

Distributed Index

Distributed indexing

- For web-scale indexing (don't try this at home!):
 - must use a distributed computing cluster
- Individual machines are fault-prone
 - Can unpredictably slow down or fail
- How do we exploit such a pool of machines?

Web search engine data centers

- Web search data centers (Google, Bing, Baidu) mainly contain commodity machines.
- Data centers are distributed around the world.
- Estimate: Google ~1 million servers, 3 million processors/cores (Gartner 2007)

Distributed indexing

- Maintain a *master* machine directing the indexing job.
- Break up indexing into sets of (parallel) tasks.
- Master machine assigns each task to an idle machine from a pool.

Parallel tasks

- We will use two sets of parallel tasks
 - Parsers
 - Inverters
- Break the input document collection into *splits*
- Each split is a subset of documents

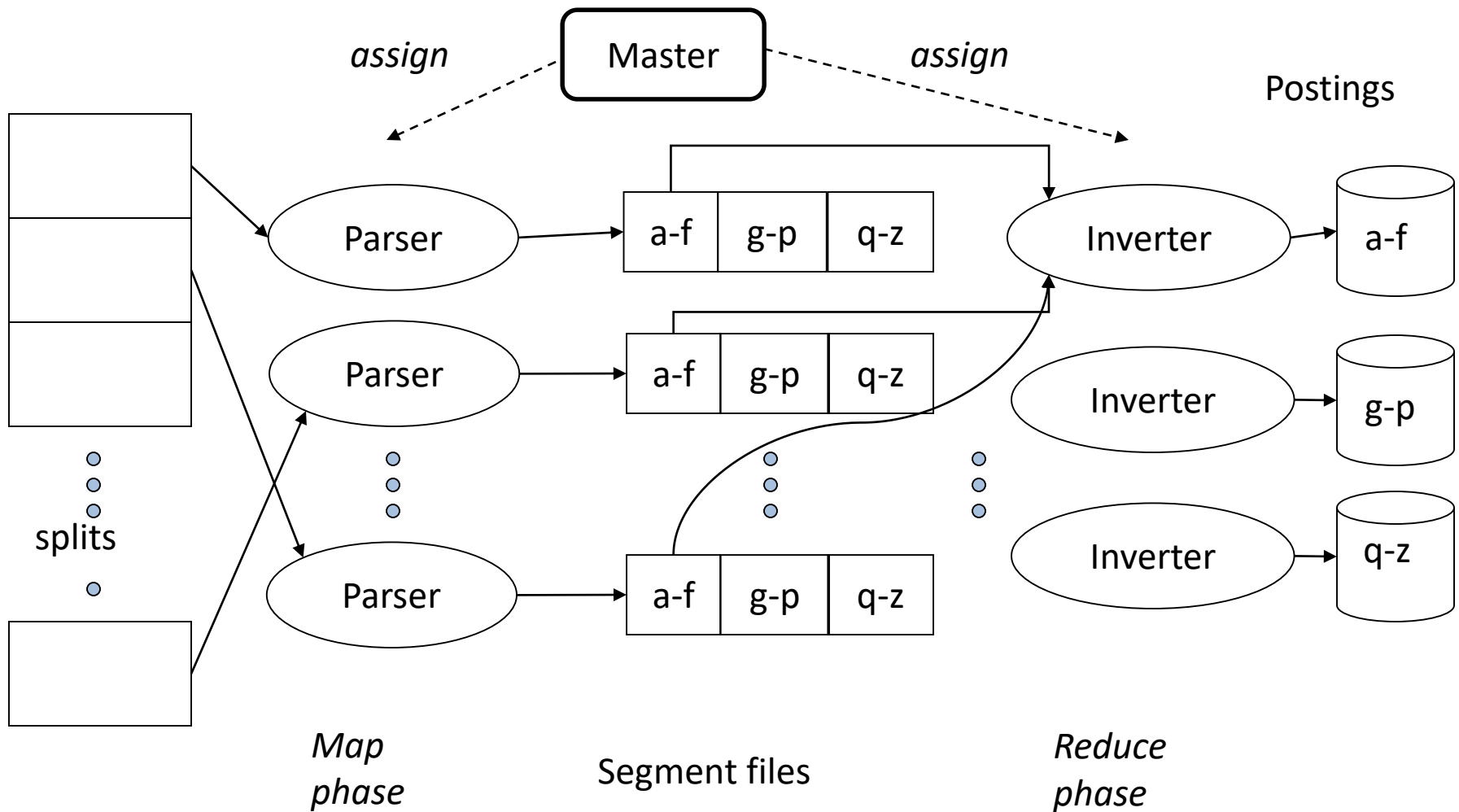
Parsers

- Master assigns a split to an idle parser machine
- Parser reads a document at a time and emits (term, doc) pairs
- Parser writes pairs into j partitions
- Each partition is for a range of terms' first letters
 - (e.g., ***a-f***, ***g-p***, ***q-z***) – here $j = 3$.
- Now to complete the index inversion

Inverters

- An inverter collects all (term,doc) pairs (= postings) for one term-partition.
- Sorts and writes to postings lists

Data flow



MapReduce

- The index construction algorithm we just described is an instance of *MapReduce*.
- MapReduce (Dean and Ghemawat 2004) is a robust and conceptually simple framework for distributed computing without having to write code for the distribution part.
- They describe the Google indexing system (ca. 2002) as consisting of a number of phases, each implemented in MapReduce.

Dynamic indexing

- Up to now, we have assumed that collections are static.
- They rarely are:
 - Documents come in over time and need to be inserted.
 - Documents are deleted and modified.
- This means that the dictionary and postings lists have to be modified:
 - Postings updates for terms already in dictionary
 - New terms added to dictionary

Simplest approach

- Maintain “big” main index
- New docs go into “small” auxiliary index
- Search across both, merge results
- Deletions
 - Invalidation bit-vector for deleted docs
 - Filter docs output on a search result by this invalidation bit-vector
- Periodically, re-index into one main index

Merging main and auxiliary indexes

- Merging of the auxiliary index into the main index is efficient if we keep a separate file for each postings list.
- Merge is the same as a simple append.
- But then we would need a lot of files – separate file for each word, inefficient for OS.
- In reality: Use a scheme somewhere in between (e.g., split very large postings lists, collect postings lists of length 1 in one file etc.)

Dynamic indexing at search engines

- All the large search engines now do dynamic indexing
- Their indices have frequent incremental changes
 - News items, blogs, new topical web pages
- But (sometimes/typically) they also periodically reconstruct the index from scratch
 - Query processing is then switched to the new index, and the old index is deleted