CSx55: Distributed Systems [MapReduce]

To Orchestrate a Job in a Cluster

A job comprises many a task
What could be so hard, you ask?
A job's done, when every task wraps up
Circumventing every hiccup
Machines may slowdown or go bust
For no reason nor rhyme
Try to complete, you must
All tasks, at roughly the same time

Shrideep Pallickara
Computer Science
Colorado State University

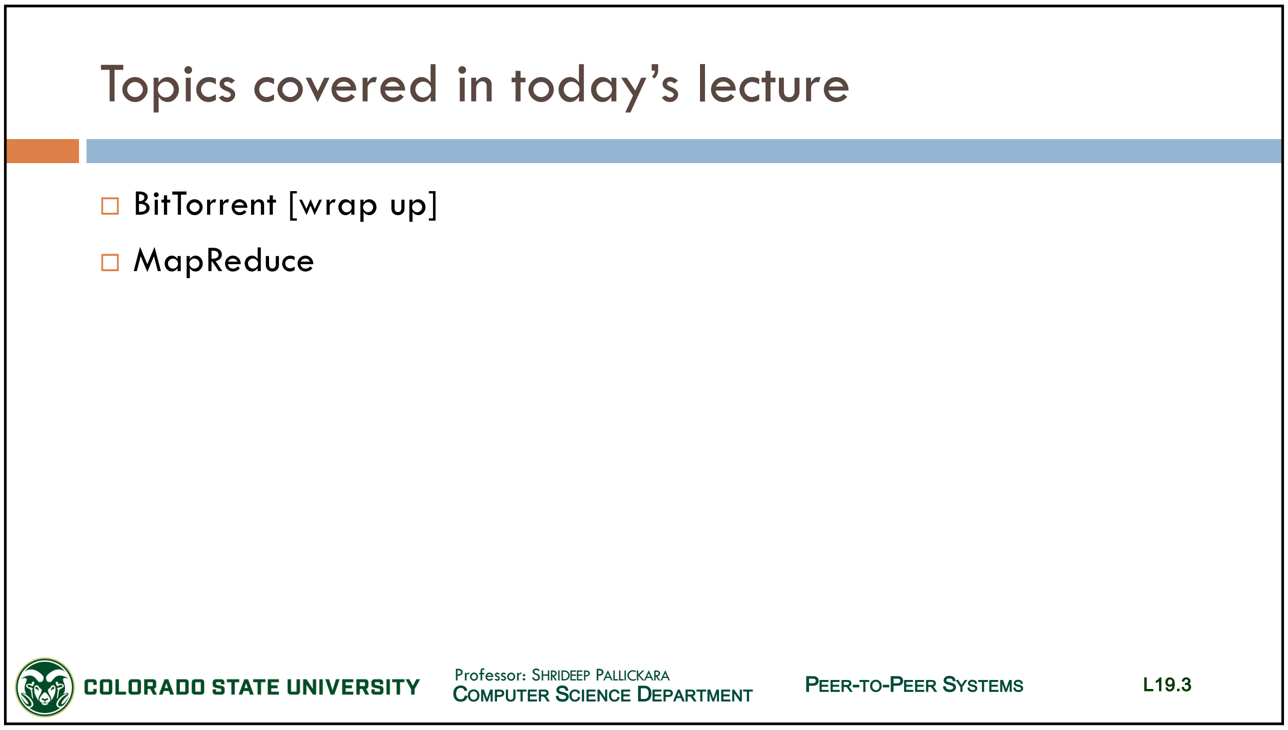
COMPUTER SCIENCE DEPARTMENT   COLORADO STATE UNIVERSITY

1

Frequently asked questions from the previous class survey

- Turning off multicasting? Who does this? Still used?
- During gossiping in unstructured P2P systems, is it possible for a search to dead end?
- In an unstructured P2P system, is it possible for the search to continue even though you have found what you are looking for?
- Do systems such as as Cassandra use gossiping?
- Chunk sizes in BitTorrent?

2



Topics covered in today's lecture

- BitTorrent [wrap up]
- MapReduce

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L19.3

3

BITTORRENT [WRAP-UP]

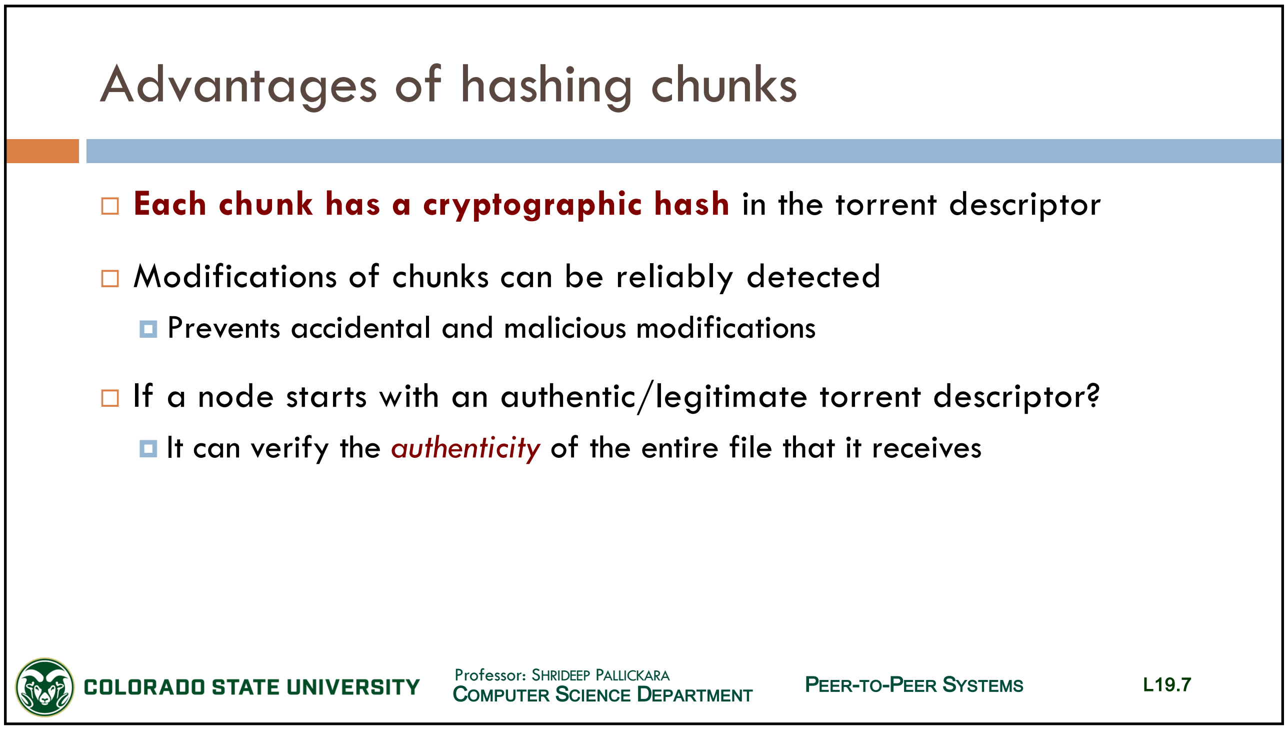
4

BitTorrent: Protocol summary

- Splits files into fixed-sized **chunks**
- Chunks are then made available at various peers across the P2P network
- Clients can *download* a number of chunks **in parallel** from different sites
  - Reduces the burden on a particular peer to service the entire download

The BitTorrent protocol

- When a file is made available in BitTorrent, a `.torrent` file is created
  - Holds **metadata** associated with that file
- Metadata
  - The name and length of the file
  - Location of a **tracker** (URL)
    - Centralized server that manages download for that file
  - Checksum
    - Associated with each chunk
    - Generated using the SHA-1 algorithm

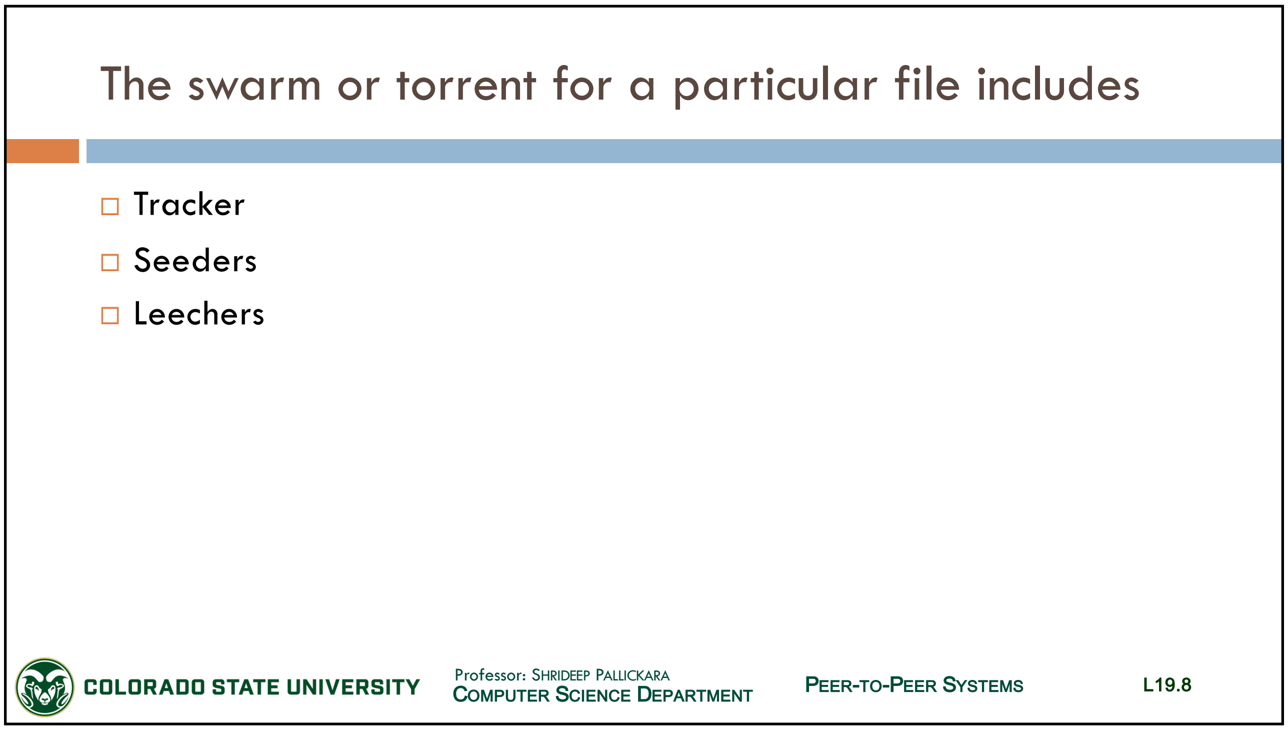


Advantages of hashing chunks

- **Each chunk has a cryptographic hash** in the torrent descriptor
- Modifications of chunks can be reliably detected
  - Prevents accidental and malicious modifications
- If a node starts with an authentic/legitimate torrent descriptor?
  - It can verify the *authenticity* of the entire file that it receives

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L19.7

7



The swarm or torrent for a particular file includes

- Tracker
- Seeders
- Leechers

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L19.8

8

Trackers

- The use of trackers, compromises a core P2P principle
  - But *simplifies* the system
- Trackers are responsible for **tracking the download status** for a particular file

