Distributed Systems I: MapReduce, Google File System CS 571: Operating Systems (Spring 2022)

Lecture 11 Yue Cheng

Some material taken/derived from:

• Princeton COS-418 materials created by Michael Freedman and Wyatt Lloyd.

MIT 6.824 by Robert Morris, Frans Kaashoek, and Nickolai Zeldovich

Licensed for use under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

Announcement

- Grade of mini exam 2 released on BB
- Project presentation video due in two weeks
 - Make sure your video is ready by Monday, May 2
- If you prefer to do an online demo (Friday, May 6), let me know
 - We can only schedule 4-5 teams in the online session, so FCFS; rest of 9-10 teams will do the inclassroom demo on Wednesday

What is a distributed system?



- Multiple computers
- Connected by a network
- Doing something together
- A *distributed system* is many cooperating computers that appear to users as a single service

Today's outline

How can large computing jobs be parallelized?

- 1. MapReduce
- 2. Google File System

Today's outline

How can large computing jobs be parallelized?

1. MapReduce

2. Google File System

Applications			
Web	Data	Data	Emerging
apps	processing	storage	apps?
Resource management			
Compute resources	Memory resources	Storage resources	Network resources



Datacenter H/W infrastructure





Review: Shared memory



- Shared memory: multiple processes to share data via memory
- Applications must locate and and map shared memory regions to exchange data

Review: Shared memory vs. Message passing



- Shared memory: multiple processes to share data via memory
- Applications must locate and and map shared memory regions to exchange data



- Message passing: exchange data explicitly via IPC
- Application developers define protocol and exchanging format, number of participants, and each exchange

Review: Shared memory vs. Message passing

- Easy to program; just like a single multithreaded machines
- Hard to write high perf. apps:
 - Cannot control which data is local or remote (remote mem. access much slower)
- Hard to mask failures

- Message passing: can write very high perf. apps
- Hard to write apps:
 - Need to manually decompose the app, and move data
- Need to manually handle failures

Shared memory: Pthread

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX (e.g., Linux) OSes

Shared memory: Pthread

```
void *myThreadFun(void *vargp) {
    sleep(1);
    printf("Hello world!\n");
    return NULL;
}
int main() {
    pthread_t thread_id_1, thread_id_2;
    pthread_create(&thread_id_1, NULL, myThreadFun, NULL);
    pthread_create(&thread_id_2, NULL, myThreadFun, NULL);
    pthread_join(thread_id_1, NULL);
    pthread_join(thread_id_2, NULL);
    exit(0);
}
```

Message passing: MPI

- MPI Message Passing Interface
 - Library standard defined by a committee of vendors, implementers, and parallel programmers
 - Used to create parallel programs based on message passing
- Portable: one standard, many implementations
 - Available on almost all parallel machines in C and Fortran
 - De facto standard for the HPC & parallel computing community

Message passing: MPI

```
int main(int argc, char **argv) {
      MPI Init(NULL, NULL);
      // Get the number of processes
      int world_size;
      MPI_Comm_size(MPI_COMM_WORLD, &world_size);
      // Get the rank of the process
      int world rank;
      MPI Comm rank(MPI COMM WORLD, *world rank);
      // Print off a hello world message
      printf("Hello world from rank %d out of %d processors\n",
            world rank, workld size);
      // Finalize the MPI environment
      MPI Finalize();
}
```

Message passing: MPI

mpirun -n 4 -f host_file ./mpi_hello_world

```
int main(int argc, char **argv) {
    MPI_Init(NULL, NULL);
```

```
// Get the number of processes
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```

```
// Get the rank of the process
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, *world_rank);
```

```
// Finalize the MPI environment
MPI_Finalize();
```

}

MapReduce

The big picture (motivation)

Datasets are too big to process using a single computer

The big picture (motivation)

- Datasets are too big to process using a single computer
- Good parallel processing engines are rare (back then in the late 90s)

The big picture (motivation)

- Datasets are too big to process using a single computer
- Good parallel processing engines are rare (back then in the late 90s)
- Want a parallel processing framework that:
 - is general (works for many problems)
 - is **easy to use** (no locks, no need to explicitly handle communication, no race conditions)
 - can automatically parallelize tasks
 - can automatically handle machine failures

Context (Google circa 2000)

- Starting to deal with massive datasets
- But also addicted to cheap, unreliable hardware
 - Young company, expensive hardware not practical
- Only a few expert programmers can write distributed programs to process them
 - Scale so large jobs can complete before failures



Context (Google circa 2000)

- Starting to deal with massive datasets
- But also addicted to cheap, unreliable hardware
 - Young company, expensive hardware not practical
- Only a few expert programmers can write distributed programs to process them
 - Scale so large jobs can complete before failures
- Key question: how can every Google engineer be imbued with the ability to write parallel, scalable, distributed, fault-tolerant code?
- Solution: abstract out the redundant parts
- Restriction: relies on job semantics, so restricts which problems it works for

What MapReduce is good at?



What MapReduce is not good at?



Application: Word Count

