## **Big Data Processing**



جامعة الملك عبدالله للعلوم والتقنية King Abdullah University of Science and Technology

CS 240: Computing Systems and Concurrency Lecture 9

Marco Canini

### **Distributed Systems, Why?**

• BIG DATA really demands distributed systems!



Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018

# **Distributed Systems, Why?**

• BIG DATA really demands distributed systems!

Large-scale computing with:

- Scalability and parallelism
- Fault tolerance
- Load management
- Consistency (exactly-once processing guarantees)
- Transparency (programming abstractions and highlevel languages)

#### **BIG DATA Landscape evo**

© Matt Turck (@mattturck), John Wu (@john\_d\_wu) & FirstMark (@firstmarkcap)



#### 2021

		MACHINE LEAN	VING, ANTIFICIAL INTELLIGENCE, AND DATA (MAD) DATDSCAFE 2021	
INFRASTRUC	TURE	ANALYTICS	MACHINE LEARNING & ARTIFICIAL INTELLIGENCE	APPLICATIONS – ENTERPRISE
STORAGE HADOOP DATA LAKES	DATA STREAMING/	BI PLATFORMS VISUALIZATION	DATA SCIENCE DATA SCIENCE ML PLATFORMS	SALES MARKETING - MARKETING - B2C CUSTOMER EXPERIENCE / SERVICE HUMAN CAPITAL
SI S	icks	lobker 🐡 🔽 💷 🎲 + oble ou 🖬 Power81	School and the second	Atta 200 - 200 Alexandro pressue ACTIONO CODES CONTRACTOR CODES AND ALEXANDED AND A ACTIONO
O Territore Continues de Ann and Annie Statistica	ange		Binder colob	Ci Cleri (Rannessa Pristan Cinetta Cin
O mar transfer a worki	Contraction Contraction	Louis T. Progetiget - Dallar Ober	One of the Content Operator Operator Operator	A proce process of times attender States Commune Vietness Commune Transfer Commune Transfer Commune Co
A numero process?	Strin George Onteria	ATSCALE OIK @ 2001 Months and	A tore taxing Desparts \$888 TIBC9	Contractor Annual Annual Annual Annual Annual Annual State Order and Annual State Order
antitering CONCINY jethes Touten	California Association	C Othermotic		anter attorne) we whether 20 generative second bran draws take hims protect cettare discover and
Canada VAST CONTEA Ingrande OFF	Pluotol Informatic Elements Security Methods	G Gentre Based & Dirat	And	trodigin erigens wE40540 attestation (FEELEDE) Deven Sales Gravest Commen yold' Spring Typelowal moengage Fahrener
Ocure BM Indigney 2000	Origipate Monters Barrier	Manufitrategy Alean to SRID Preset	Streets EE2 Contain Axami Anamulo Edottoto datmo Sima Scogie e	televalyai prosta prostate parater and the company and
RDRMS NesSOL DATABASES NewSOL	DATABASES REALTIME GRAPH DBs	DATA ANALYST PLATFORMS AUGMENTED ANALYTICS	DATA GENERATION MODEL RUILDING FEATURE STORE DEPLOY MC	DELRAVEL #BOISCO COMPLIANCE AMPLAN BARN TANK COMPLIANCE AMPLAN BARN
	DATABASES		& LABELLING MENT & PRO- MC	NITORING Communic Oliro \$100 Channel Annual
CRACLE	SingleState	Microsoft @ penteho alteryx T. ThoughtSpot.	amagementercelure desires 141 Weights & Blases TOCION @ services DUCION & 0	Internal Provide Liberal Contraction Statistics and an and a statistics of the stati
TAN ANY ORACLE Inspects Makes	O Rivetal O imply IBM M	M HODE Datameer - 200	scale Upwork sopper "1; Corr dazo" C Google Coud	A Destroy Contra
P time Chatter	O Munice Towards (ROCKSET) ORACLE OPENER		Query Ligenmont Inteller Constant RASOO Dissertion Write Art	har PARTNERSHIPS One Trust Conduct Control on the Second S
Coethese prinstnik) spice	VOLTON STOOL IN ChildHouse	ENDOR in Riversi ATTIWO narrative Oscience	scortex distance in the same scortex	December 2000 2000 Contraction
The second second second	Martin formation	incorta. gu/wus @excelled enfacet macheve?	Sistion and the second stream Sistion	ETG
	Akinty Openay Storage	whether a strength of the start	Disperiela Standard Researce Baselor Course Course Lo	5/20 L
di AnnetOB 📫 CRATE.10	W reprinted to the second	switchessare operation and schoole cliff.of	A re menter W	APPLICATIONS - INDUSTRY
MPP DBsETL / ELT / REVERSE ETL	DATA INTEGRATION DATA GOVERNANCE	DATA CATALOS METRICS LOG ANALYTICS	COMPUTER VISION SPEECH NLP SYNTHETIC MEDI/	A ADVERTISING EDUCATION _ REAL ESTATE _ GOVT & COMMERCE FINANCE - LENDING INSURANCE -