The roles of participants in BitTorrent: Seeder

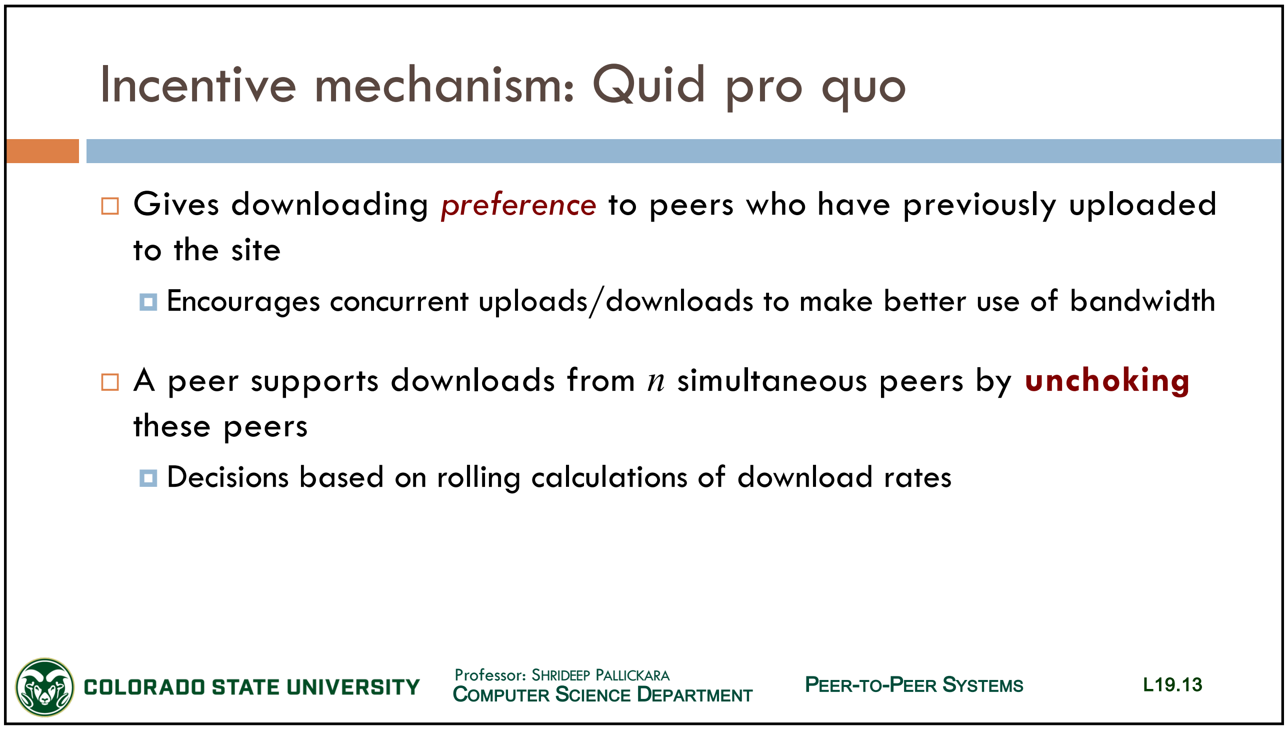
- Peer with a complete version of a file (i.e., with all its chunks) is known as a **seeder**
- Peer that initially creates the file, provides the initial seed for file distribution

The roles of participants in BitTorrent: Leechers

- Peers that want to download a file are known as **leechers**
  - A given leecher, at any given time, contains a number of chunks for that file
- Once a leecher downloads all chunks for a file, it can become a seeder for subsequent downloads
  - Files **spread virally** based on demand

When a peer wants to download a file

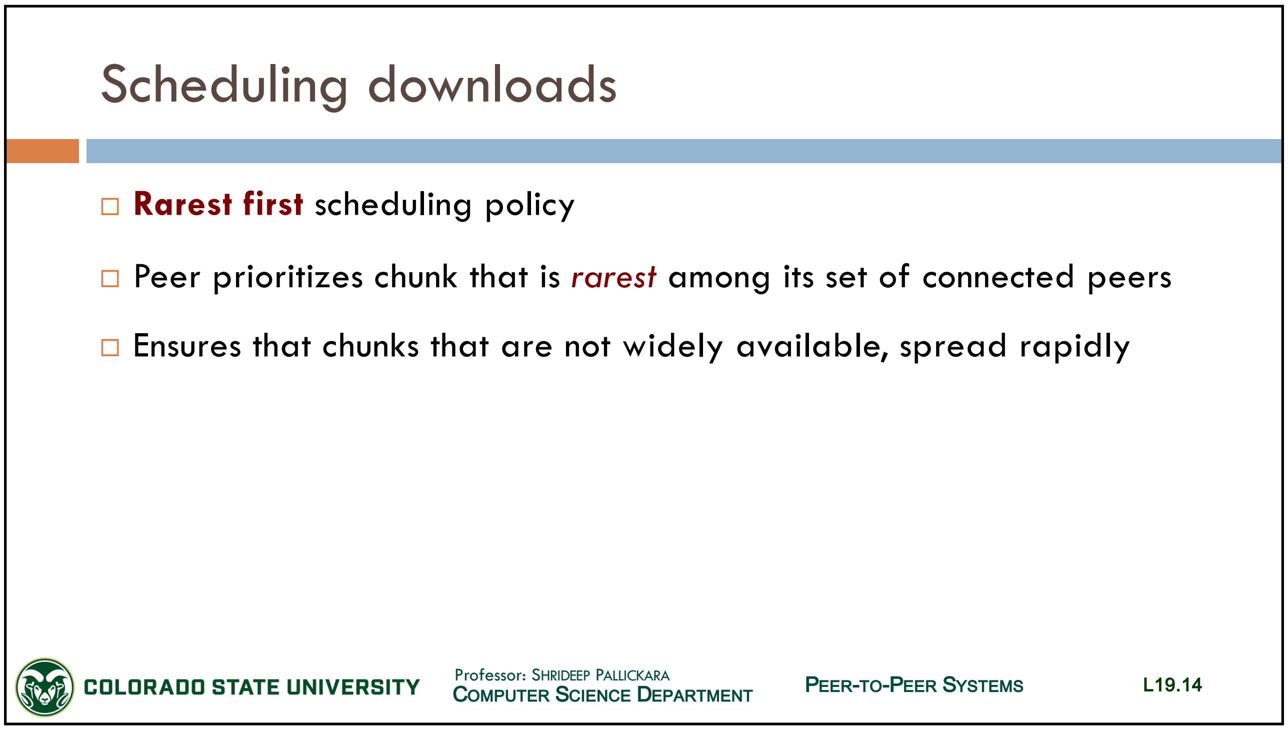
- Contacts the tracker
- Is given a **partial view** of the torrent
  - The set of peers that can support the download
  - The tracker does not participate in scheduling the downloads
    - Decentralized
- Chunks are requested and transmitted in **any order**



Incentive mechanism: Quid pro quo

- Gives downloading *preference* to peers who have previously uploaded to the site
  - Encourages concurrent uploads/downloads to make better use of bandwidth
- A peer supports downloads from n simultaneous peers by **unchoking** these peers
  - Decisions based on rolling calculations of download rates

13



Scheduling downloads

- **Rarest first** scheduling policy
- Peer prioritizes chunk that is *rarest* among its set of connected peers
- Ensures that chunks that are not widely available, spread rapidly

14

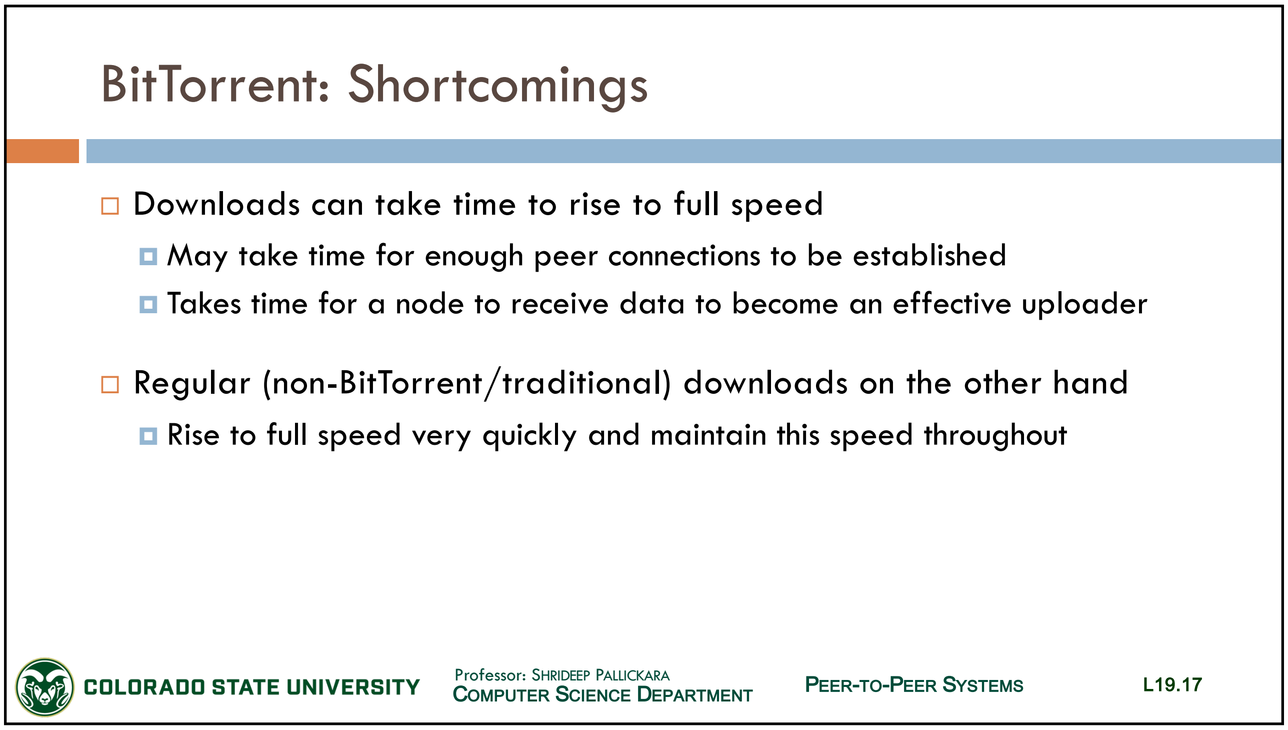
How BitTorrent differs from a classic download

	BitTorrent	**Classic download**
Connections	Many small data requests over different IP connections to different machines	One TCP connection to one machine
Download Order	Random or "rarest first" to ensure high-availability	Sequential

** Allows BitTorrent to achieve lower cost, higher redundancy, and resistance to abuse

BitTorrent: Advantages

- Advantages
  - Lower costs, greater redundancy, higher resistance to abuse or "flash crowds"
- Shortcomings
  - Non-contiguous download precludes progressive download
  - No streaming playback
    - Beta BitTorrent Streaming protocol was made available for testing in 2013; this was not successful
    - A service BitTorrent Live was released as Public Beta in Spring 2019

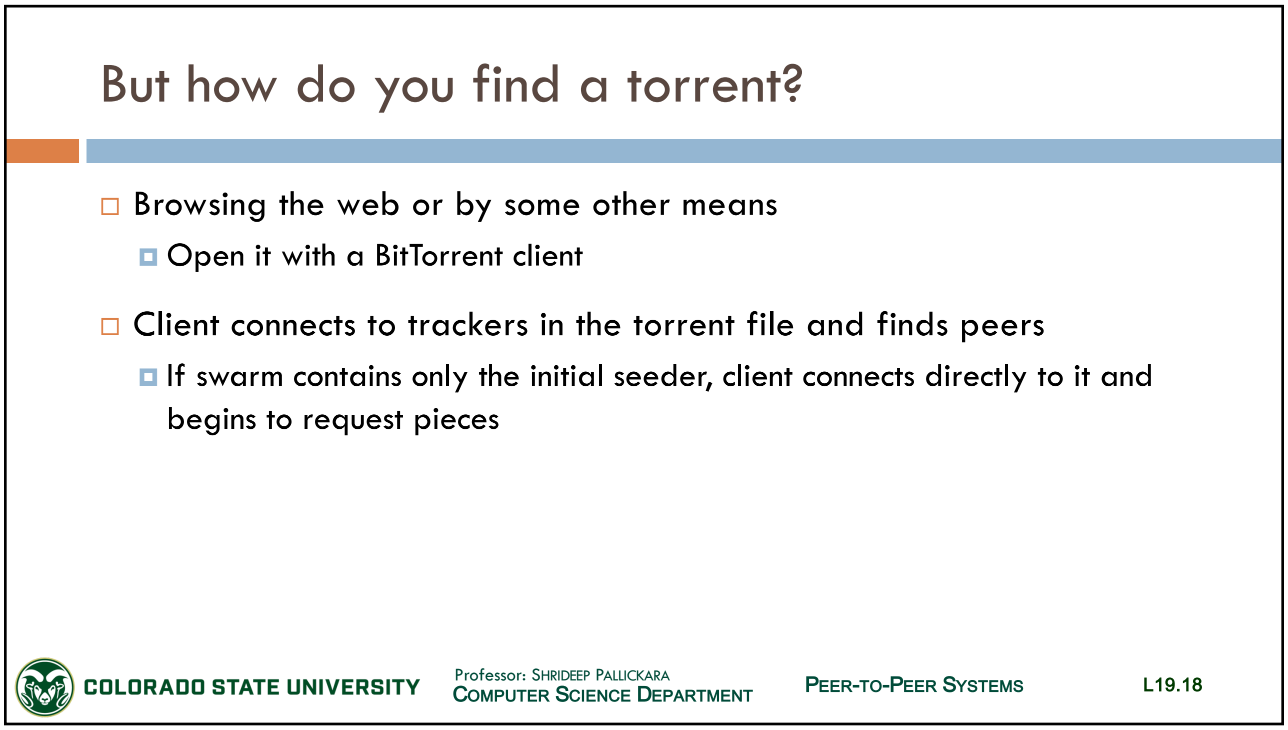


BitTorrent: Shortcomings

- Downloads can take time to rise to full speed
  - May take time for enough peer connections to be established
  - Takes time for a node to receive data to become an effective uploader
- Regular (non-BitTorrent/traditional) downloads on the other hand
  - Rise to full speed very quickly and maintain this speed throughout

