Google's MapReduce and Sawzall

CSCI 334
Stephen Freund

Computations Over Data
- Word Count
- Reverse Link Map
- Word Index
- Links out of a domain
- Page Rank
- Log file processing
- But.... many terabytes of data
  - 1 terabyte = 1000 Gigabytes
  = 1 099 511 627 776 bytes

Computing Infrastructure
- Millions of computers
- Distributed network of clusters around world
- Problems:
  - Need to coordinate computers
  - Machines fail constantly
  - Network, failure, computer/data locations, etc.
    should be transparent to user running analyses.
MapReduce and Sawzall

- MapReduce (Dean and Ghemawat)
  - Map/reduce from FP
  - distributed computer management

- Sawzall (Pike et al.)
  - language for writing code to perform data analysis

- Papers up on web page
- Cloud Compute Services:
  - Hadoop, Amazon EC2, IBM SmartCloud, ...

Summary

- Page Rank: 24 separate map-reduce operations
- Sawzall/MapReduce execution model:
  - specify data set, map fn, reduce fn
  - most map/reduce functions < 50 lines of code
  - hides details of distributed system
  - fault tolerant, fast, flexible architecture
# of Queries for Each Latitude/Longitude

proto "querylog.proto"

queries_per_degree: table sum[lat: int][lon: int] of int;

log_record: QueryLogProto = input;

loc: Location = locationinfo(log_record.ip);

emit queries_per_degree[int(loc.lat)][int(loc.lon)] <- 1;

map phase produces key-value pairs of form <(lat,lon),1>
reduce phase sums up values for each key

Page with Highest Page Rank

proto "document.proto"

max_pagerank_url:

table maximum(1) [domain: string] of url: string

weight pagerank: int;

doc: Document = input;

emit max_pagerank_url[domain(doc.url)] <- doc.url

weight doc.pagerank;

Dean & Ghemawat

We designed a new abstraction that allows us to express the simple computations we were trying to perform, but hides the messy details of parallelism, fault-tolerance, data distribution, and load-balancing. Our abstraction is inspired by the map and reduce primitives in Lisp and many other functional languages.

- Map: apply fn to every element of list
- Reduce: combine values in list to form a single "summary value"
  (eg: reduce list of nums by summing them)