TERADATA Census	Thestories @ Signat & Hornets	splunk>	The Manual Asses 🔮 Classi Carlos State Carlos State 💮 Service State Carlos State	Mundare and Anthe Reprint Repr
VERTICA Nightsuch	Contractor Contractor	Contors		DID. Control and the second se
Theres Stitch ALMATILLION	Garrier hierony schools TBM howers	Trace	OSynthesia ( Synthesia )	descript Critto, IASterne Greeker Statemer treases Thousand greeker statemer
ganon fans Sheeppe Chanacter A the Construction	Categories of traylo Straylo	solarworld solarwords kbana	Colland. Col	Ca Career Career Career Career Career Career Career Career
Cobon et an and a sector a reddentation	Contraction of the second seco	stemma Stemia Microsa	Tanton trar @ such Southard In. @Mond man @ Southing M prev	septence Genetara du FINAACE-INVESTING Generated as an and
Rognitio & marters and Starr @Alleges Official Librars	CATINITY ZALENT Import.io OKTRA MILLA	(+) SELECT STAR • transform	Dente antico Terre Olive 26 Orientic otres Univer	Linestyte albert guingum Stilleling Standbate Frence AYASDI KENSHC O Humpton Solid
Evened Million Steven Million Arounteese	Informarka anownaw @ flatfile	El Sereda d'Caster EQL	Sciences ACE Instantia State Sec. Madel In Orenation Orena & Instant V	VOCAD Data and VORDT GEORY CTUTION A STREET OF STREET
C. C		1 Obtaine		
PRIVACY & DATA OBSERVABILITY MGMT/MONITOR	LESS CLUSTER SWCS	ENGINE SLARCH	GPU DBS & CLOUD A HARDWARE	HEALTHCARE LIFE SCIENCES TRANSPORTATION AGRICULTURE INDUSTRIAL OTHER
Q VERV GOOD SECURITY Y CONTRACT MONTY CARLO	teu feit: 12 splank:	elazioaach elazioaach evela	Watson @ OpenAI O DeepMind KinEtico Google TPU Orm (	And A fatton ATAVUS GATABON Z AMAN T Serving T- UBER TESLA
PRIVABERA ANTA @ Oston D APPOYNAMICS	Orubsk Odynstate 🖉 America 🖄 🛲	V Greinio	O Vormant O V	UPHCORE Descent SOMed @rent TEMPUS Control States Control Asso Charts Control Asso
Forst Countries Country Statement Country Statements	st www.srocar octifio	Starburst	summer and attack with a	and a service spectral discussions and a service and a ser
Same Standard DATA QUALITY > Cribi 5740	anat determenter 0 d	🚖 ahasa 🗳 Lucidworks 🧿 swiftype ATTIV/O	A Namenta 🖸 Colleges Petroam	ALLO Dave stream units service tone and Rec Ores G7 @ Receiver
stoflow () talend \$998. 0 flow	S and a show	2 amplies	-GREEAL COLORED OSARO IL Affectivo	W. Cover June Birth Bran Break Instro
C Block Damer - District Contra	V source mudio packet Grooppon	St EXALEND in aprisense on incos	N-Steam Instanting	etre terminen sector getre demonitent sector getre demonitent comment
Allowatage	Cartering Concerned	WARADA MAANA & QUICKWIT CHAOSSEARCH	Paulon of I NESS FLORE CONTRACT	anteres and an and an and an an
	Advanta a second		V terms 900	
- 0PEN SOURCE				
FRAMEWORKS FORMAT QUERY/DATA FLOW	ACCESS	ORCHESTRATION INFRA- DATA OPS STREAMING & MESSAGING	STAT TOOLS & MLOPS & INFRA AI / MACHINE LEARNING / DEEP LEARNING	SLARCH LOGGING & MONITORING - VISUALIZATION COLLABORATION SECURITY -
Story Park, alleger and and alleger and	Uber Division	Anter Contractor Human	- Q Bons man (h) washes Flexorfice Man Standormers () Operie ;	10 10 Contract BERT Filters Calle Anter Solf" antered Extens more 35 00 terror more than the sole Anter Kinger Kin
The same drawn officer of the same		Contraction of Payton Contraction Contraction	A glance Gary Grant D. Control Control The Control Charton	And
	n Muli	DK Commenter Comments Operation De Mi		Contract Reader and Re
fielder Browney HELK	Cana Emeren annen Custon Opieci men ia	ined Deen I ogo Lerenteers Australians () Si	and the second process process and the second second space and	Mar Alarity (and the set of the s
DATA MARTINALS _ RANCAL & CORONC DATA _ ARL (SRC) SA _ ROULOUS J _ ROULDON SA _ ROU				
Sources Statement signite Statement of	windward Windwards	Quantcast Entris Supername	kar	nde ElectrifAl fractales Junz. de Datacine galvaniza 🛞 erris "reacter, 🔤 👘 🖓 datacine galvaniza
AC ADDRESS O 1908 O 2000 ADDRESS O 2				