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L19.17

17

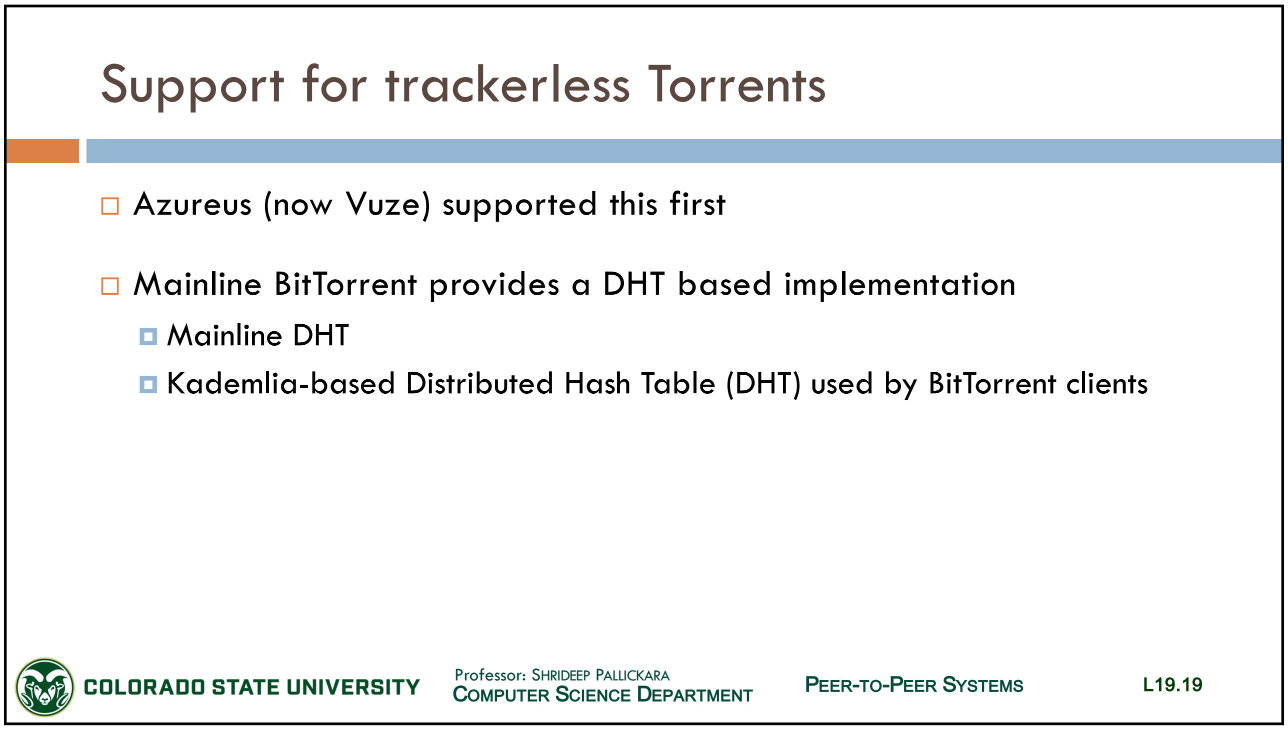


But how do you find a torrent?

- Browsing the web or by some other means
  - Open it with a BitTorrent client
- Client connects to trackers in the torrent file and finds peers
  - If swarm contains only the initial seeder, client connects directly to it and begins to request pieces

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L19.18

18



Support for trackerless Torrents

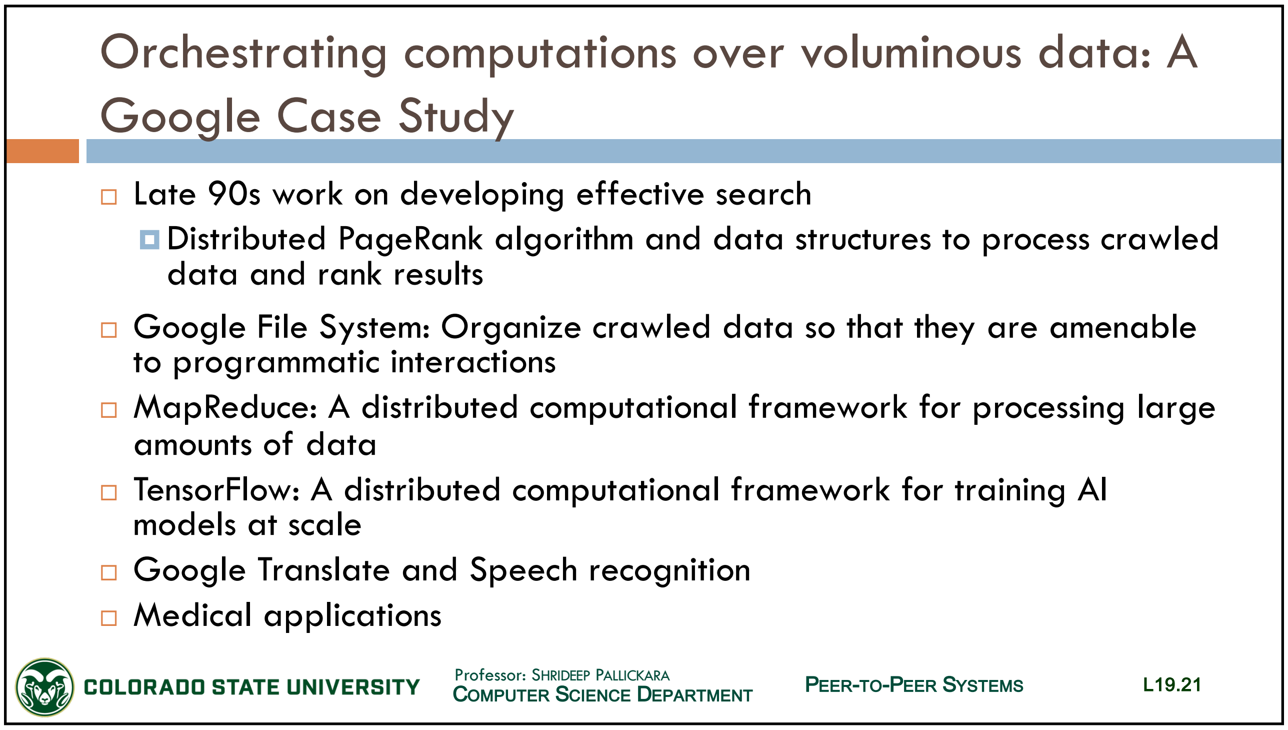
- Azureus (now Vuze) supported this first
- Mainline BitTorrent provides a DHT based implementation
  - Mainline DHT
  - Kademlia-based Distributed Hash Table (DHT) used by BitTorrent clients

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L19.19

19

MAPREDUCE

20

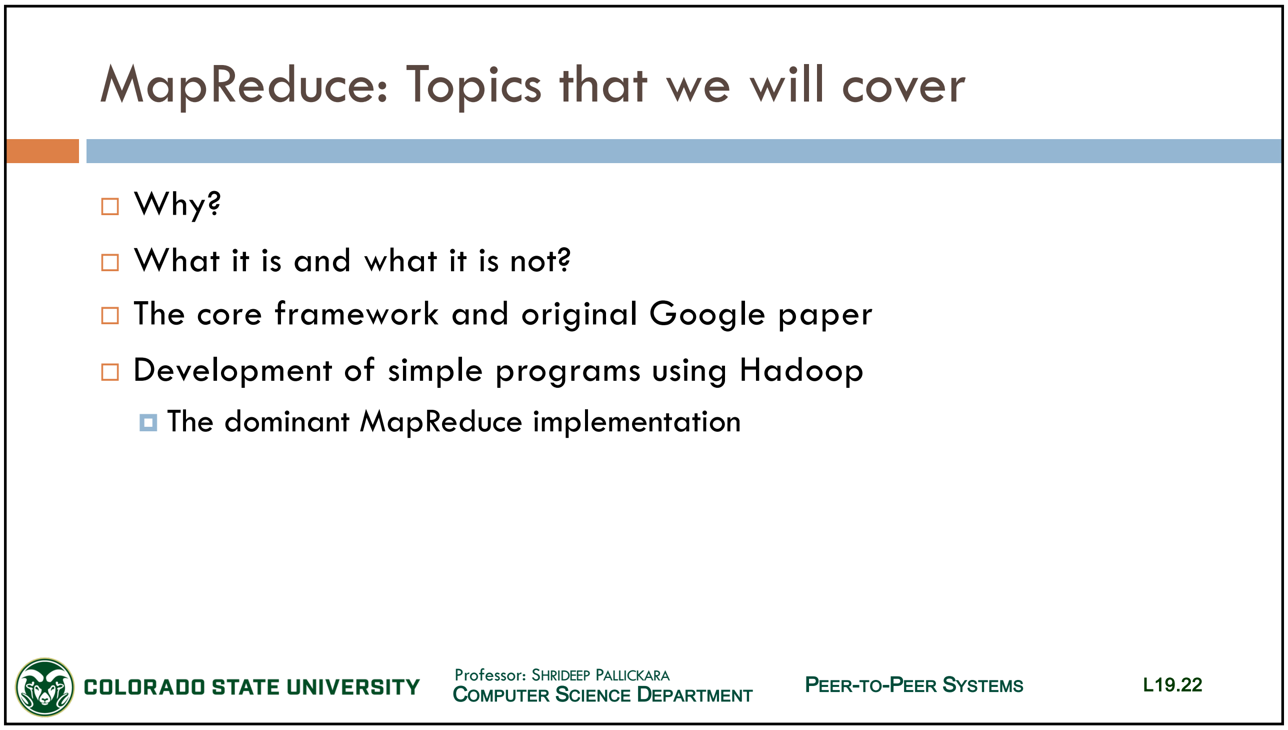


Orchestrating computations over voluminous data: A Google Case Study

- Late 90s work on developing effective search
  - Distributed PageRank algorithm and data structures to process crawled data and rank results
- Google File System: Organize crawled data so that they are amenable to programmatic interactions
- MapReduce: A distributed computational framework for processing large amounts of data
- TensorFlow: A distributed computational framework for training AI models at scale
- Google Translate and Speech recognition
- Medical applications