```
cat data.txt
    | tr -s `[[:punct:][:space:]]' `\n'
    | sort | uniq -c
```

SELECT count(word), word FROM data GROUP BY word

Deal with multiple files?

1. Compute word counts from individual files

Deal with multiple files?

- 1. Compute word counts from individual files
- 2. Then merge intermediate output

Deal with multiple files?

- 1. Compute word counts from individual files
- 2. Then merge intermediate output
- 3. Compute word count on merged outputs

What if the data is too big to fit in one computer?

- 1. In parallel, send to worker:
 - Compute word counts from individual files
 - Collect results, wait until all finished

What if the data is too big to fit in one computer?

- 1. In parallel, send to worker:
 - Compute word counts from individual files
 - Collect results, wait until all finished
- 2. Then merge intermediate output

What if the data is too big to fit in one computer?

- 1. In parallel, send to worker:
 - Compute word counts from individual files
 - Collect results, wait until all finished
- 2. Then merge intermediate output
- 3. Compute word count on merged intermediates

MapReduce: Programming interface

- map(k1, v1) \rightarrow list(k2, v2)
 - Apply function to (k1, v1) pair and produce set of intermediate pairs (k2, v2)

- reduce(k2, list(v2)) \rightarrow list(k3, v3)
 - Apply aggregation (reduce) function to values
 - Output results

MapReduce: Word Count

