ISTM 6212 - Week 10
Map / Reduce, Trifecta Wrangler

Daniel Chudnov, dchud@gwu.edu

2016-11-15

Agenda

- Schedule check
- Map / Reduce
- Project 03
- Wrangling with Trifecta
- Exercise 05 work session

Schedule check

Map / Reduce

How did we get here?

- In the beginning: the command line, pipes and filters
- Relational databases, normalization, transactions
- Dimensional models, denormalization, conformance
- What does each bring us?

Command line - benefits

- Fast, free, ubiquitous, flexible
- Small, focused tools do one thing well
- Text/line orientation keeps things simple
- Pipeline model allows sophisticated workflows
- Filters can be in any language
- Scales up pretty well, to an extent

Command line - drawbacks

- Feels a bit arcane
- Hundreds of little things to learn
- Easy to screw up
- Different environments vary (e.g. Ubuntu vs. MacOS)
- Takes a lot of practice, helps to know a wizard

Relational databases - benefits

- Rock-solid basis in relational algebra
- Wealth of commercial and free/open source products
- Long-developed SQL specs for DDL, DML
- Sophisticated tools for access control, scaling up, data consistency, application development
- SQL mostly the same across systems

Relational databases - drawbacks

- SQL mostly the same across systems
- Heavy-duty products require dedicated staff
- Overhead of schema development, normalization
- Schema requirements limit flexibility, slow development
- Transactional designs not ideal for analysis

Dimensional models - benefits

- Time-tested concepts: dimension, fact, grain, conformance
- Optimal strategies for analytical needs
- Allow analysts to move quickly in extracting data
- Denormalization keeps schema and queries relatively simple
- BI tool integration simplifies focus on analysis, reporting

Dimensional models - drawbacks

- Still requires dedicated staff
- Still requires schema overhead
- ETL process must be carefully managed
- Ad hoc designs don't grow and scale well
- Centrally managed designs may move too slowly

Map / Reduce

- Dean and Ghemawat, 2004
- Simplified programming model
- Flexible data model
- Optimized computing model