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L19.21

21

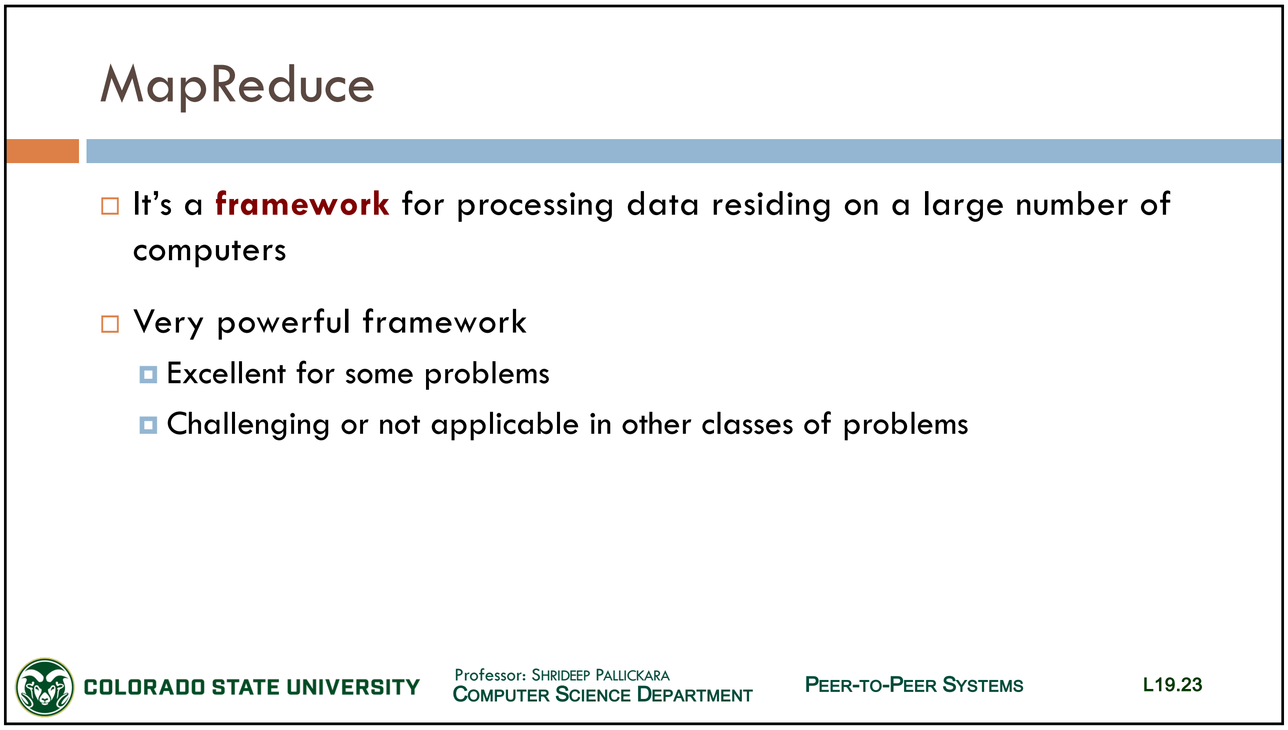


MapReduce: Topics that we will cover

- Why?
- What it is and what it is not?
- The core framework and original Google paper
- Development of simple programs using Hadoop
  - The dominant MapReduce implementation

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L19.22

22

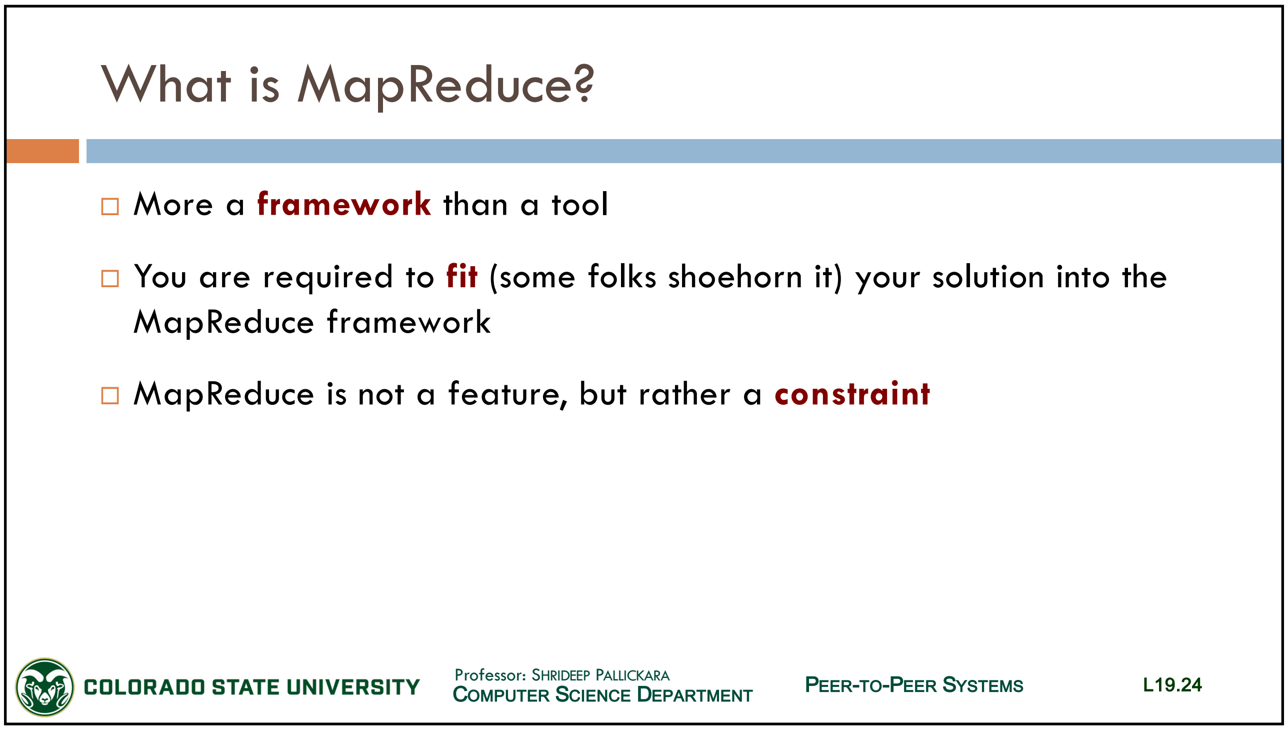


MapReduce

- It's a **framework** for processing data residing on a large number of computers
- Very powerful framework
  - Excellent for some problems
  - Challenging or not applicable in other classes of problems

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L19.23

23



What is MapReduce?

- More a **framework** than a tool
- You are required to **fit** (some folks shoehorn it) your solution into the MapReduce framework
- MapReduce is not a feature, but rather a **constraint**

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L19.24

24

What does this constraint mean?

- It makes problem solving easier *and* harder
- Clear boundaries for what you can and cannot do
  - You actually need to consider **fewer options** than what you are used to
- But solving problems with constraints requires planning and a change in your thinking

But what does this get us?

- Tradeoff of being confined to the MapReduce framework?
  - Ability to process data on a large number of computers
  - But, more importantly, **without having to worry about** concurrency, scale, fault tolerance, and robustness

A challenge in writing MapReduce programs

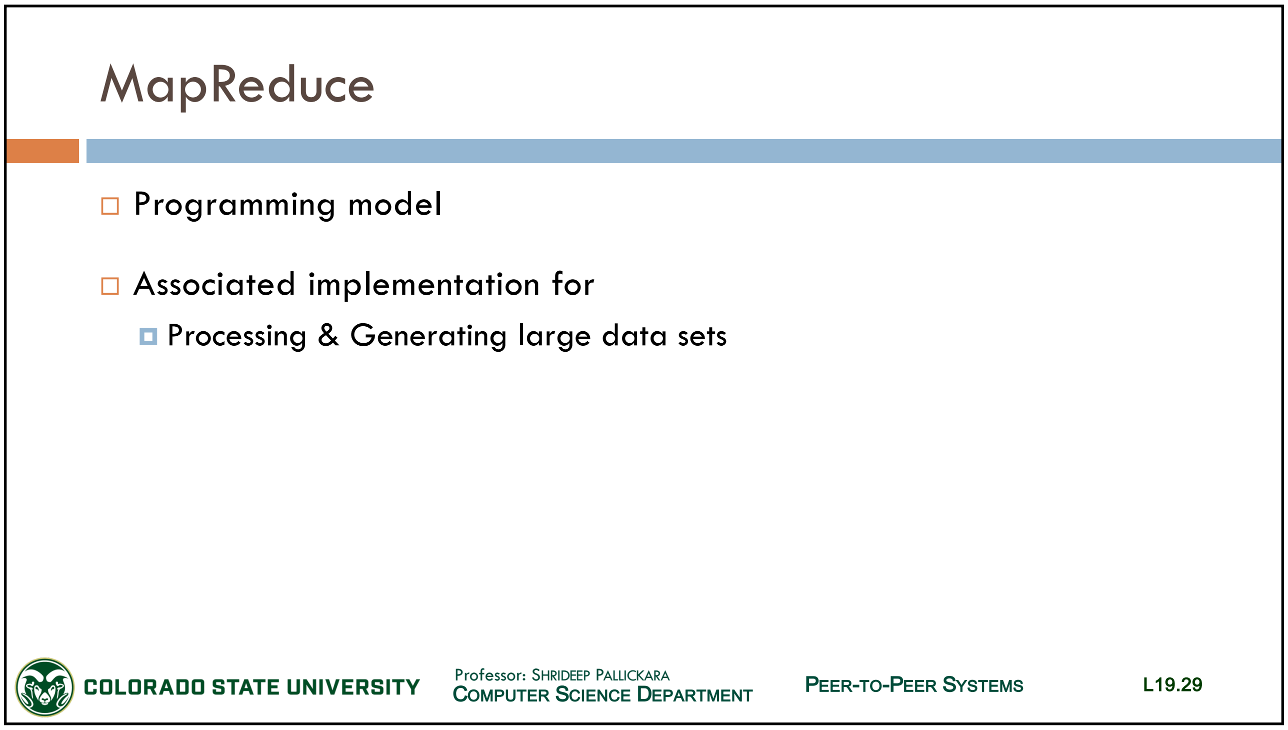
- **Design**!
  - Good programmers can produce bad software due to poor design
  - Good programmers can produce bad MapReduce algorithms
- Only in this case your **mistakes will be amplified**
  - Your job may be distributed on 100s or 1000s of machines and operating on a Petabyte of data

27

MAPREDUCE

MATERIALS BASED ON
JEFFREY DEAN and SANJAY GHEMAWAT: *MapReduce: Simplified Data Processing on Large Clusters*. OSDI 2004: 137-150

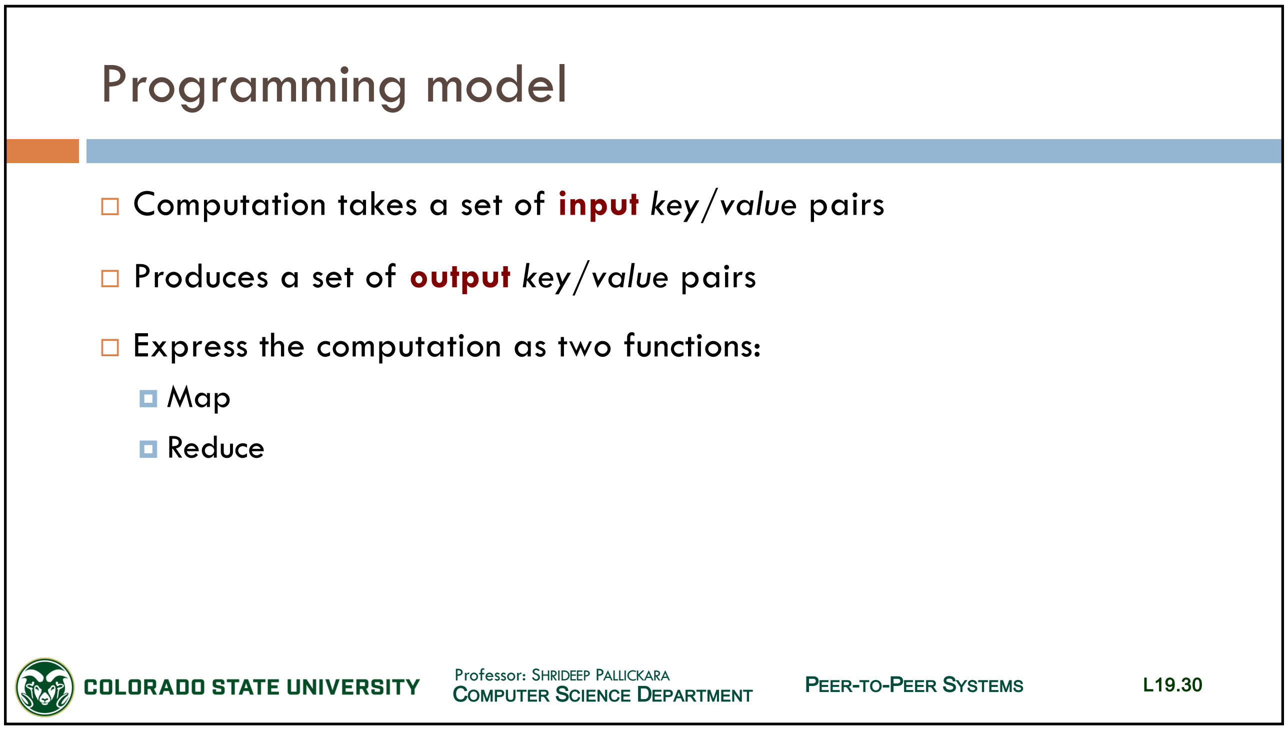
28



MapReduce

- Programming model
- Associated implementation for
  - Processing & Generating large data sets

29



Programming model

- Computation takes a set of **input** *key/value* pairs
- Produces a set of **output** *key/value* pairs
- Express the computation as two functions:
  - Map
  - Reduce

30

Map

- Takes an input pair
- Produces a set of intermediate key/value pairs

31

Mappers

- If map operations are **independent** of each other they can be performed in parallel
  - **Shared nothing**
- This is usually the case

32

MapReduce library

- **Groups** all intermediate values with the same intermediate key
- **Passes** them to the Reduce function

33

Reduce function

- Accepts intermediate *key* I and
  - Set of *values* for that *key*
- **Merge** these *values* together to get
  - Smaller set of *value*

34

Counting number of occurrences of each word in a large collection of documents

```
map (String key, String value)
     //key: document name
     //value: document contents

   for each word w in value
      EmitIntermediate(w, “1”)
```

Counting number of occurrences of each word in a large collection of documents

```
reduce (String key, Iterator values)
     //key: a word
    //value: a list of counts

    int result = 0;
    for each v in values
       result += ParseInt(v);
    Emit(AsString(result));
```

Sums together all counts emitted for a particular word

MapReduce specification object contains

- Names of
  - Input
  - Output
- Tuning parameters

37

Map and reduce functions have associated types drawn from different domains

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

38

What's passed to-and-from user-defined functions?

