MIE1512 Data Analytics

MapReduce II

UofT

How many Map and Reduce jobs?

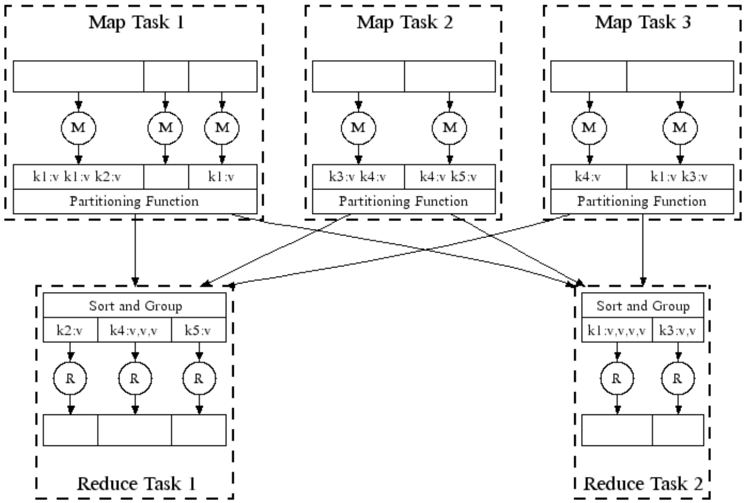
- M map tasks, R reduce tasks
- Rule of thumb:
  - Make M and R much larger than the number of nodes in cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds recovery from worker failure
- Usually R is smaller than M, because output is spread across R files

Combiners

- Often a map task will produce many pairs of the form (k,v1), (k,v2), … for the same key k
  - E.g., popular words in Word Count
- Can save network time by pre-aggregating at mapper
  - combine(k1, list(v1)) → v2
  - Usually same as reduce function
- Works only if reduce function is commutative and associative