mattturck.com/data2021

2012

© Matt Turck (@mattturck), John Wu (@john\_d\_wu) & FirstMark (@firstmarkcap)

Version 3.0 - November 2021

FIRSTMARK

- Batch vs streaming data
  - Is data available in full before its processing begins?
  - Is data produced incrementally over time?

- Generality vs specialization
  - A general system can be used for many different applications, but not ideally suited to any
  - A specialized system focuses on the needs of a class of application and takes advantage of their characteristics





Unified



#### **Data-Parallel Computation**

## Ex. Five top pages on class website

#### input: access.log

10.1.1.1 cs240.kaust.edu.sa - [05/Oct/2022:13:50:00 +0300] "GET <u>/course/CS240/assignment2</u> HTTP/1.1" 200 17618 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7) [...]"

# Write a MapReduce\* program that solves this problem

#### output

47 /course/CS240/assignment2
35 /course/CS240/assignment1
20 /courselist
18 /auth/page/kaust
4 /admin/CS240

\* NOTE: MapReduce automatically sorts by key the output of mappers

### MapReduce is a General System

• Can express large computations on large data; enables fault tolerant, parallel computation

But ...

- Fault tolerance is an inefficient fit for many applications Parallel programming model (map, reduce) within synchronous rounds is an inefficient fit for many applications
- The range of problems you can solve with a single MapReduce job is limited
  - Very common for MapReduce jobs to be chained into workflows

## Ex. Five top pages on class website

MapReduce workflows can be complex and tedious to write Can it be easier?

## Ex. Five top pages on class website

MapReduce workflows can be complex and tedious to write

Can it be easier?

What we wish to write ...

## MapReduce for Google's Index

- Flagship application in original MapReduce paper
- Q: What is inefficient about MapReduce for computing web indexes?
  - "MapReduce and other batch-processing systems cannot process small updates individually as they rely on creating large batches for efficiency."
- Index moved to Percolator in ~2010 [OSDI '10]
  - Incrementally process updates to index
  - Uses OCC to apply updates
  - 50% reduction in average age of documents

### **MapReduce for Iterative Computations**

- Iterative computations: compute on the same data as we update it
  - e.g., PageRank
  - e.g., Logistic regression
- *Q*: What is inefficient about MapReduce for these?
  - Writing data to disk between all iterations is slow
- Many systems designed for iterative computations, most notable is Apache Spark
  - Key idea 1: Keep data in memory once loaded
  - Key idea 2: Provide fault tolerance via *lineage* (record ops)

### **MapReduce for Stream Processing**

- Stream processing: Continuously process an infinite stream of data
  - e.g., estimating traffic conditions from GPS data
  - e.g., identify trending hashtags on twitter
  - e.g., detect fraudulent ad-clicks

• Q: What is inefficient about MapReduce for these?