```
map(key, value):
   for each word w in value:
       EmitIntermediate(w, "1");
reduce(key, values):
   int result = 0;
   for each v in values:
       results += ParseInt(v);
   Emit(AsString(result));
```

Word Count execution



Word Count execution



Y. Cheng

Word Count execution



MapReduce data flows



GMU CS571 Spring 2022
MapReduce processes

Map Shuffle Reduce

Reduce

Reduce

- Map workers write intermediate output to local disk, separated by partitioning. Once completed, tell master node
- Reduce worker told of location of map task outputs, pulls their partition's data from each mapper, execute function across data
- Note:
 - "All-to-all" shuffle b/w mappers and reducers
 - Written to disk ("materialized") b/w each state

GMU CS571 Spring 2022

Map

Map

Map

Apache Hadoop



- An open-source implementation of Google's MapReduce framework
 - Hadoop MapReduce atop Hadoop Distributed File System (HDFS)

A Brief History of Hadoop



DATA & AI LANDSCAPE 2019

INFRASTRUCTURE		ANALYTICS & MACHINE INTELLIGENCE		APPLICATIONS – ENTERPRISE				
HADOOP ON-PREMISE HADOOP IN THE CLOUD	STREAMING / IN-MEMORY	DATA ANALYST PLATFORMS	DATA SCIENCE PLATFORMS	SALES	MARKETING - B2B	MARKETING - B2C	CUSTOMER EXPERIENCE / S	
cloudera Hortonworks	Amazon Kinesis 🧔 engradu batabricks	Microsoft Opentation alteryx	IEM 🔹 databricks 🕗 iku	BUCIDE CALES COM	RADIUS App Annie	braze ACTIONIC PRUECOPE	Qualtrics" 🔷 MEDALLIA 🧥 Sur @CLARABRIDGE zendesk 🤅	Kustomer @freshdesk
MAPR. Pivotal.			SDOMINO DOMINO TIBCO				Drift OLIVEPERSON	Gainsight Apendo ORACLE
IBM InfoSphere IBM InfoSphere Orm	Sstriim hazelcast	ATTIV/O Datameer incorto.	O Ssas 🖧	Clari A aviso tact.ai (TROOPS	-+ tubular 🛞 📋	Amplero. 🤱 amperity 📓 QUANTIFIND	CHEAP CAmplitude Of Watson A	ssistant he platogrow GURU lumiata
jethro	GIGASPACES & Wattaroouxes VP FASTDATA" KX	inter ana. MODE ENDOR		fuse machines 🔽 Clearbit 🧐	KNOTCH MIDS	eremesh	CallDesk metorni	talla® Kasisto
NoSOL DATABASES Society Country Streams Managebra SCYLLA. New SOL DATABASES SAME Classific Screen Society Streams Same Screen Same Screen Sa	MPP DBs VERTICA Without stress Gooden Could Coogle Could Marcel Without Stress Coogle Could Marcel Without Stress Marcel Ma	BIPLATFORMS USUALIZATI USUA	ON MACHINE LEARNING MACHINE LEARNING MACHINE LEARNING MARK New Mark New	HUMAN CAPITAL //www.exponetris hill @ constrained @ Autyoo Creation @ Woodostwindy Stella @ www.exponetris @ www.e	REGTECH & FIN/ COMPLIANCE Compliance Vero Vero Vero PARTNERSHIPS COMPLIANCE Vero Vero PARTNERSHIPS COMPLIANCE Vero Vero PARTNERSHIPS COMPLIANCE Vero Vero PARTNERSHIPS COMPLIANCE Vero Vero Vero Vero Vero Vero Vero Vero	ANCE BACK OFFICE SEA DUPARTIN A RPA AUTOMATION & RESHIF VIDA DO A Automatica Automatic	CURITY TANILM & EYLANCE CODE42 Conscious pindrop & exabean of SCINFY deSeare Bbitglass & Birtlan Conscious film Conscious film Conscious film Buberexagon Semile Conscious	Community Sector
			SPEECH & NLP	APPLICATIONS – INDUSTRY				
DATA TRANSFORMATION - talend @ pentoho alteryx @ THFACTA * tarmr @ Paxata * tarmr @ Paxata * tarmr @ Paxata * stream Sets Willer * stream Sets Wil	Certification Month Control Ning Scaling of the sector Scaling of the sector Scaling of the sector Provide the sector Scaling of the sector Spaning of the sector	Microsof Azure Microsof Azure Services datification Stream datificati	Coope Curve C	ADVERTISING	EDUCATION こは は にない の にない の にない で にない で にない にない で にない で の にない で の にない で の にない で の にない で の にない で の にない で の にの で の つ ろ つ の つ ろ の つ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ ろ つ つ つ つ つ つ つ つ つ つ つ つ つ	GOVT GOVELUGENCE - QPalantir Batantire Governor Gov	FINANCE - EI INVESTING KENSHO Cuantopian A cooper ISENTUM ALGORIZ APMORPACK APAGAWA	NDING Him Parties Markenditech A ∨ A N T Visance → Upstart ARDBANC → Upstart ArdBan
StorAde Weight Storade Storade CLUSTER SVCS Storade DATA GENERATION (ALBELLING Storade AIOP ALBELLING Storade AIOP Storade AIOP ALBELLING Storade AIOP ALBELLING Storade AIOP Storade AIOP	S GPU DBS & HARDWARE CLOUD TITIMITA al datino bytyt freezew termined.	SERRET LOGANALYICS SO ancourt in ancourt ancourt in ancourt in a	CIALANALYTICS Hootsuite spiritur ETBASE synthesio track itty ssimilarWeb	HEALTHCARE	HEADSTOP HEADST	TRANSPORTATION UBER TESLE Contraction Contraction UBER TESLE Contraction Contrection Contraction Contract	AGRICULTURE Auroro PILOTA Market Auroro PILOTA Agricultaria Auroro Aur	COMMERCE A Instance STITUTE TA STITUTE
	CROSS-INFRASTRUCTURE/	NALYTICS			twoRAR	N netradyne German Austalau Chvil Maps		VERDICRIS duetto Jukedeck Spectrum
							Hoveworks	
			OPEN SOURCE					