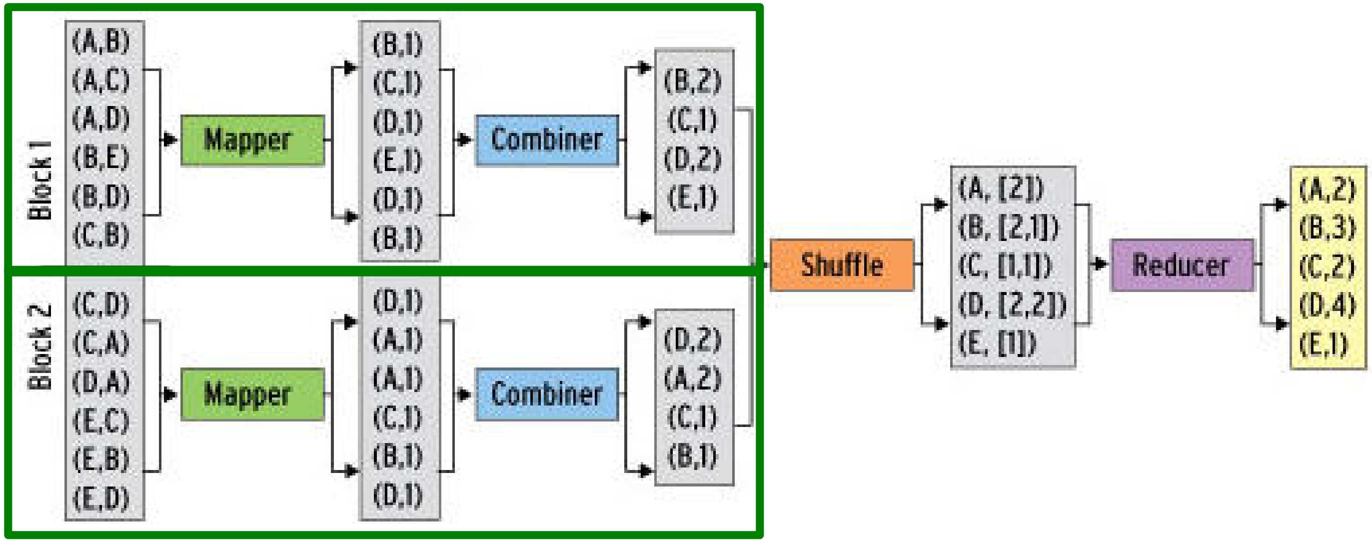
Refinement: Combiners

- Often a Map task will produce many pairs of the form *(k,v₁), (k,v₂), …* for the same key *k*
  - E.g., popular words in the word count example
- **Can save network time by pre-aggregating values in t mapper:**
  - combine(k, list(v_1)) → v_2
  - Combiner is usually same as the reduce function
- Works only if reduce function is commutative and associative



Refinement: Combiners

- **Back to our word counting example:**
  – Combiner combines the values of all keys of a single mapper (single machine):



  – Much less data needs to be copied and shuffled!

UofT

Refinement: Partition Function

- **Want to control how keys get partitioned**
  - Inputs to map tasks are created by contiguous splits of input file
  - Reduce needs to ensure that records with the same intermediate key end up at the same worker
- **System uses a default partition function:**
  - **hash(key) mod *R***
- **Sometimes useful to override the hash function:**
  - E.g., **hash(hostname(URL)) mod *R*** ensures URLs from a host end up in the same output file

UofT

Example: Join By Map-Reduce

- **Compute the natural join *R(A,B)* ⋈ *S(B,C)***
- *R* and *S* are each stored in files
- Tuples are pairs *(a,b)* or *(b,c)*

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

⋈

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

=

A	C
a_3	c_1
a_3	c_2
a_4	c_3

Map-Reduce Join

- **Use a hash function *h* from B-values to *1...k***
- **A Map process turns:**
  - Each input tuple *R(a,b)* into key-value pair *(b,(a,R))*
  - Each input tuple *S(b,c)* into *(b,(c,S))*

- **Map processes** send each key-value pair with key *b* to Reduce process *h(b)*
  - Hadoop does this automatically; just tell it what *k* is.
- Each **Reduce process** matches all the pairs *(b,(a,R))* with all *(b,(c,S))* and outputs *(a,b,c)*.

Big Data Support on the Cloud

How? Use a Cloud processing framework that processes **parallel tasks** across commodity machines while taking care of the underlying **communication, scheduling and monitoring of tasks**

- **Low-level:** e.g. processing log files using Hadoop
- **Records:** e.g. relations (HadoopDB, Hive)
- **Nested Records:** e.g. JSON (Pig, Jaql), XML (ChuQL)
- **Graphs:** e.g. SPARQL (PigSPARQL), RDF Analytics (RAPID, ExpLOD), message-passing algorithms (Pregel)

Understanding How Semi-Structured Datasets are Processed on the Cloud

- **Describe and compare low-level operations**, such as **joins** and **iterations**, and how they leverage the **data**, **processing**, and **storage** models of Hadoop
  - Processing model includes the task scheduler
  - Low-level operations: hash join (default), repartition, theta-, broadcast, semi-, set-similarity; iterations
- Integration of low-level operations within **high-level record, nested-record, and graph** data models, and consider **analytical, query, and algorithmic** approaches

UofT

Example: Summing Ad Click Values

- **adinfo** (adID, cost, year), each record has a unique advertisement identifier, with its creation cost and year
- **clicklog** (vID, adclickID, adID, value, ..), each vendor provides its vendor identifier, a unique click identifier, the identifier of the advertisement that was clicked, and the click value. **Semi-Structured.**

```
Vendor1 log:
vID, adclickID, adID, value, browser
1, 3000, 5, 0.03, Chrome
1, 3001, 6, 0.05, IExplore
```

```
Vendor2 log:
vID, adclickID, adID, value, networkid, useridhash
2, 2500, 5, 0.04, sports, C560363
2, 1500, 5, 0.06, life, C560363
```