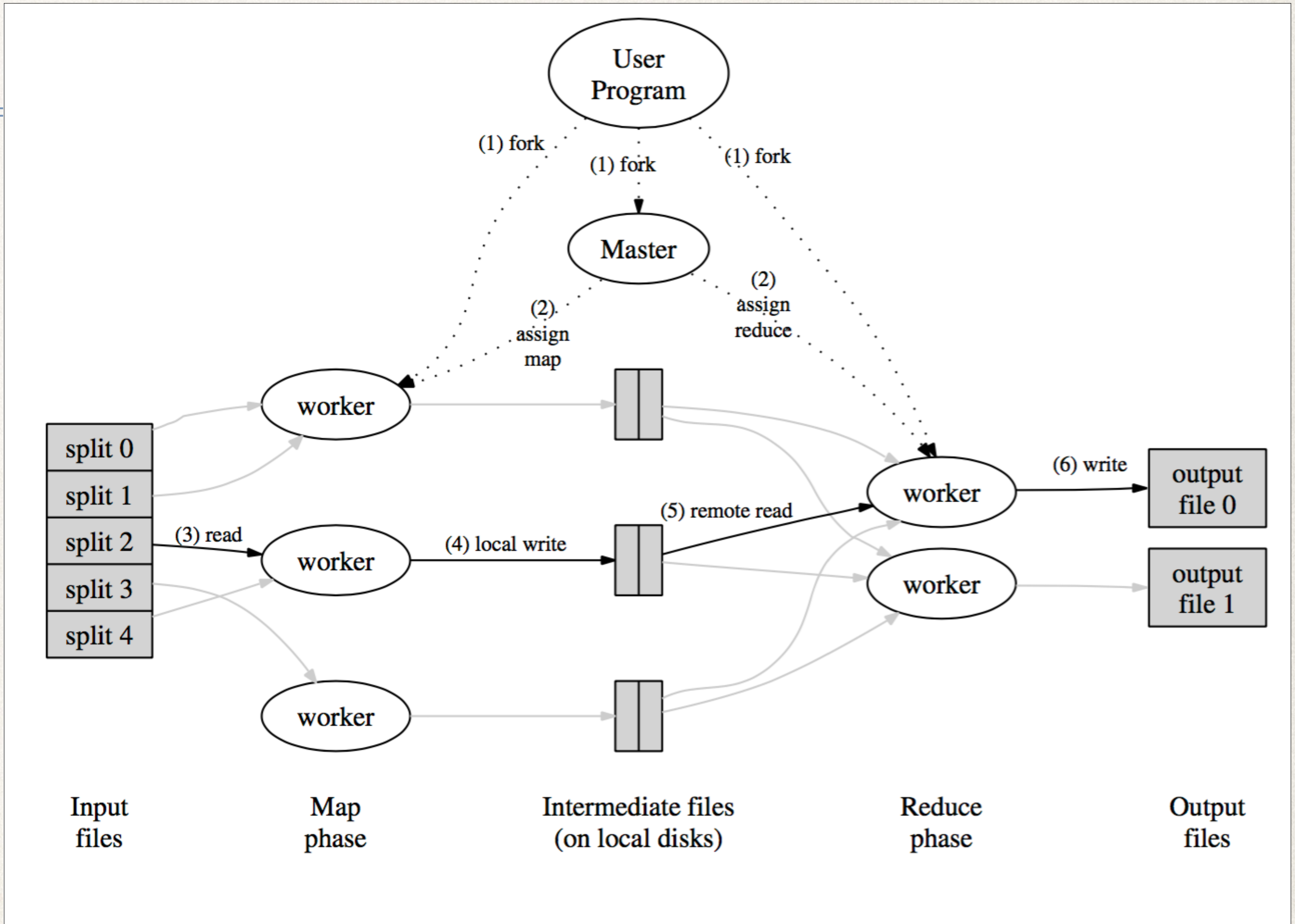


Figure 1: Execution overview

Map / Reduce

- Original paradigm 50+ years old (functional programming in Lisp, etc.)
- Developed at Google in the early 2000s
- Concept spread widely, Apache Hadoop project 10+ years old (see en.wikipedia.org/wiki/Apache_Hadoop)
- Mature platform ~5 years, rapidly growing ecosystem

How does it work?

- Start with any source data
- **Map** function extracts keys and values
- **Reduce** function summarizes values by key

- That's it!

Map function example

- Source data:

ALPS AND SANCTUARIES OF PIEDMONT AND THE CANTON TICINO
by Samuel Butler

Author's Preface to First Edition

I should perhaps apologise for publishing a work which professes to
deal with the sanctuaries of Piedmont, and saying so little about
the most important of them all--the Sacro Monte of Varallo. My
excuse must be, that I found it impossible to deal with Varallo
without making my book too long. Varallo requires a work to
itself; I must, therefore, hope to return to it on another
occasion.

- Mapped data:

1 alps
1 and
1 sanctuaries
1 of
1 piedmont
1 and
1 the
1 canton
1 ticino

Map function

- grep -oE '\w{2,}' example.txt | tr '[:upper:]' '[:lower:]' | uniq -c

Reduce function example

- Mapped data:

1 alps
1 and
1 sanctuaries
1 of
1 piedmont
1 and
1 the
1 canton
1 ticino

- Reduced data:

1 about
1 all
1 alps
3 and
1 another
1 apologise
1 author
1 be
1 book

Reduce function

- sort | uniq -c

that's Map / Reduce*

*or most of it, anyway

Unique value from each

- Command line - free, flexible, & powerful assembly of small pieces
- RDBMS - consistent data management, standardized declarative query, statistics-driven optimization
- Dimensional - optimized query experience for analysts
- Map/Reduce - flexible data, simplified code model, optimized computing strategy scales linearly

