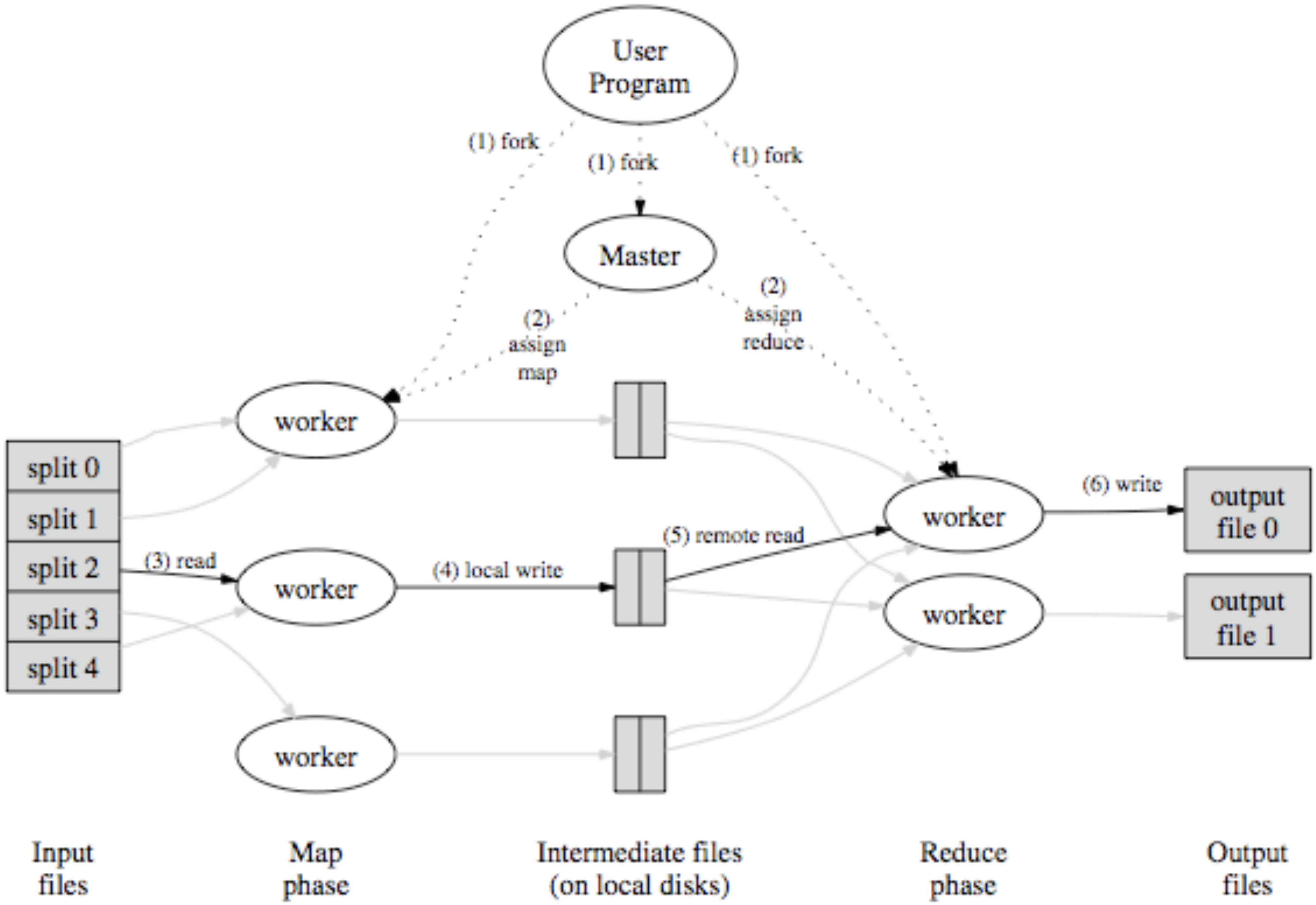
The Point

- Programs in Map/Reduce are automatically parallelized by the Map/Reduce runtime system

- No programmer interaction with parallelism
  - Except encoding problem well in Map/Reduce
- No explicit knowledge of
  - # of machines
  - Location of machines
- Scale up parallelism for arbitrary configurations

- (For embarassingly parallel problems?)