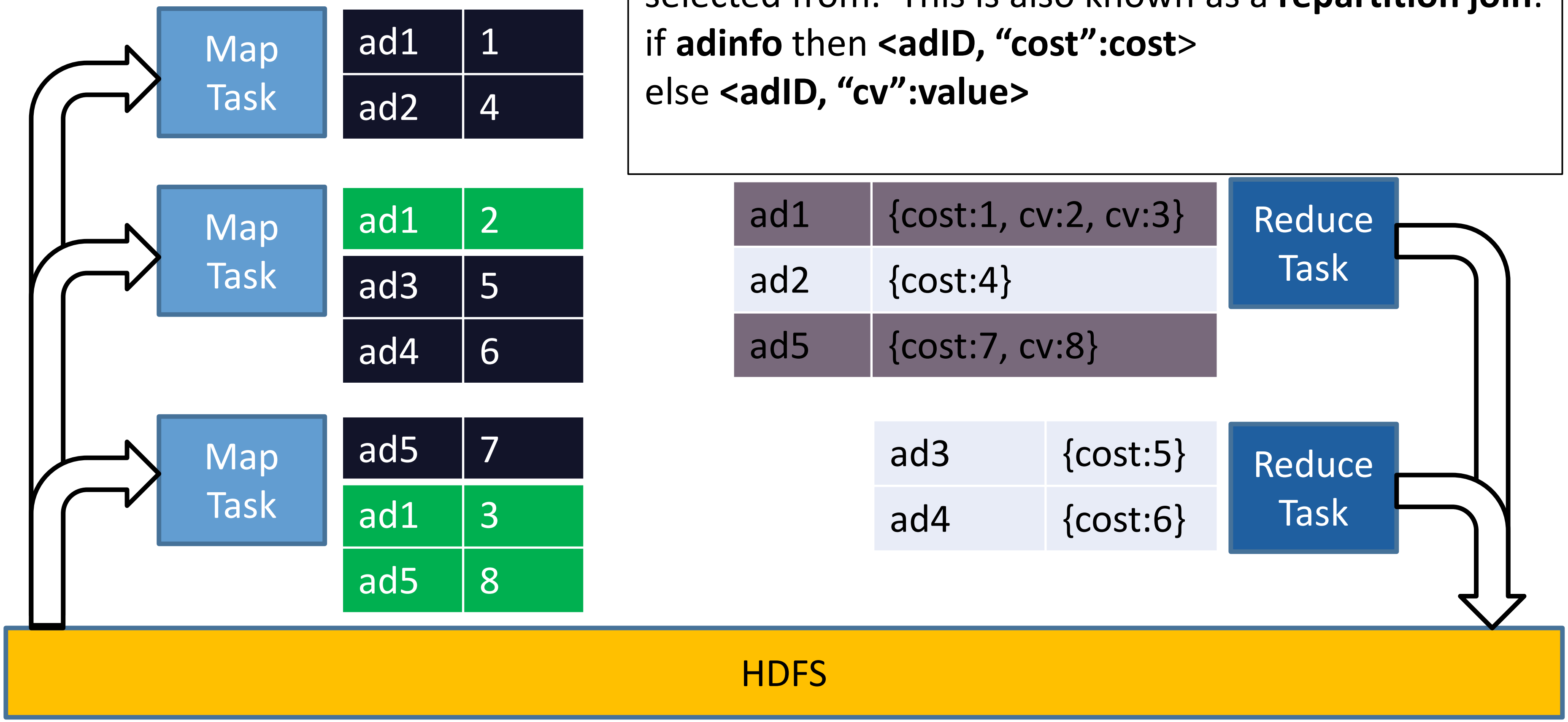
Hive [TSJ10], record-based join expression:

```sql
INSERT OVERWRITE TABLE alladcosts SELECT ai.adID, ai.cost, av.vsum
  FROM (SELECT cl.adID, SUM(cl.value) AS vsum
  FROM clicklog cl GROUP BY cl.adID) av
JOIN adinfo ai WHERE (ai.adID = av.adID); ;
```