- Strings
- User code converts between
  - String
  - Appropriate types

39

EXAMPLES

40

Programs expressed as MapReduce computations: Distributed Grep

- Map
  - Emit line if it matches specified pattern
- Reduce
  - Just copy intermediate data to the output
    - The reducer here is an identity function

Counts of URL access frequency

- Map
  - Process logs of web page requests
  - Output `<URL, 1>`
- Reduce
  - Add together all values for a particular URL
  - Output `<URL, total count>`

Reverse Web-link Graph

- Map
  - Outputs <`target, source`> pair for each `target` URL found in page `source`
- Reduce
  - Concatenate list of all sources for a `target` URL
  - Output <`target, ` *`list`*`(source)`>

Term-Vector per Host

- Summarizes important terms that occur in a set of documents <`word, frequency`>
- For each input document, the Map
  - Emits <`hostname, term vector`>
- Reduce function
  - Has all per-document vectors for a given host
  - Add term vectors; discard away infrequent terms
    - <`hostname, term vector`>

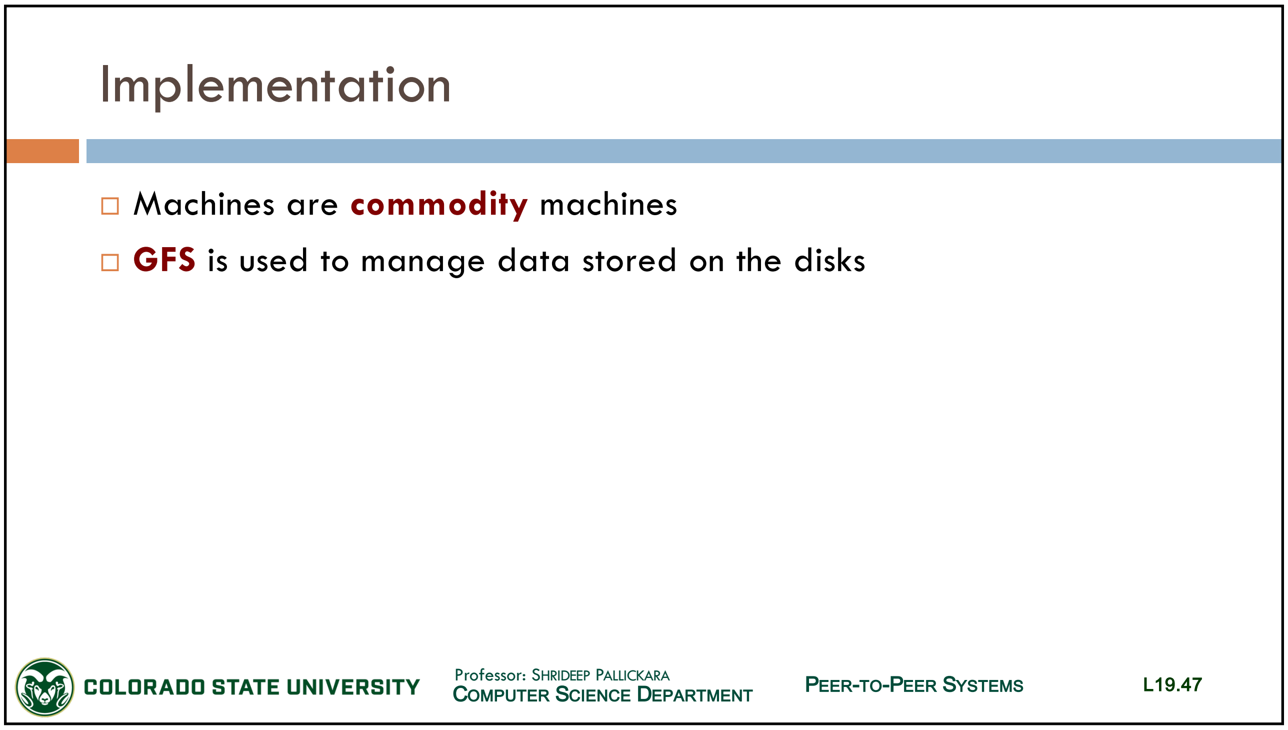
Inverted Index

- Map
  - Parse each document
  - Emit <`word, document ID`>
- Reduce
  - Accept all pairs for a given `word`
  - Sort `document IDs`
  - Emit <`word, list(document ID)`> pair

45

IMPLEMENTATION

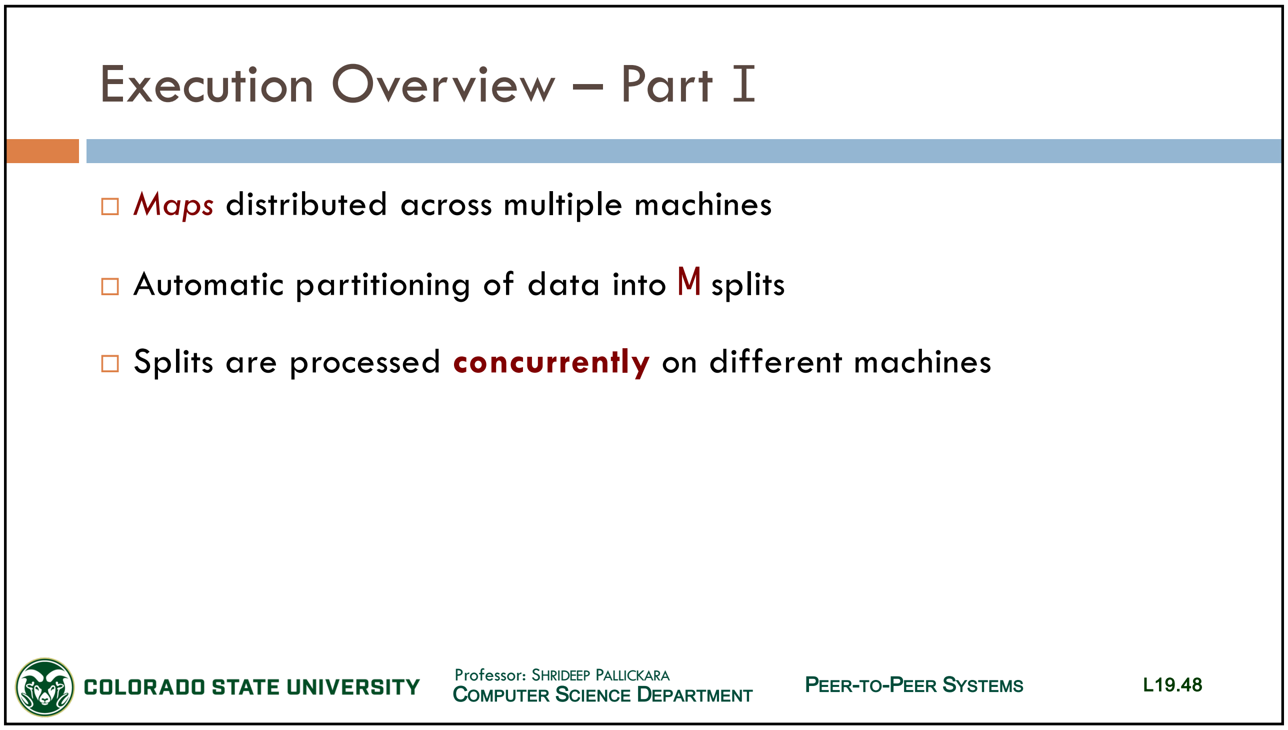
46



Implementation

- Machines are **commodity** machines
- **GFS** is used to manage data stored on the disks

47



Execution Overview – Part I

- *Maps* distributed across multiple machines
- Automatic partitioning of data into M splits
- Splits are processed **concurrently** on different machines

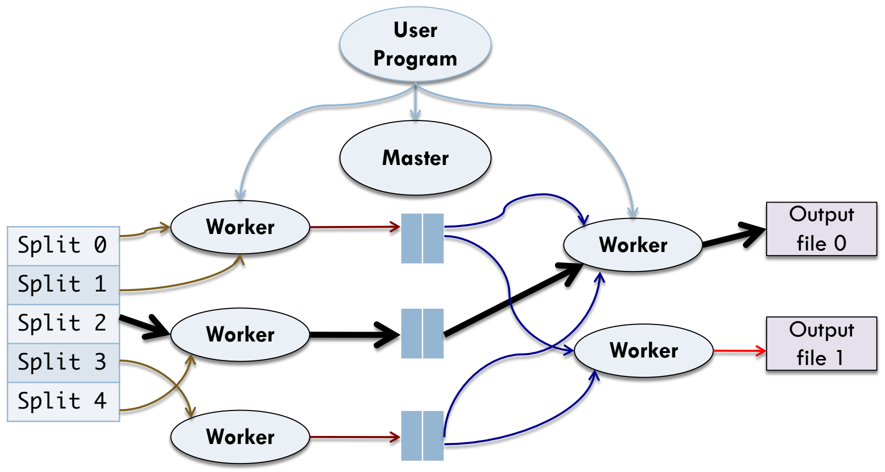
48

Execution Overview – Part II

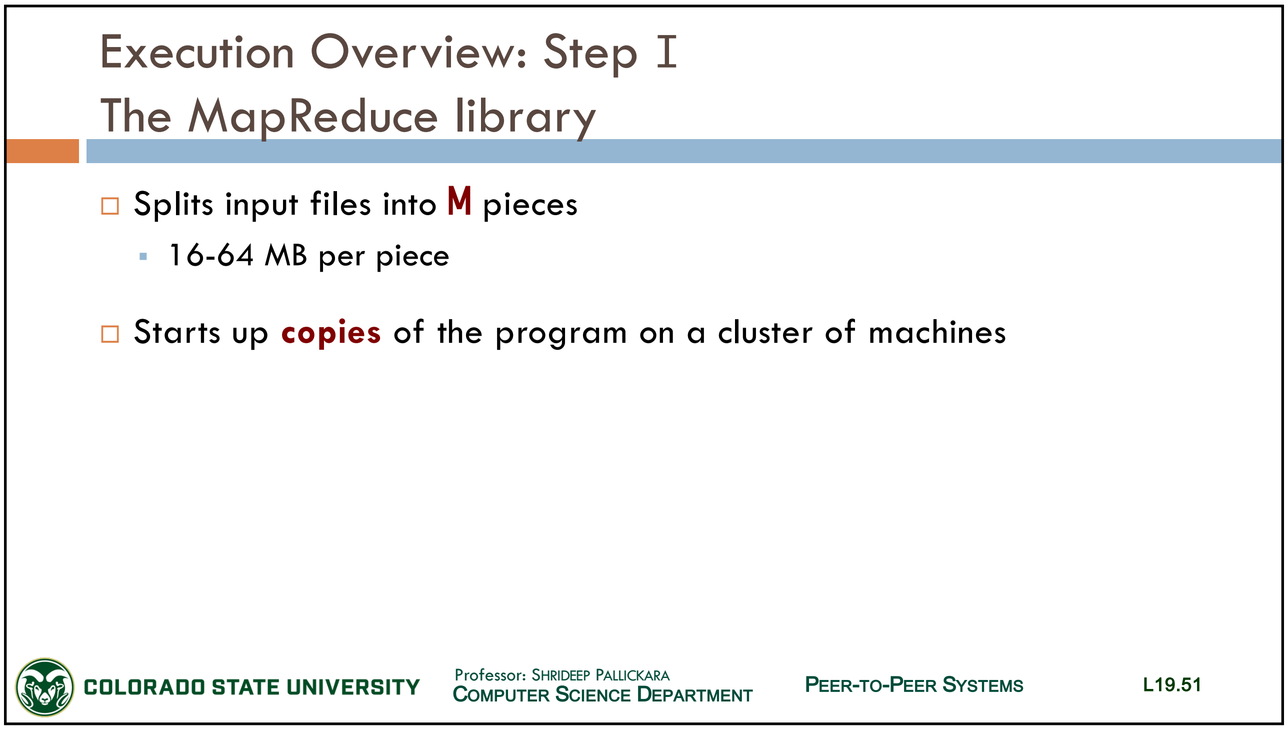
- Partition *intermediate* key space into R pieces
- E.g. hash(key) **mod** R
- User specified parameters
  - **Partitioning** function
  - **Number** of partitions (R)