How map/reduce works

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

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));
```

- Simplified programming model
- Commodity hardware
- Scheduler manages jobs, storage
- Network file system helps jobs run near data

How map/reduce works (2)

- Input split into standard-sized chunks
- Chunked data stored redundantly on networked file system
- Scheduler assigned compute nodes to map data they have stored locally ("locality")
- Mapped data written back to file system

How map/reduce works (3)

- Reduce jobs scheduled to work on mapped data
- Reducers sort intermediate keys to group/combine results
- Reducers write results back to file system
- Scheduler tracks storage, job status, re-assigns around failures, duplicates workload near the end

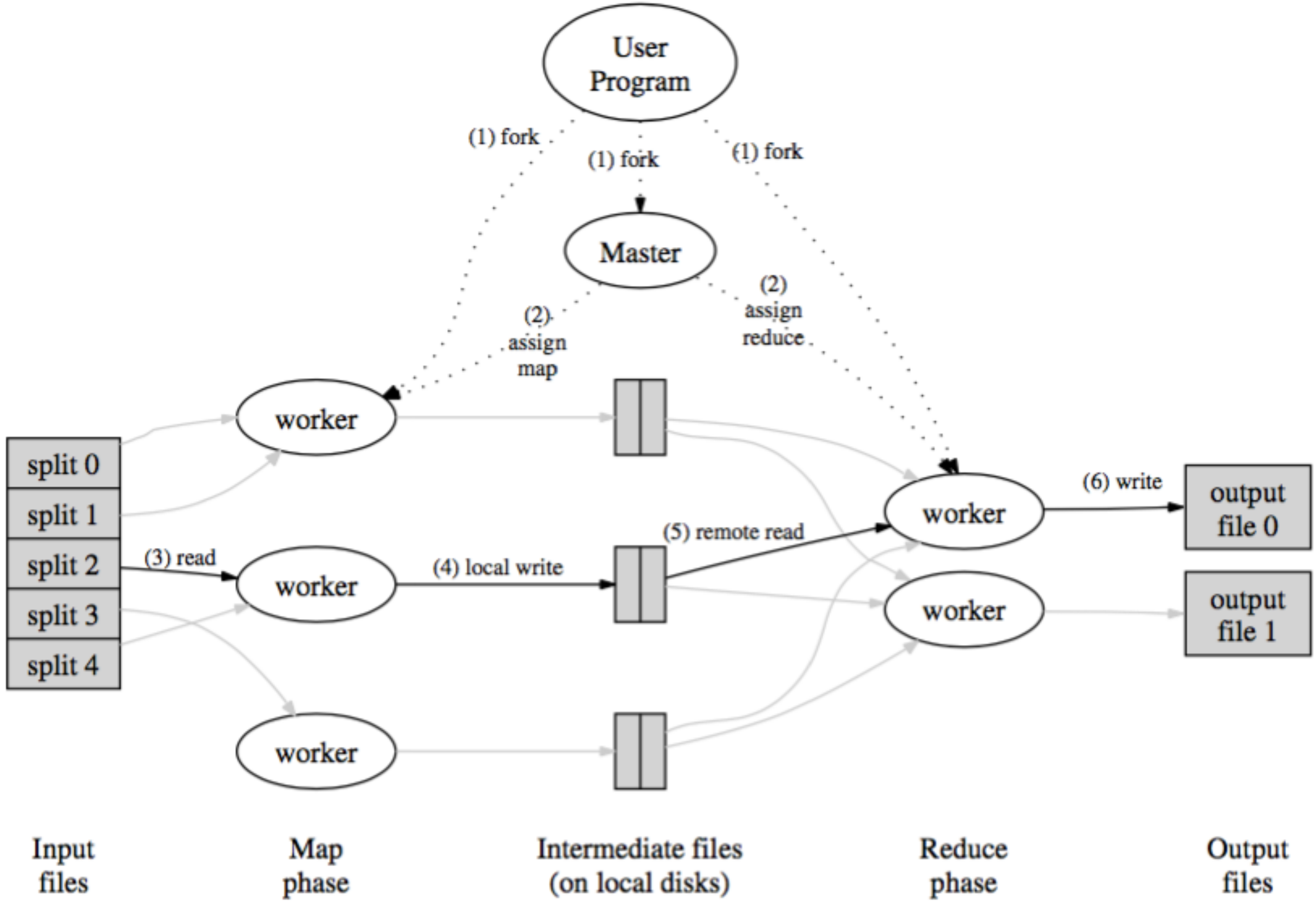
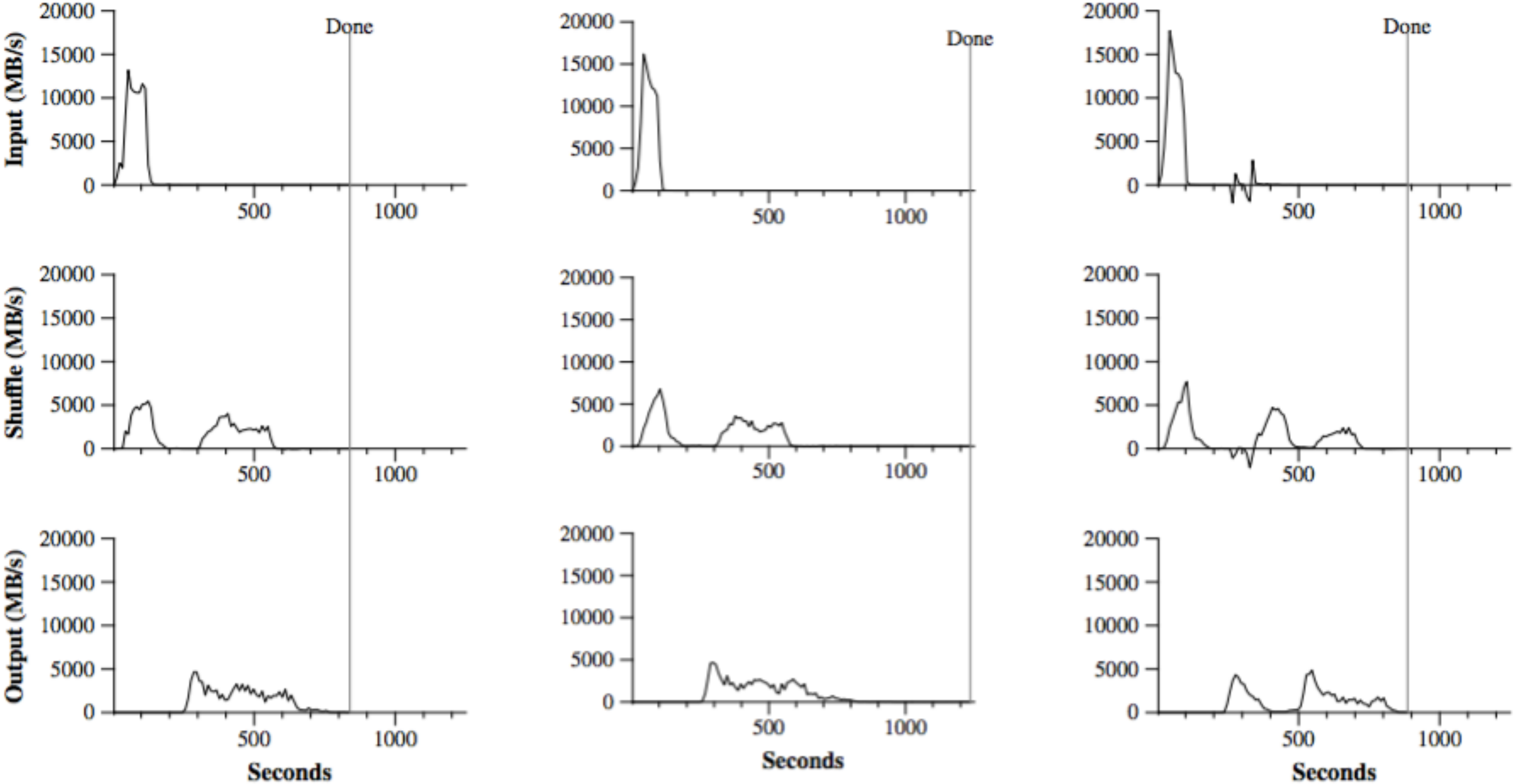


Figure 1: Execution overview

Compute optimizations

- **Locality** of redundant data ensures jobs occur on nodes with copies of data - no extra network traffic
- **Fault tolerance** allows jobs to finish even if whole blocks of machines become unavailable
- **Backup tasks** route around "stragglers"
- Hash optimization, bad records, counters, etc.



(a) Normal execution

(b) No backup tasks

(c) 200 tasks killed

Figure 3: Data transfer rates over time for different executions of the sort program

What does this mean for you?

- Map/reduce is widely used; **Hadoop** is a mature free/open source implementation
- Most organizations using data at scale have clusters onsite, via AWS or BigTable+Google, or plan to
- Schema-free flexibility, simple programming model, great performance means we can just “pile up our data” and transform/conform it when we need it

How is this good for you?

- **Data lakes*** still require conforming dimensions for merging/drilling across sources