[TSJ10] Thusoo, A.; Sarma, J. S.; Jain, N.; Shao, Z.; Chakka, P.; Zhang, N.; Antony, S.; Liu, H. & Murthy, R. Hive -- A Petabyte Scale Data Warehouse Using Hadoop. *ICDE,* 2010, 996-1005

Summing Click Values

Records are "tagged" with the relation they are selected from. This is also known as a **repartition join**:
if **adinfo** then **<adID, "cost":cost>**
else **<adID, "cv":value>**



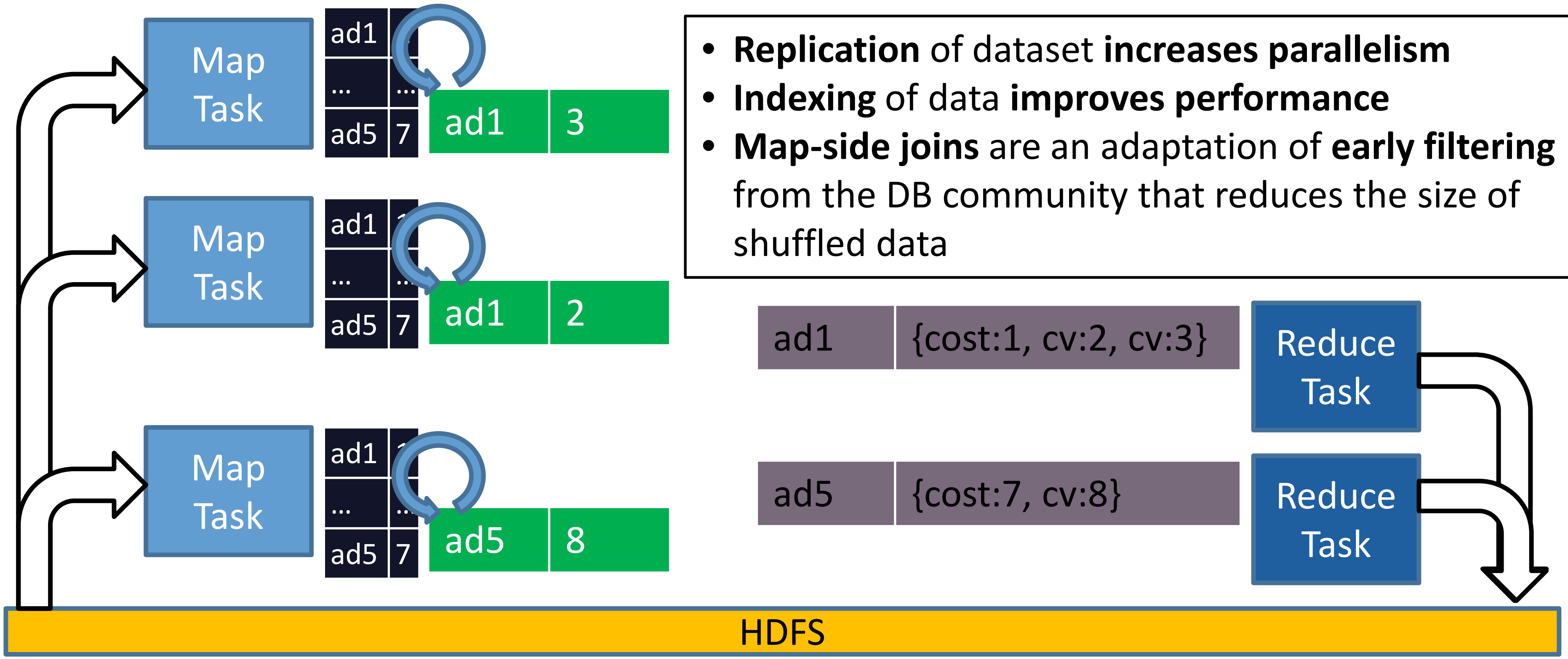
<ad1, {cv:5, cost:1}>
<ad5, {cv:8, cost:7}>

adinfo(adID, cost, year),
clicklogven1(vID, adclickID, adID, value, browser),
clicklogven2(vID, adclickID, adID, value, networkid, uidhash)