### **Stream Processing Systems**

- Data is only produced incrementally over time
   Can't batch process it all at once!
- Streaming applications are long-running:
  - Definite computation ahead of time
  - Setup machines to run specific parts of computation and pass data around (topology)
  - Stream data into topology
  - Repeat forever (trickiest part: fault tolerance!)
- Specialization is much faster
  - E.g., click-fraud detection at Microsoft
    - Batch-processing system: 6 hours
    - w/ StreamScope [NSDI'16]: 20 minutes on average

### In-Memory Data-Parallel Computation

### **Spark: Resilient Distributed Datasets**

- Let's think of just having a big block of RAM, partitioned across machines...
  - And a series of operators that can be executed in parallel across the different partitions

- That's basically Spark
  - A distributed memory abstraction that is both fault-tolerant and efficient

### **Spark: Resilient Distributed Datasets**

- Restricted form of distributed shared memory
  - Immutable, partitioned collections of records
  - Can only be built through *coarse-grained* deterministic transformations (map, filter, join, ...)
  - They are called **Resilient Distributed Datasets** (RDDs)
- Efficient fault recovery using *lineage* 
  - Log one operation to apply to many elements
  - Recompute lost partitions on failure
  - No cost if nothing fails

### **Example: Log Mining**

• Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
messages.persist()
Bas Transformed RDD
Worker
Block 1
```

```
messages.filter(_.contains("foo")).count
messages.filter(_.contains("bar")).count
```



### **Efficient Fault Recovery via Lineage**



# **Generality of RDDs**

- Despite their restrictions, RDDs can express many parallel algorithms
  - These naturally apply the same operation to many items
- Unify many programming models
  - Data flow models: MapReduce, Dryad, SQL, ...
  - Specialized models for iterative apps: BSP (Pregel), iterative MapReduce (Haloop), bulk incremental, ...
- Support new apps that these models don't
- Enables apps to efficiently *intermix* these models

#### **Stream Processing**

### Simple stream processing



- Single node/process
  - Read data from input source (e.g., network socket)

#### – Process

- Write output

#### **Examples: Stateless conversion**



- Convert Celsius temperature to Fahrenheit
  - Stateless operation: emit (input \* 9 / 5) + 32

### **Examples: Stateless filtering**



- Function can filter inputs
  - if (input > threshold) { emit input }

### **Examples: Stateful conversion**



- Compute EWMA of Fahrenheit temperature
  - new\_temp =  $\alpha$  \* ( CtoF(input) ) + (1-  $\alpha$ ) \* last\_temp
  - last\_temp = new\_temp
  - emit new\_temp

### **Examples: Aggregation (stateful)**



- E.g., Average value per window
  - Window can be # elements (10) or time (1s)
  - Windows can be fixed (every 5s)



- Windows can be "sliding" (5s window every 1s)



### Stream processing as chain



### Stream processing as directed graph



## The challenge of stream processing

- Large amounts of data to process in (near) real time
- Examples
  - Social network trends (#trending)
  - Intrusion detection systems (networks, datacenters)
  - Sensors: Detect earthquakes by correlating vibrations of millions of smartphones
  - Fraud detection
    - Visa: 2000 txn / sec on average, peak ~47,000 / sec

# Scale "up": batching

#### **Tuple-by-Tuple**

input  $\leftarrow$  read