49

Execution Overview



50

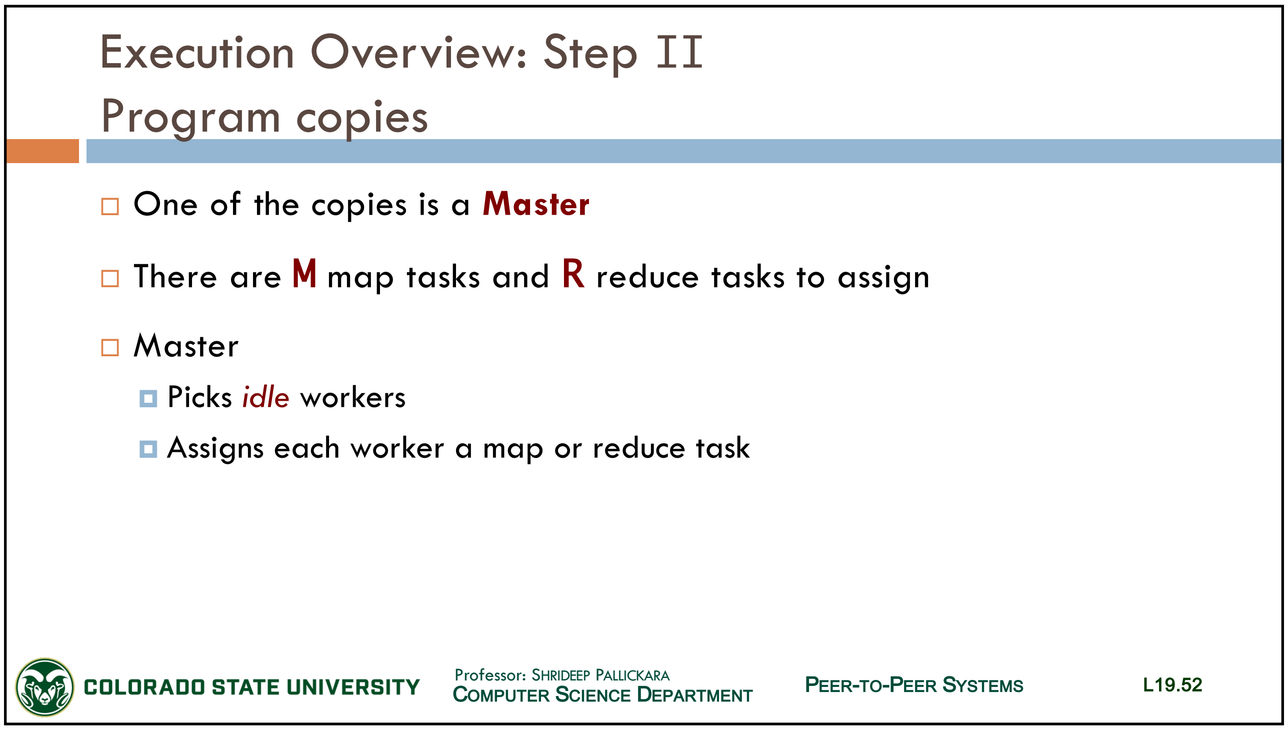


Execution Overview: Step I
The MapReduce library

- Splits input files into **M** pieces
  - 16-64 MB per piece
- Starts up **copies** of the program on a cluster of machines

COLORADO STATE UNIVERSITY — Professor: SHRIDEEP PALLICKARA, COMPUTER SCIENCE DEPARTMENT — PEER-TO-PEER SYSTEMS — L19.51

51



Execution Overview: Step II
Program copies

- One of the copies is a **Master**
- There are **M** map tasks and **R** reduce tasks to assign
- Master
  - Picks *idle* workers
  - Assigns each worker a map or reduce task

COLORADO STATE UNIVERSITY — Professor: SHRIDEEP PALLICKARA, COMPUTER SCIENCE DEPARTMENT — PEER-TO-PEER SYSTEMS — L19.52

52

Execution Overview: Step III
Workers that are assigned a map task

- Read contents of their input split
- Parses <*key, value*> pairs out of the input data
- Pass each pair to user-defined *Map* function
- Intermediate <*key, value*> pairs from *Maps*
  - Buffered in Memory

53

Execution Overview: Step IV
Writing to disk

- Periodically, **buffered pairs** are written to disk
- These writes are partitioned
  - By the partitioning function
- **Locations** of buffered pairs on local disk
  - *Reported* back to Master
  - Master *forwards* these locations to reduce workers

54

Execution Overview: Step V
Reading Intermediate data

- Master notifies *Reduce* worker about locations
- Reduce worker reads buffered data from the **local disks** of *Maps*
- Read *all* intermediate data; sort by intermediate key
  - All occurrences of the same key are grouped together
  - Many different keys map to the same *Reduce* task

Execution Overview: Step VI
Processing data at the Reduce worker

- Iterate over sorted intermediate data
- For each unique key pass
  - *Key* **+** set of *intermediate values* to Reduce function
- Output of the Reduce function is appended
  - To output file of the reduce partition

Execution Overview: Step VII
Waking up the user

- After all Map & Reduce tasks have been completed
- Control returns to the user code

Master Data Structures

- For each Map and Reduce task
  - **State**: {*idle*, *in-progress*, *completed*}
  - Worker **machine** identity
- For each completed Map task store
  - **Location** and **sizes** of **R** intermediate file regions
- Information pushed incrementally to *in-progress* Reduce tasks

The contents of this slide-set are based on the following references

- *Distributed Systems: Concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. 5th Edition. Addison Wesley. ISBN: 978-0132143011.* [Chapter 10]
- Jeffrey Dean, Sanjay Ghemawat: *MapReduce: Simplified Data Processing on Large Clusters.* OSDI 2004: 137-150
- Jeffrey Dean, Sanjay Ghemawat: MapReduce: simplified data processing on large clusters. Commun. ACM 51(1): 107-113 (2008)

59

FAULT TOLERANCE

60

Worker failures

- Master **pings** worker periodically
- After a certain number of failed pings
  - Master marks worker as having failed
- Any Map task completed by failed worker?
  - **Reset** to initial *idle* state
  - Eligible for **rescheduling**

Why completed Map tasks are reexecuted

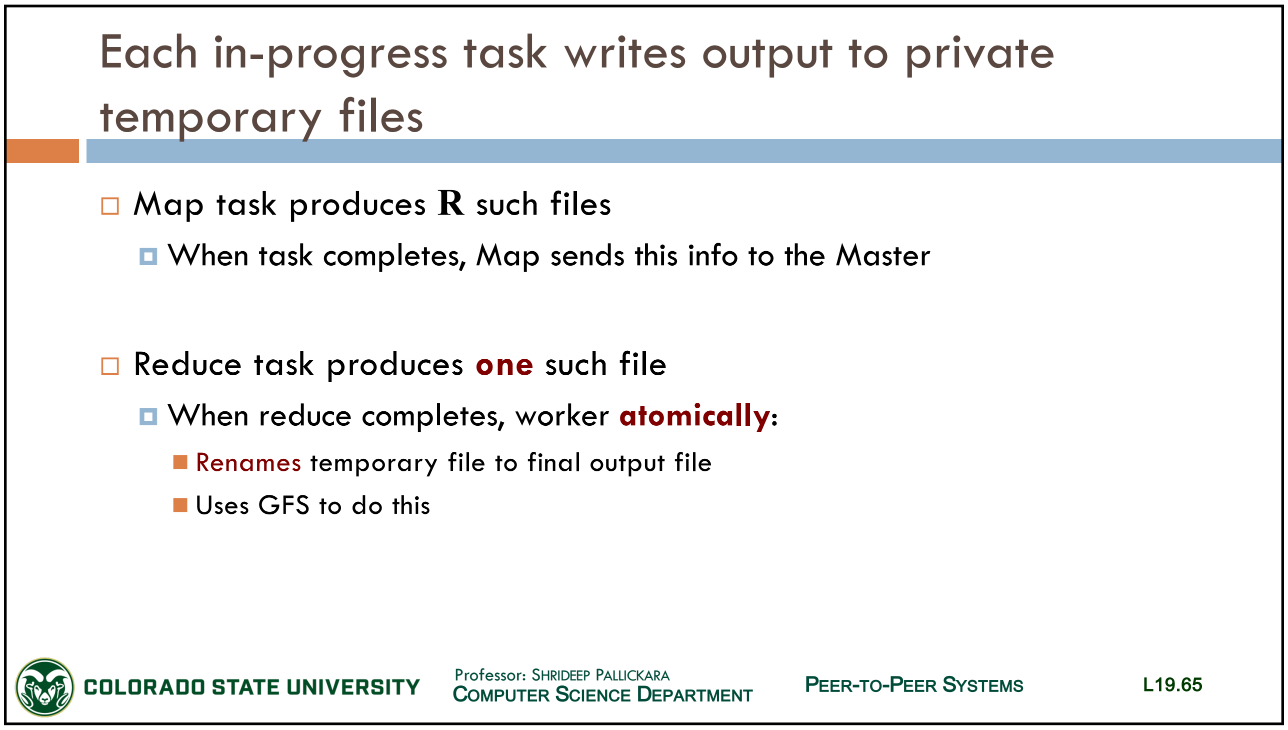
- Output is stored on **local disk** of failed machine
  - Inaccessible
- All reduce workers are notified about reexecution
- Reduce tasks *do not* need to be reexecuted
  - Output stored in GFS

Master Failures

- Could **checkpoint** at the Master
  - Data structures are well-defined
- However, since there is only one Master
  - Assumption is that failure is unlikely
- If there is a Master failure?
  - MapReduce computation is **aborted**!
  - Client must *check and retry* MapReduce operation

Semantics in the presence of failures:
If *map* and reduce operators are deterministic

- Distributed execution output is identical to
  - Non-faulting, sequential execution
- Atomic commits of map and reduce task outputs help achieve this

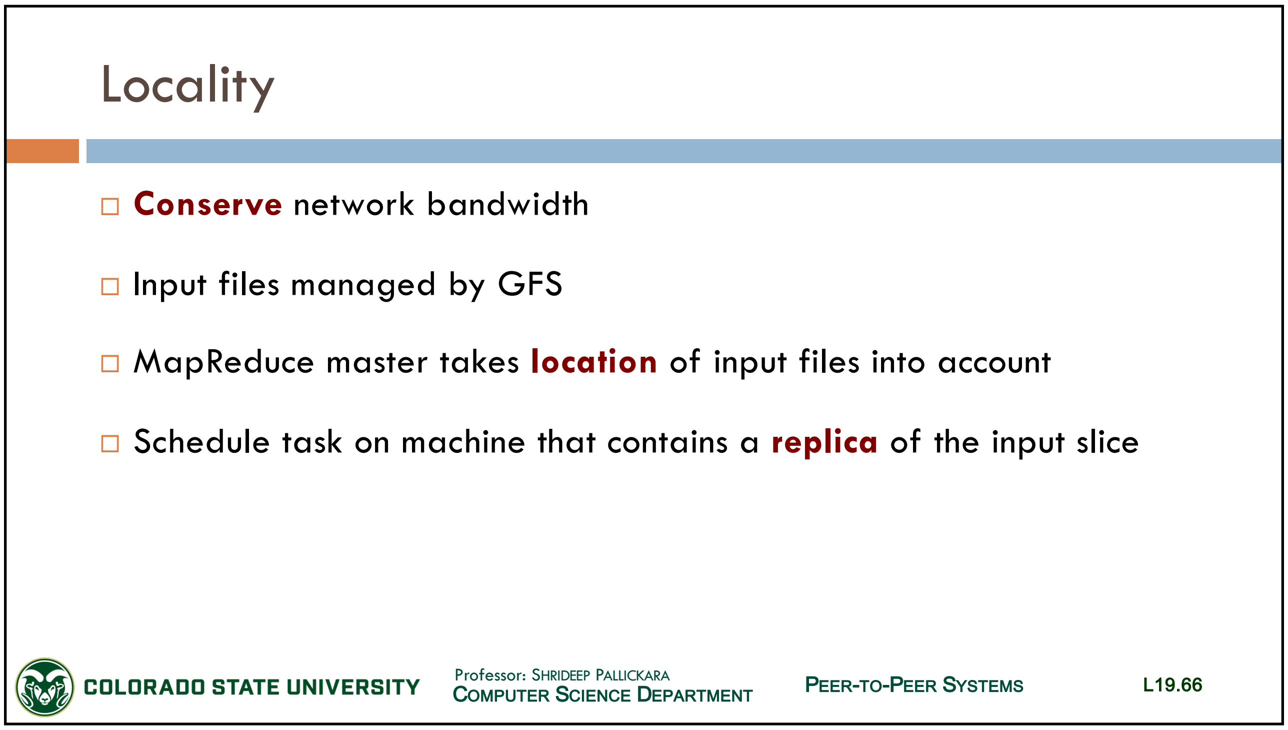


Each in-progress task writes output to private temporary files

- Map task produces **R** such files
  - When task completes, Map sends this info to the Master
- Reduce task produces **one** such file
  - When reduce completes, worker **atomically**:
    - Renames temporary file to final output file
    - Uses GFS to do this