Sum Click Values Using Broadcast join

- adinfo relation replicated to all machines
- vendor logs streamed and joined with main-memory hash map of adinfo relation

- **Replication** of dataset **increases parallelism**
- **Indexing** of data **improves performance**
- **Map-side joins** are an adaptation of **early filtering** from the DB community that reduces the size of shuffled data



adinfo(adID, cost, year),
clicklogven1(vID, adclickID, adID, value, sessionid),
clicklogven2(vID, adclickID, adID, value, browser)

<ad1, {cv:5, cost:1}>
<ad5, {cv:8, cost:7}>

High-Level Approaches: Records, Nested-Records, Graphs

- Low-level approaches require coding using a programming language like Java, making it hard to maintain and reuse code amongst jobs
- Data-driven approaches to processing semi-structured data on the Cloud abstract away low-level implementation details behind high-level languages
- Support **inter-job optimizations** while **integrating intra-job** features/optimizations
- Support integration of data-driven and programmatic (imperative) approaches for different degrees of control

Summary of High-Level Approaches

HadoopDB	Hive	Pig Latin	Jaql	ChuQL
Relational	Nested-Record	Nested-Record	JSON	XML extended with records
Declarative query (HiveQL)	Declarative query (HiveQL)	Declarative script	Declarative script	Mixed functional expressions and imperative scripting
Equi-joins (partial results from RDBMS joins are hash-joined)	Equi-joins (hash or broadcast join)	Equi-joins (hash or broadcast join)	Equi-joins (hash or broadcast join)	Equi-joins based on XML's *fn:deep-equals* semantics (hash join)
Low-level	Low-level	Low-level	Low-level and functional	Low-level and functional

- HadoopDB's use of RDBMSes is a distinct approach of caching and indexing for improving performance
- Jaql [BEG11] and Pig Latin [ORS08] are declarative scripting languages that do not support loops, while some loops are supported by ChuQL's[KCS11, KCS11C] imperative language features
- Few integrated low-level operations
- Jaql and ChuQL give users the option to route data through the underlying MapReduce processing model

Our Own Work on MR

XML Related

- [KCS11] Khatchadourian, S.; Consens, M. P. & Siméon, J. Having a ChuQL at XML on the Cloud. *AMW,* 2011
- [KCS11C] Khatchadourian, S.; Consens, M. P. & Siméon, J. ChuQL: Processing XML with XQuery using Hadoop. *CASCON,* 2011

RDF Related

- [KC10] Khatchadourian, S. & Consens, M. P. Understanding Billions of Triples with Usage Summaries, Semantic Web Challenge ISWC, 2011
  - [KC10] Khatchadourian, S. & Consens, M. P. ExpLOD: Summary-Based Exploration of Interlinking and RDF Usage in the Linked Open Data Cloud. *ESWC,* 2010
  - [KC10D] Khatchadourian, S. & Consens, M. P. Exploring RDF Usage and Interlinking in the Linked Open Data Cloud using ExpLOD. *LDOW,* 2010

MIE1512 Data Analytics

Spark

Spark Bibliography

- https://spark.apache.org/research.html

- Required reading

Spark SQL: Relational Data Processing in Spark. Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, Matei Zaharia. *SIGMOD 2015*. June 2015.

- Introductory slides from KDD2015 Tutorial by J. Shanahan, L. Dai

  http://kdd2015-sparktutorial.droppages.com/

UofT