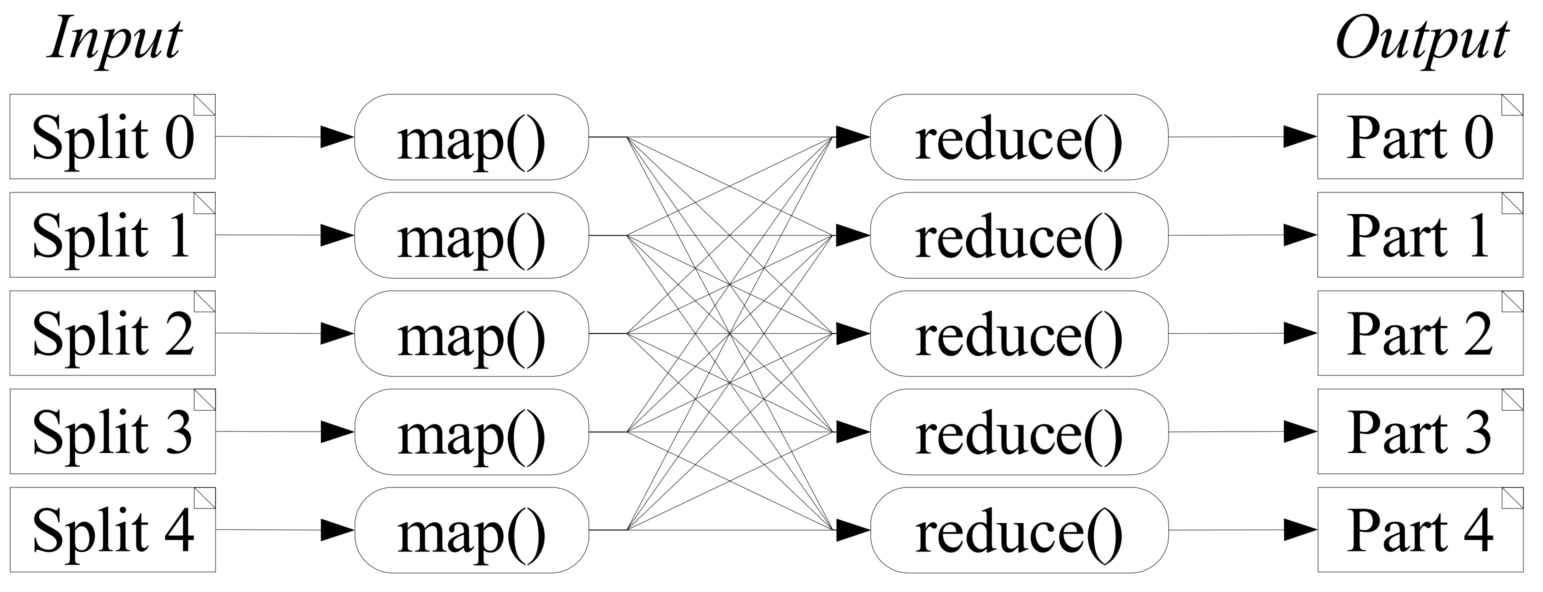
MapReduce in Nutch

Doug Cutting
20 July 2005

MapReduce: Background

- Invented by Google
  - http://labs.google.com/papers/mapreduce.html
- Platform for reliable, scalable computing.
- Implemented in Java as a part of Nutch
- Programmer specifies two primary methods:
  - map(k, v) → <k', v'>*
  - reduce(k', <v'>*) → <k', v'>*
  - also partition(), compare(), & others
- All v' with same k' are reduced together, in order.

MapReduce Diagram



MapReduce: Pros & Cons

- Not always a natural fit,
  - but, with moderate force, many things will fit.
- Not always optimal,
  - but not far off, and often cheaper in the end.
- Developing large-scale systems is expensive
- Shared platform:
  - minimizes development & debug time
  - maximizes optimizations, tools, etc.