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L19.65

65



Locality

- **Conserve** network bandwidth
- Input files managed by GFS
- MapReduce master takes **location** of input files into account
- Schedule task on machine that contains a **replica** of the input slice

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT PEER-TO-PEER SYSTEMS L19.66

66

Locality and its impact when running large MapReduce tasks

- Most input data is read **locally**
- Consumes no network bandwidth

67

TASK GRANULARITY

68

Task Granularity

- Subdivide map phase into **M** pieces
- Subdivide reduce phase into **R** pieces
- **M**, **R** >> number of worker machines
- Each worker performing many different tasks:
  - Improves **dynamic load balancing**
  - Speeds up **recovery** during failures

Practical bounds on how large M and R can be

- Master must make $O(\boldsymbol{M} + \boldsymbol{R})$ scheduling decisions
- Keep $O(\boldsymbol{MR})$ state in memory

Practical bounds on how large M and R can be

- **M** is chosen such that
  - Input data is roughly 16 MB to 64 MB
- **R** constrained by users
  - Output of each reduce is in a separate file
- **R** is a *small multiple* of the number of machines that will be used

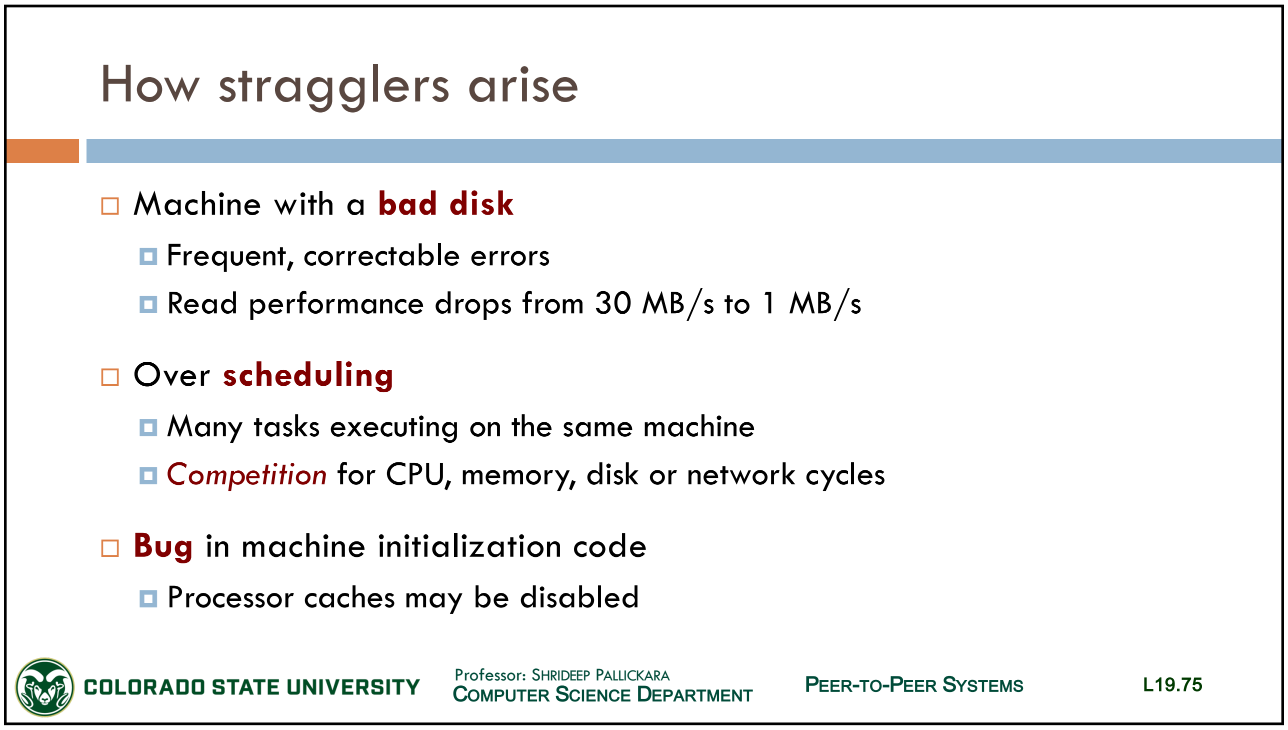
Typical values used at Google

- **M** = 200,000
- **R** = 5,000
- **W** = 2,000 worker machines

BACKUP TASKS

Stragglers

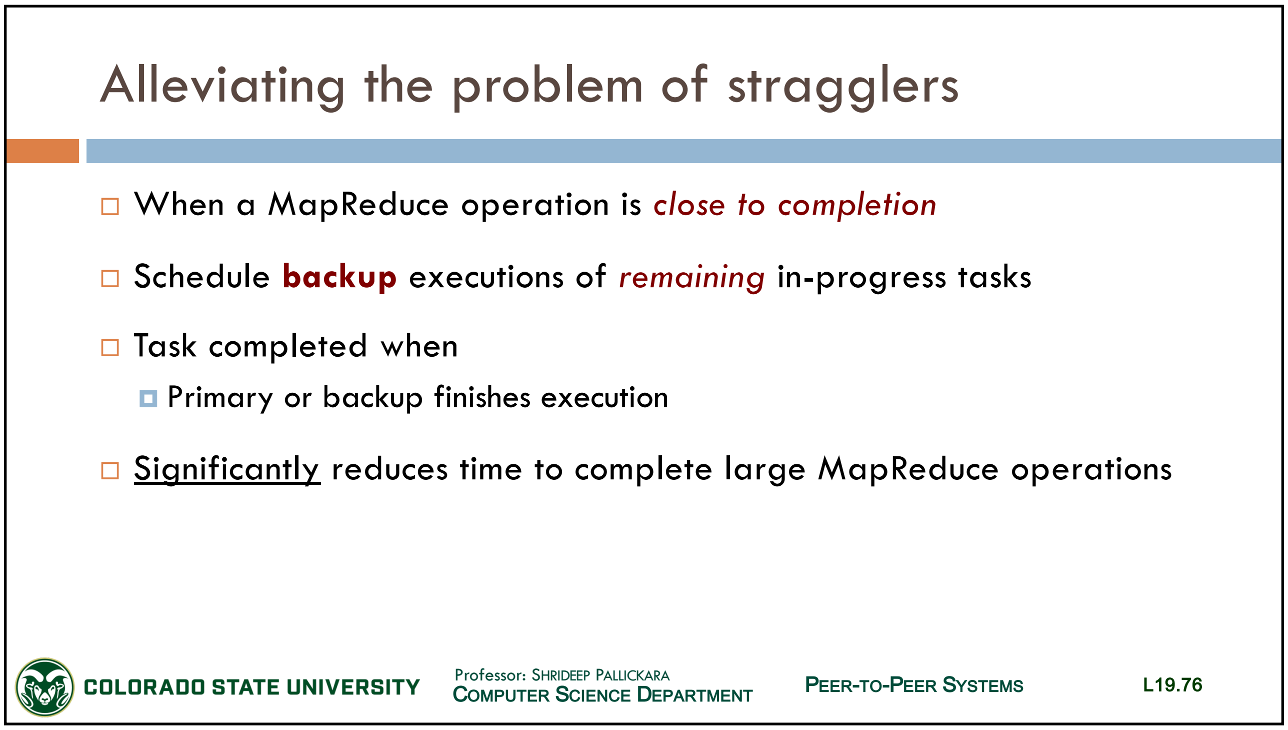
- Machine that takes an **unusually long time** to complete a map or reduce operation
- Can slow down entire computation



How stragglers arise

- Machine with a **bad disk**
  - Frequent, correctable errors
  - Read performance drops from 30 MB/s to 1 MB/s
- Over **scheduling**
  - Many tasks executing on the same machine
  - *Competition* for CPU, memory, disk or network cycles
- **Bug** in machine initialization code
  - Processor caches may be disabled

75



Alleviating the problem of stragglers

- When a MapReduce operation is *close to completion*
- Schedule **backup** executions of *remaining* in-progress tasks
- Task completed when
  - Primary or backup finishes execution
- <u>Significantly</u> reduces time to complete large MapReduce operations

76

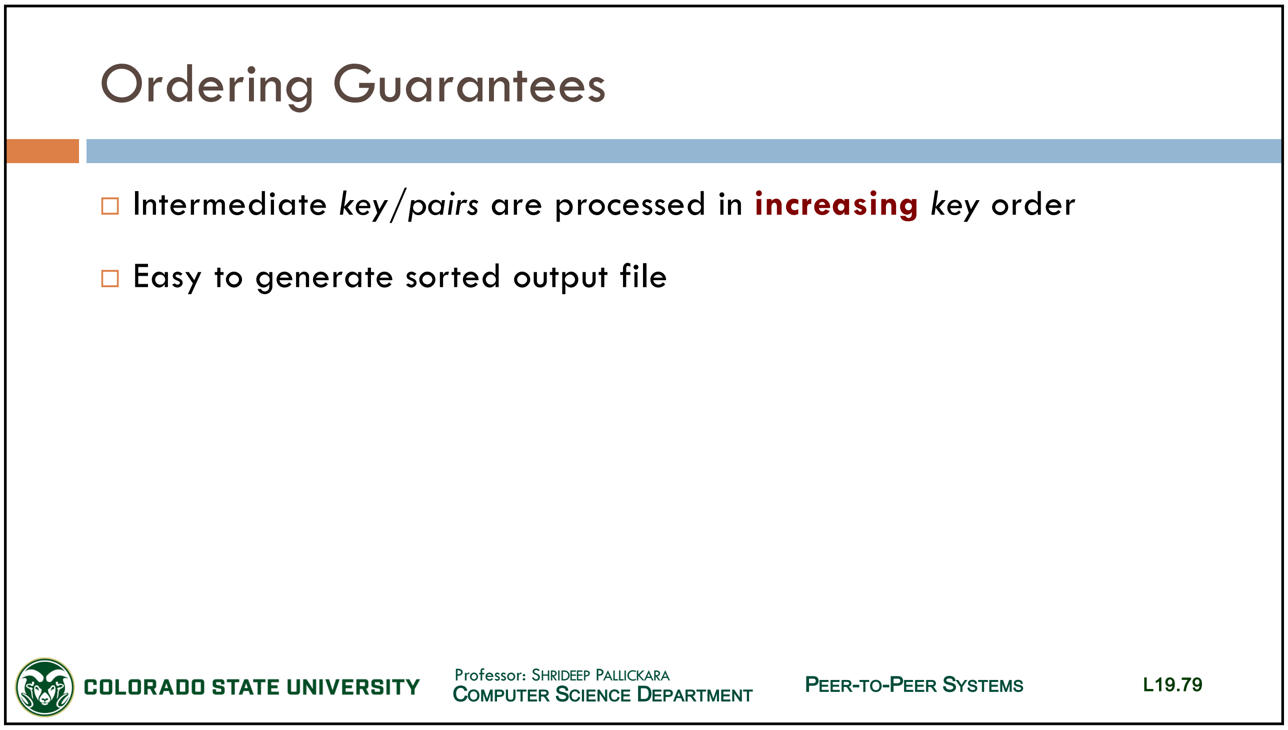
REFINEMENTS

77

Partitioning Function

- Users simply specify R
  - The number of output files
- Default partitioning
  - `hash(key) mod R`
- Sometimes output keys are URLs
  - Entries from a host must go to same output file
  - `hash(Hostname(urlkey)) mod R`