FRAMEWORKS OUERY/DATA FLOW DATA	ACCESS & DATABASES	N & MGMT STREAMING & MESSAGING Sport @ nifile Sport @ nifile Fink B bern & ration water & ration water B call a bern Apache RockerM2 State Andread	AI OPS AI / MACHINE LEAR ES & INFRA & INFRA		affe Eduration SEA elast wsum FeatureFu SC Elasti Park Control Control Instal Park Control	RCH LOGGING & MONITORIN easticsearch & kibana & logstash Orioneth Viruentbit Viruento O Virecou	G VISUALIZATION matplitib ress Grafana	COLLABORATION BeskerX Apache Ranger KNOX Sentry Sentry SecurriuLo
		DATA SOURCES & APIs					DATA RESOURCES	
HEALTH IOT	FINANCIAL & ECONOMIC DATA	AIR / SPACE / SEA	PEOPLE / ENTITIES LOCA	TION INTELLIGENCE	OTHER D	ATA SERVICES INCUBATO	RS & SCHOOLS	RESEARCH
👉 🐨 VALIDIC 🛛 🛞 GE Digital	Bloomberg 🔄 THOMSON REUTERS D	DOW JONES Orbital Insight planet.	acxi@m. experian.	RSQUARE SapAnything Omaphox	DATA.GOV D	OPERA DO MU SAMA	🚯 galvanize	OpenAI focebook research
fitbit GARMIN Ø helium 8 samsara	🚳 🧕 🕼 🖉 🖉 🖉 🖉	UNDERSTORY () tellusiabs	Crimson Hexagon Place	elle 😻 esri 🐺 factual.	Maglioscare f	ractaleee kaggle		
	SteckTwits xignite #Thinknum earne	WINDWARD DroneDea	ploy AASIS SAFEGRAPH	ebiq A Radar Streetine	(ligrafiti.io)		The Data Incubator METIS	

July 16, 2019 - FINAL 2019 VERSION

© Matt Turck (@mattturck), Lisa Xu (@lisaxu92), & FirstMark (@firstmarkcap) mattturck.com/data2019

FIRSTMARK





- Tail latency means some workers (always) finish late
- Q: How can MR work around this?
 - Hint: its approach to fault-tolerance provides the right tool

Resilience against stragglers

- If a task is going slowly (i.e., straggler):
 - Launch second copy of task on another node
 - Take the output of whichever finishes first

More design

• Master failure

Locality

• Task granularity

MapReduce usage statistics over time

Number of jobs				
Average completion time (secs)				
Machine years used				
Input data read (TB)				
Intermediate data (TB)				
Output data written (TB)				
Average worker machines				

Aug, '04	Mar, '06	Sep, '07	Sep, '09
29K	171K	2,217K	3,467K
634	874	395	475
217	2,002	11,081	25,562
3,288	52,254	403,152	544,130
758	6,743	34,774	90,120
193	2,970	14,018	57,520
157	268	394	488

* Jeff Dean, LADIS 2009

GFS usage at Google

- 200+ clusters
- Many clusters of 1000s of machines
- Pools of 1000s of clients
- 4+ PB filesystems
- 40 GB/s read/write load
 - In the presence of frequent hardware failures

* Jeff Dean, LADIS 2009

• What will likely serve as a performance bottleneck for Google's MapReduce used back in 2004 (or even earlier)? CPU? Memory? Disk? Network? Anything else?

- What will likely serve as a performance bottleneck for Google's MapReduce used back in 2004 (or even earlier)? CPU? Memory? Disk? Network? Anything else?
- How does MapReduce reduce the effect of slow network?

How does MapReduce jobs get good load balance across worker machines?

- Consider the indexing pipeline where you start with HTML documents. You want to index the documents after removing the most commonly occurring words:
 - 1. Compute the most common words;
 - 2. Remove them and build the index

What are the main shortcomings of using MapReduce to support such pipeline-like applications?



Today's outline

How can large computing jobs be parallelized?

1. MapReduce

2. Google File System

Review: MapReduce assumptions

- Commodity hardware
 - Economies of scale!
 - Commodity networking with less bisection bandwidth
 - Commodity storage (hard disks) is cheap
- Failures are common
- Replicated, distributed file system for data storage

Review: Fault tolerance

- If a task crashes:
 - Retry on another node
 - Why is this okay?
 - If the same task repeatedly fails, end the job

Review: Fault tolerance

- If a task crashes:
 - Retry on another node
 - Why is this okay?
 - If the same task repeatedly fails, end the job
- If a node crashes:
 - Relaunch its current tasks on another node
 - What about task inputs?

Google file system (GFS)

- Goal: a global (distributed) file system that stores data across many machines
 - Need to handle 100's TBs
- Google published details in 2003
- Open source implementation:
 - Hadoop Distributed File System (HDFS)



Workload-driven design

- MapReduce workload characteristics
 - Huge files (GBs)
 - Almost all writes are appends
 - Concurrent appends common
 - High throughput is valuable
 - Low latency is not

Example workloads: Bulk Synchronous Processing (BSP)



*Leslie G. Valiant, A bridging model for parallel computation, Communications of the ACM, Volume 33 Issue 8, Aug. 1990

MapReduce as a BSP system



- Read entire dataset, do computation over it
 - Batch processing
- Producer/consumer: many producers append work to file concurrently; one consumer reads and does work

Workload-driven design

- Build a global (distributed) file system that incorporates all these application properties