Nutch Algorithms

- *inject* urls into a crawl db, to bootstrap it.
- loop:
  - *generate* a set of urls to fetch from crawl db;
  - *fetch* a set of urls into a *segment*;
  - *parse* fetched content of a segment;
  - *update* crawl db with data parsed from a segment.
- *invert links* parsed from segments
- *index* segment text & inlink anchor text

Data Structure: Crawl DB

- CrawlDb is a directory of files containing:

  <URL, CrawlDatum>

- CrawlDatum:

  <status, date, interval, failures, linkCount, ...>

- Status:

  {db_unfetched, db_fetched, db_gone,
  linked,
  fetch_success, fetch_fail, fetch_gone}

Algorithm: Inject

- MapReduce1: Convert input to DB format

  In: flat text file of urls

  Map(line) → <url, CrawlDatum>; status=db_unfetched

  Reduce() is identity;

  Output: directory of temporary files

- MapReduce2: Merge into existing DB

  Input: output of Step1 and existing DB files

  Map() is identity.

  Reduce: merge CrawlDatum's into single entry

  Out: new version of DB

Algorithm: Generate

- MapReduce1: select urls due for fetch

  In: Crawl DB files

  Map() → if date≥now, invert to <CrawlDatum, url>

  Partition by value hash (!) to randomize

  Reduce:

  - compare() order by decreasing CrawlDatum.linkCount
  - output only top-N most-linked entries

- MapReduce2: prepare for fetch

  Map() is invert; Partition() by host, Reduce() is identity.

  Out: Set of <url,CrawlDatum> files to fetch in parallel

Algorithm: Fetch

- MapReduce: fetch a set of urls
  - In: <url,CrawlDatum>, partition by host, sort by hash
  - Map(url,CrawlDatum) → <url, FetcherOutput>
    - multi-threaded, async map implementation
    - calls existing Nutch protocol plugins
  - FetcherOutput: <CrawlDatum, Content>
  - Reduce is identity
  - Out: two files: <url,CrawlDatum>, <url,Content>

Algorithm: Parse

- MapReduce: parse content

  In: <url, Content> files from Fetch

  Map(url, Content) → <url, Parse>
  calls existing Nutch parser plugins

  Reduce is identity.

  Parse: <ParseText, ParseData>

  Out: split in three: <url,ParseText>, <url,ParseData> and <url,CrawlDatum> for outlinks.

Algorithm: Update Crawl DB

- MapReduce: integrate fetch & parse out into db
  - In: <url,CrawlDatum> existing db plus fetch & parse out
  - Map() is identity
  - Reduce() merges all entries into a single new entry
    - overwrite previous db status w/ new from fetch
    - sum count of links from parse w/ previous from db
  - Out: new crawl db

Algorithm: Invert Links

- MapReduce: compute inlinks for all urls
  - In: <url,ParseData>, containing page outlinks
  - Map(srcUrl, ParseData> → <destUrl, Inlinks>
    - collect a single-element Inlinks for each outlink
    - limit number of outlinks per page
  - Inlinks: <srcUrl, anchorText>*
  - Reduce() appends inlinks
  - Out: <url, Inlinks>, a complete link inversion

Algorithm: Index

- MapReduce: create Lucene indexes
  - In: multiple files, values wrapped in <Class, Object>
    - <url, ParseData> from parse, for title, metadata, etc.
    - <url, ParseText> from parse, for text
    - <url, Inlinks> from invert, for anchors
    - <url, CrawlDatum> from fetch, for fetch date
  - Map() is identity
  - Reduce() create a Lucene Document
    - call existing Nutch indexing plugins
  - Out: build Lucene index; copy to fs at end

MapReduce Extensions

- Split output to multiple files
  - saves subsequent i/o, since inputs are smaller
- Mix input value types
  - saves MapReduce passes to convert values
- Async Map
  - permits multi-threaded Fetcher
- Partition by Value
  - facilitates selecting subsets w/ maximum key values

Summary

- Nutch's major algorithms converted in 2 weeks.
- Before:
  - many were undistributed scalabilty bottlenecks
  - distributable algorithms were complex to manage
  - collections larger than 100M pages impractical
- After:
  - all are scalable, distributed, easy to operate
  - code is substantially smaller & simpler
  - should permit multi-billion page collections