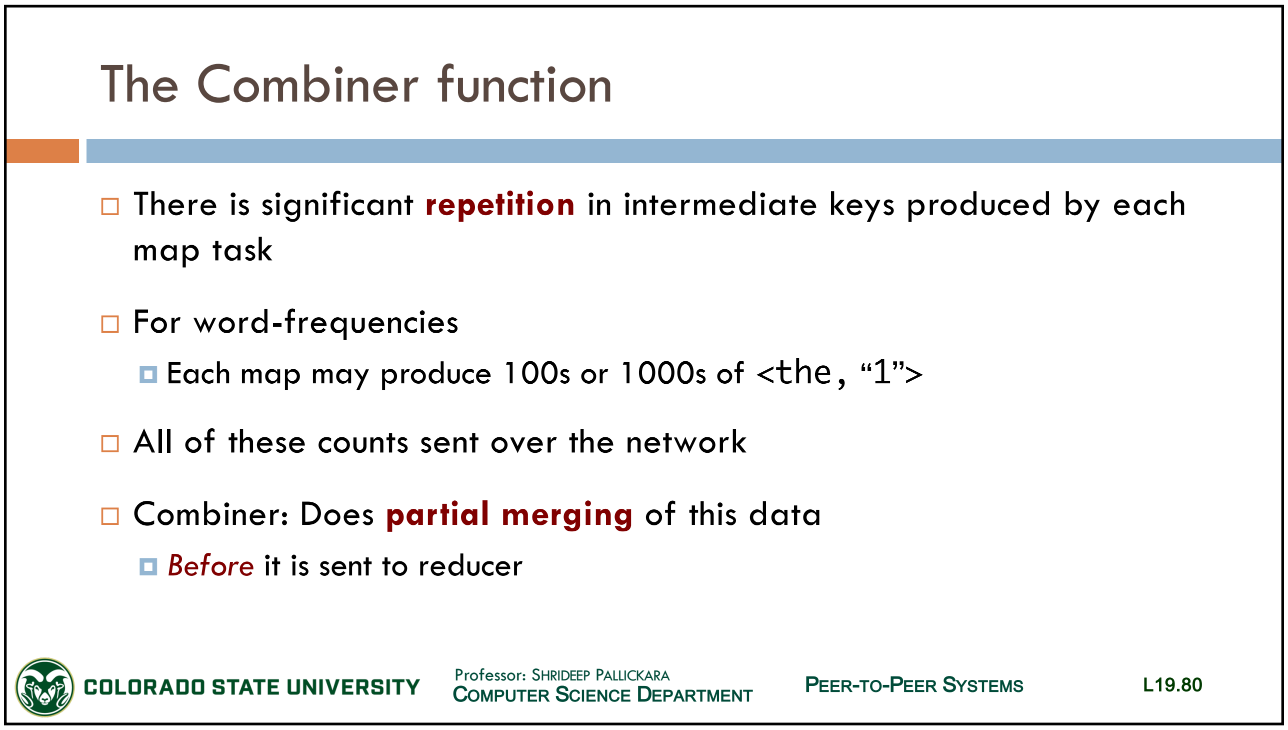
78



Ordering Guarantees

- Intermediate *key/pairs* are processed in **increasing** *key* order
- Easy to generate sorted output file

79



The Combiner function

- There is significant **repetition** in intermediate keys produced by each map task
- For word-frequencies
  - Each map may produce 100s or 1000s of <the, “1”>
- All of these counts sent over the network
- Combiner: Does **partial merging** of this data
  - *Before* it is sent to reducer

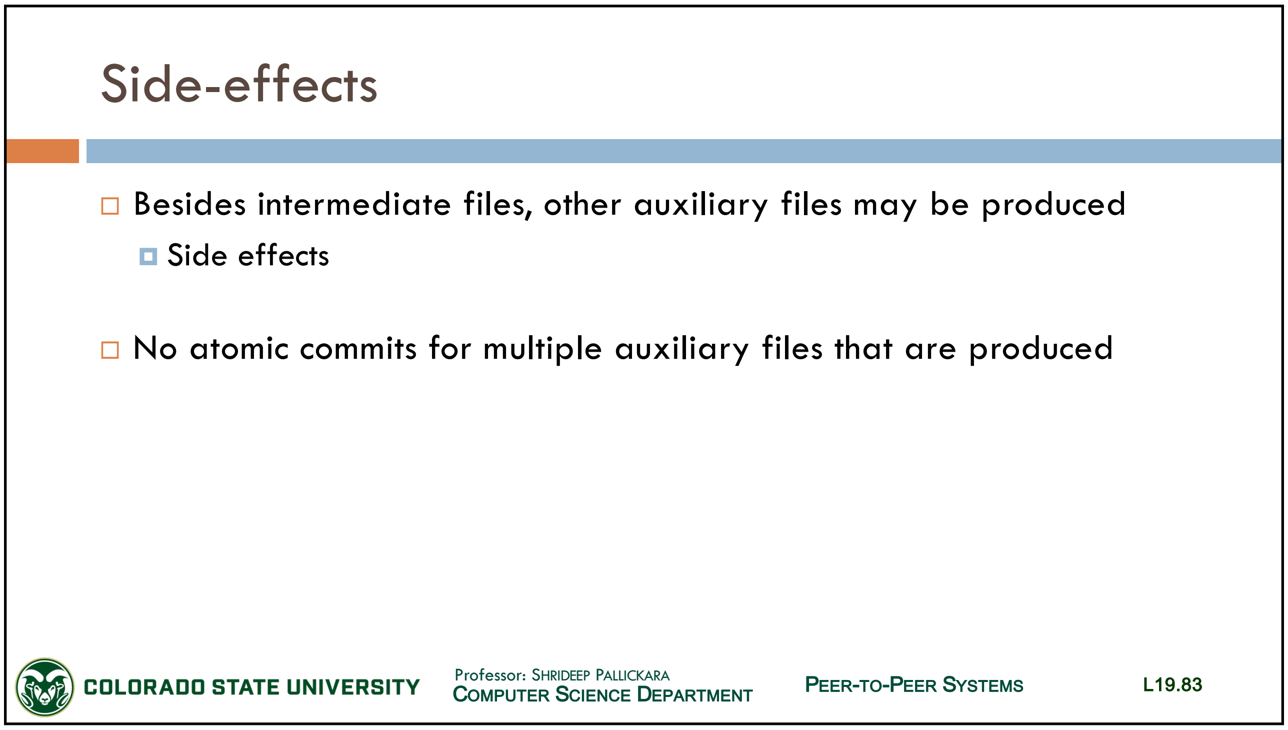
80

Combiner function

- Executed on each machine that performs map task
- Code implementing combiner & reduce function
  - *Usually* the same … [We will see an example where this is not true.]
- Difference?
  - COMBINE: Output written to *intermediate* file
  - REDUCE: Output written to *final output* file

Input/Output Types: Support for reading input data in different formats

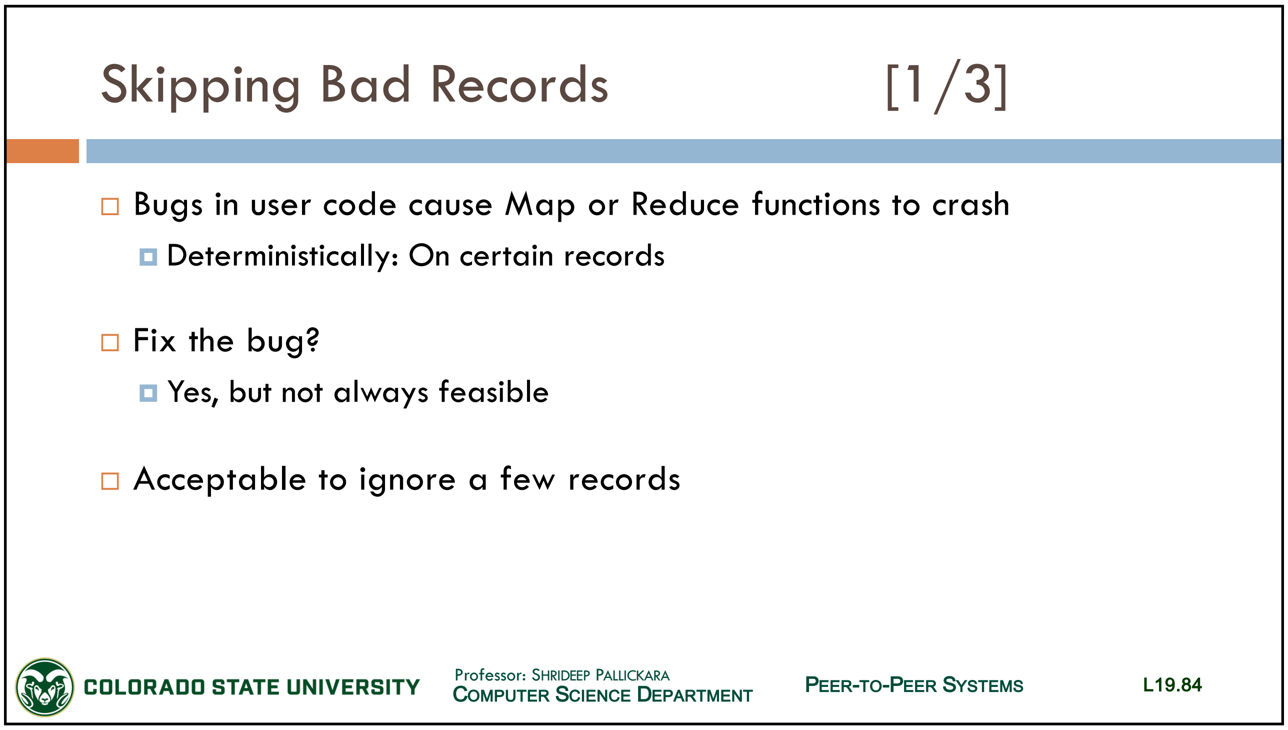
- Text mode treats every line as a <*key, value*> pair
  - Key: Offset in the file
  - Value: Contents of the line
- <*key, value*> pairs are sorted by *key*
- Each input type *knows how to split itself* for
  - Processing as separate map tasks
  - Text mode splitting occurs only at line boundaries



Side-effects

- Besides intermediate files, other auxiliary files may be produced
  - Side effects
- No atomic commits for multiple auxiliary files that are produced

83



Skipping Bad Records [1/3]

- Bugs in user code cause Map or Reduce functions to crash
  - Deterministically: On certain records
- Fix the bug?
  - Yes, but not always feasible
- Acceptable to ignore a few records

84

Skipping Bad Records [2/3]

- Optional mode of operation
  1. Detect records that cause *deterministic crashes*
  2. Skip them
- Each worker installs a **signal handler** to catch segmentation violations and bus errors

Skipping Bad Records [3/3]

- Signal handler sends *last gasp* UDP packet to the Master
  - Contains sequence number
- When Master sees more than 1 failure at that record
  - Indicates record should be skipped during next execution

Local Execution

- Support for **sequential execution** of MapReduce operation on a single machine
  - Helps with debugging, profiling, and testing
- Controls to *limit* computation to a particular map
- Invoke programs with a special flag
  - Use debugging and testing tools

Status Information

- Master runs internal HTTP Server
- Exports pages for viewing
- Show the progress of a computation
  - Number of tasks in progress
  - Number of tasks that completed
  - Bytes of input
  - Bytes of intermediate data
  - Processing rate

The contents of this slide-set are based on the following references

- *Distributed Systems: Concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. 5th Edition. Addison Wesley. ISBN: 978-0132143011.* [Chapter 10]
- Jeffrey Dean, Sanjay Ghemawat: *MapReduce: Simplified Data Processing on Large Clusters.* OSDI 2004: 137-150
- Jeffrey Dean, Sanjay Ghemawat: MapReduce: simplified data processing on large clusters. Commun. ACM 51(1): 107-113 (2008)