- Only supports features required by applications
- Avoid difficult local file system features, e.g.:
 - links

Replication



Replication



Resilience against failures



Resilience against failures





Replicating A to maintain a replication factor of 2



Replicating C to maintain a replication factor of 3



Machine may be dead forever, or it may come back



Machine may be dead forever, or it may come back





Data Rebalancing

Deleting one A to maintain a replication factor of 2





Data Rebalancing

Deleting one C to maintain a replication factor of 3



Question: how to maintain a global view of all data distributed across machines?
GFS architecture: logical view



GFS architecture: logical view



BTW, what is RPC?

• RPC = Remote procedure call

Motivation: Why RPC?

- The typical programmer is trained to write singlethreaded code that runs in one place
- Goal: Easy-to-program network communication that makes client-server communication transparent
 - Retains the "feel" of writing centralized code
 - Programmer needn't think about the network
 - Avoid tedious socket programming

What's the goal of RPC?

- Within a single program, running in a single process, recall the well-known notion of a **procedure call**:
 - Caller pushes arguments onto stack,
 - jumps to address of callee function
 - Callee reads arguments from stack,
 - executes, puts return value in register,
 - returns to next instruction in caller

What's the goal of RPC?

- Within a single program, running in a single process, recall the well-known notion of a **procedure call**:
 - Caller pushes arguments onto stack,
 - jumps to address of **callee** function
 - Callee reads arguments from stack,
 - executes, puts return value in register,
 - returns to next instruction in caller

RPC's Goal: make communication appear like a local procedure call: transparency for procedure calls – way less painful than sockets...

1. Client calls stub function (pushes parameters onto stack)



- 1. Client calls stub function (pushes parameters onto stack)
- 2. Stub marshals parameters to a network message



2. Stub marshals parameters to a network message

3. OS sends a network message to the server

Client machine	Server machine
Client process k = add(3, 5)	
Client stub (RPC library)	
Client OS proc: add I int: 3 I int: 5	Server OS

3. OS sends a network message to the server

4. Server OS receives message, sends it up to stub

Client machine
Client process k = add(3, 5)
Client stub (RPC library)
Client OS

S	Server machine
	Server stub (RPC library)
	Server OS
L	proc: add int: 3 int: 5

4. Server OS receives message, sends it up to stub

5. Server stub unmarshals params, calls server function

Client machine
Client process k = add(3, 5)
Client stub (RPC library)
Client OS

Server machine
Server process Implementation of add
Server stub (RPC library)
proc: add int: 3 int: 5
Server OS

5. Server stub unmarshals params, calls server function

6. Server function runs, returns a value

Client machine
Client process k = add(3, 5)
Client stub (RPC library)
Client OS

Server machine
Server process $8 \leftarrow add(3, 5)$
Server stub (RPC library)
Server OS

6. Server function runs, returns a value

7. Server stub marshals the return value, sends message

Client machine
Client process k = add(3, 5)
Client stub (RPC library)
Client OS

Server machine	
Server process 8 ← add(3, 5)	
Server stub (RPC library)	
Result I int: 8	
Server OS	

7. Server stub marshals the return value, sends message

8. Server OS sends the reply back across the network



8. Server OS sends the reply back across the network

9. Client OS receives the reply and passes up to stub

Client machine
Client process k = add(3, 5)
Client stub (RPC library)
Client OS
Result int: 8

Server machine	
Server process 8 ← add(3, 5)	
Server stub (RPC library)	
Server OS	

9. Client OS receives the reply and passes up to stub

10. Client stub unmarshals return value, returns to client



Server machine
Server process 8 ← add(3, 5)
Server stub (RPC library)
Server OS

Then, get back to GFS

GFS architecture: physical view



Data chunks

- Break large GFS files into coarse-grained data chunks (e.g., 64MB)
- GFS servers store physical data chunks in local Linux file system
- Centralized master keeps track of mapping between logical and physical chunks

Chunk map

Master	
chunk map	
logical	phys
924 521 	s2,s5,s7 s2,s9,s11

GFS server s2



GFS server s2

Local fs

chunks/924 => data1 chunks/521 => data2

• • •















File namespace



path names mapped to logical names

GFS architecture (original paper)



MapReduce+GFS: Put everything together



GMU CS571 Spring 2022