- **Spark** and dozens of other tools build on and improve map/reduce model and Hadoop infrastructure layer
- You can run it on your laptop or via AWS/Google
- You can get started with it easily (Exercise 06 next week!)

*search “Tamara Dull data lake” for more on this concept

Is all this stuff mature?

- Yes, or at least mature enough
- You can download Spark and use it locally with minimal configuration
- Cloud-hosted services already stable and inexpensive
- SQL interfaces (Spark SQL, Apache Drill, etc.) are coming quickly and improving quickly
- SQL on top of map/reduce - SQL knowledge counts

More on overhead

- How big is your data chunk size?
- Which hashing algorithm fits best?
- Which compression should you use?
- How many nodes are needed per job?
- What level of storage redundancy makes sense?
- Where do dimension conformance specifications live?

Keep it in context

- Schema vs schemaless
- Consistency vs flexibility
- Speed vs scale
- Scaling requires staffing
- Scaling requires scheduling overhead

Next week: Spark

- skim the Apache Spark site, docs
- get a Databricks Community Edition account
- Exercise 06 - Spark on Databricks (probably!)

Project 03 (Final project)

Final project

- Groups of four people (should be 12!)
- Task like Project 02, but open to tools: commandline, db, spark, trifacta, etc.
- Dataset >= 250,000 records
- Presentation Tuesday, 6 December
- Writeup Friday, 9 December
- Reviews Tuesday, 12 December

Trifacta Wrangler

Trifacta Wrangler

- Grew out of university research
- Well-funded, quickly improving
- Aims to simplify data wrangling, push ETL out to analysts
- Example workflow: Trifacta for cleaning/wrangling, Tableau for visualization

Trifacta Wrangler - concepts

- Projects and datasets
- Transform scripts
  - Suggested transform patterns
  - Required transform language
- Develop scripts of data samples
- Generate results - output to CSV, Excel, TDE

Exercise 05 work session