Map/Reduce Runtime



Runtime Properties

- Automatically partition input data
  - 16-64 MB chunks for Google
- Create *M* map tasks: one for each chunk
  - Assign available workers (up to *M*) to tasks
- Write intermediate pairs to local (to worker) disk
- *R* reduce tasks (defined by user) read and process intermediate results
- Output is *R* files available on shared file system
- Master tracks state
  - Asssignment of *M* map tasks and *R* reduce tasks to workers
  - State and liveliness of the above

System Issues

- What kind of problems arise with which the runtime system needs to deal?

System Issues

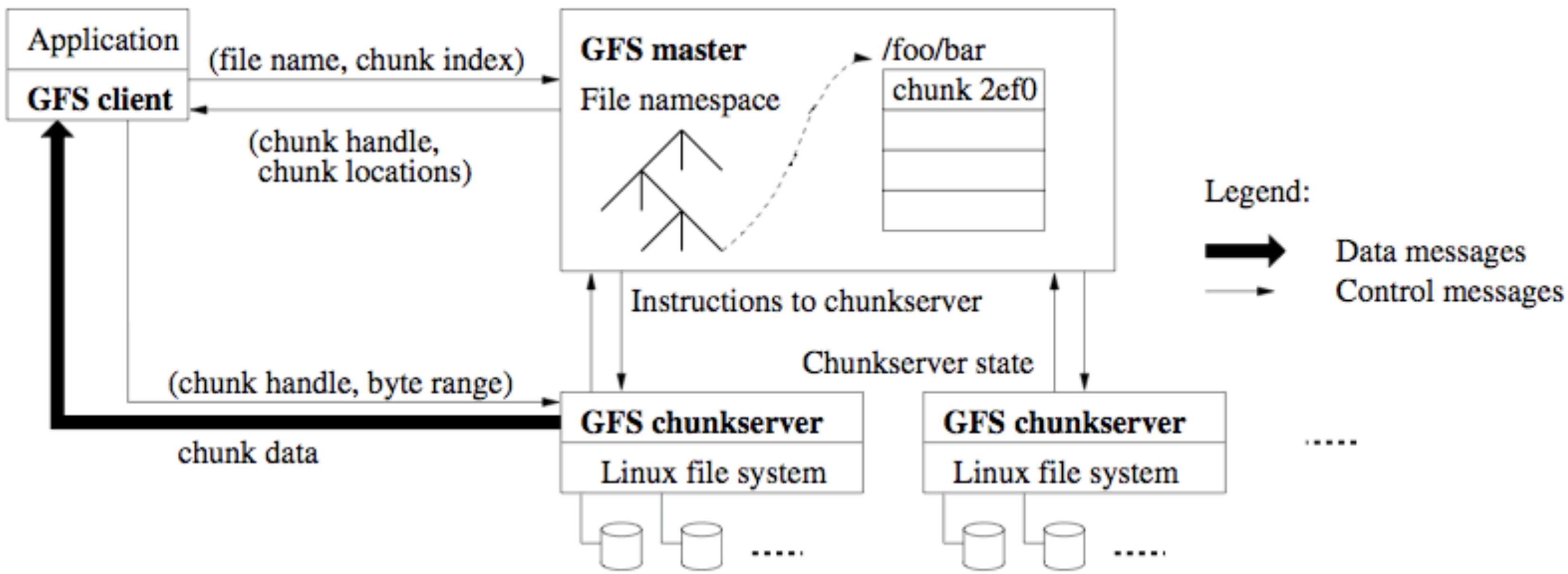
- Master failure
  - Checkpoint/restart, classic distributed systems/replication problem
- Failed worker
  - Heartbeat liveness detection, restart
- Slow worker
  - Backup tasks
- Locality of processing to data
  - Big deal, they don't really solve
- Task granularity
  - Metadata size and protocol scaling (not inherent parallelism) limit the size of *M* and *R*

Google File System

- GFS
- Same goals
  - Wide-distribution
  - Commodity hardware
  - High (aggregate) performance
- Different environment?
  - Component failures are normal behavior
  - Files are huge (specific to Google environment)
  - Most files have append-only writing

Architecture

- Looks like every other out-of-band FS



What's Cool

- Atomic checkpoint and append
  - Major mode for writing
  - Great semantics for limited usage
- In-memory metadata at Master
  - Gotta keep it small
- Re-replication
  - Keep three, detect missing on read/write