```
if (input > threshold) {
    emit input
```

#### **Micro-batch**

```
inputs \leftarrow read
```

```
out = []
```

```
for input in inputs {
```

```
if (input > threshold) {
```

out.append(input)

# Scale "up"

**Tuple-by-Tuple** 

Lower Latency

Lower Throughput

#### **Micro-batch**

**Higher Latency** 

Higher Throughput

**Why?** Each read/write is an system call into kernel. More cycles performing kernel/application transitions (context switches), less actually spent processing data.

### Scale "out"



#### Stateless operations: trivially parallelized



### State complicates parallelization

- Aggregations:
  - Need to join results across parallel computations



### State complicates parallelization

- Aggregations:
  - Need to join results across parallel computations



#### **Parallelization complicates fault-tolerance**

- Aggregations:
  - Need to join results across parallel computations



### Can parallelize joins

Compute trending keywords



### Can parallelize joins



#### **Parallelization complicates fault-tolerance**



Various fault tolerance mechanisms:

- 1. Record acknowledgement (Storm)
- 2. Micro-batches (Spark Streaming, Storm Trident)
- 3. Transactional updates (Google Cloud dataflow)
- 4. Distributed snapshots (Flink)

- 1. Record acknowledgement (Storm)
  - At least once semantics
  - Ensure each input "fully processed"
  - Track every processed tuple over the DAG, propagate ACKs upwards to the input source of data
  - Cons: Apps need to deal with duplicate or out-of-order tuples
- 2. Micro-batches (Spark Streaming, Storm Trident)
- 3. Transactional updates (Google Cloud dataflow)
- 4. Distributed snapshots (Flink)

- 1. Record acknowledgement (Storm)
- 2. Micro-batches (Spark Streaming, Storm Trident)
  - Each micro-batch may succeed or fail
  - On failure, recompute the micro-batch
  - Use lineage to track dependencies
  - Checkpoint state to support failure recovery
- 3. Transactional updates (Google Cloud dataflow)
- 4. Distributed snapshots (Flink)

- 1. Record acknowledgement (Storm)
- 2. Micro-batches (Spark Streaming, Storm Trident)
- 3. Transactional updates (Google Cloud dataflow)
  - Treat every processed record as a transaction, committed upon processing
  - On failure, replay the log to restore a consistent state and replay lost records
- 4. Distributed snapshots (Flink)

- 1. Record acknowledgement (Storm)
- 2. Micro-batches (Spark Streaming, Storm Trident)
- 3. Transactional updates (Google Cloud dataflow)
- 4. Distributed snapshots (Flink)
  - Take system-wide consistent snapshot (algo is a variation of Chandy-Lamport)
  - Snapshot periodically
  - On failure, recover the latest snapshot and rewind the stream source to snapshot point, then replay inputs

### **Graph-Parallel Computation**

#### **Properties of Graph Parallel Algorithms**

Dependency Graph



Factored Computation



Iterative Computation



#### **ML Tasks Beyond Data-Parallelism**

Data-Parallel

#### Map Reduce

Feature Cross Extraction Validation

> Computing Sufficient Statistics

#### **Graphical Models**

Gibbs Sampling Belief Propagation Variational Opt.

Collaborative Filtering Tensor Factorization Semi-Supervised Learning Label Propagation CoEM

**Graph-Parallel** 

#### **Graph Analysis**

PageRank Triangle Counting

### **Pregel: Bulk Synchronous Parallel**

Let's slightly rethink the MapReduce model for processing graphs

- Vertices
- "Edges" are really messages

Compare to MapReduce keys  $\rightarrow$  values?



### The Basic Pregel Execution Model

A sequence of *supersteps*, for each vertex V At superstep S:

- Compute in parallel at each V
  - Read messages sent to V in superstep S-1
  - Update value / state
  - Optionally change topology
- Send messages
- Synchronization
  - Wait till all communication is finished



### **Termination Test**

- Based on every vertex voting to halt
  - Once a vertex deactivates itself it does no further work unless triggered externally by receiving a message
- Algorithm terminates when all vertices are simultaneously inactive



### **Distributed Machine Learning**

# Machine learning (ML)

ML algorithms can improve automatically through experience (data)



- Most common approaches
  - Supervised learning:
  - Unsupervised learning:
  - Reinforcement learning (RL): model learns while doing

#### Training

Feed the ML model data, so that it can learn how to make decisions

#### Inference (or model serving)

the model learns by itself

ML model in use, to process live data

train the model first, then use it

### **ML training**



#### Distributed ML training Data parallel



#### Distributed ML training Model parallel or hybrid



#### Hybrid model-data parallel



### Weak scaling and strong scaling

#### Weak scaling

- Fixed local batch size perworker fixed
- More workers can process a larger global batch in one iteration
- Same iteration time, fewer iterations
- Same data transfers at each iteration
- Time to accuracy does not scale linearly with the number of workers

#### **Strong scaling**

- Fixed global batch size
- With more workers, the local batch size per-worker decreases
- Reduced iteration time (for computation)
- Same data transfers at each iteration
- More frequent synchronizations among workers (more network traffic)

## **Beyond training: AI applications**

- Different applications of AI have their specific computational tasks
- Based on these tasks, they impose some system requirements
- Ex. supervised learning application:

The stateful training task

The stateless prediction task

Impose system requirements (training stage): Tensorflow, MXNet and Pytorch

#### **ML Ecosystem**



#### **ML Ecosystem**



Emerging AI Applications require stitching together multiple disparate systems to satisfy diverse computation requirements

## Ray: unified framework for AI apps

 Goal: do all the tasks of training, serving and simulation together by a single framework



**Requirements:** 

- Distributed training: fine-grained computations, heterogeneous computations
- Serving: latency-sensitive, fine-grained computations, heterogeneous computations
- Simulations: dynamic execution

## Ray: unified framework for AI apps

- Provides a general programming model supporting task-parallel and actor-based computations
- Supports a range of computations: from lightweight and stateless computations (simulations) to long and stateful computations (training)
- Provides low latency, high scalability and fault tolerance