## Announcements

- For everyone's safety:
  - Please do not congregate after the class for Q/A -- ask questions during the lecture or make use of Piazza and OH
  - If you are sick, please watch the lectures remotely
  - Wear your mask properly covering your nose and mouth entirely at all times during the lecture
- For any private communication, use course staff email < ds-stafff21-private@lists.andrew.cmu.edu>. Not individual instructor email addresses.

# 15-440/640 Distributed Systems

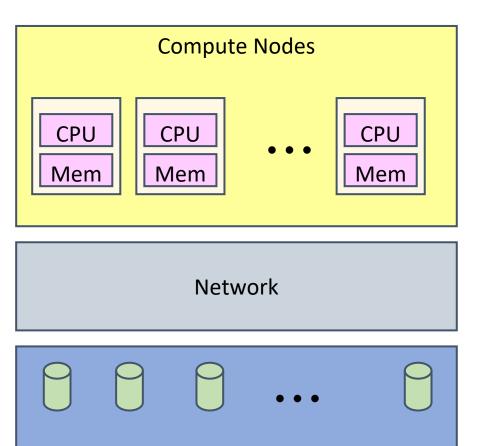
- Finish up distributed computation (MPI & MapReduce)
- In-memory cluster compute (Spark)
- Distributed ML

## **Cluster Computing**

- 1. High-performance computing (HPC)
  - Message Passing Interface (MPI)

- 2. Cluster computing
  - MapReduce

# **Recall: Typical HPC Machine**

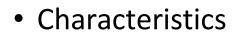


**Storage Server** 

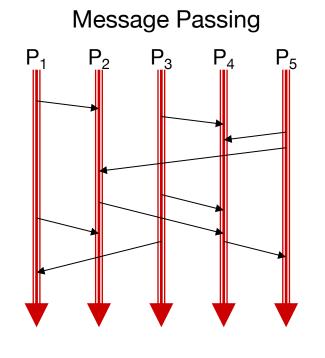
#### Compute Nodes

- Lots of high end processor(s)
- Lots of RAM
- Network
  - Specialized
  - Very high performance
- Storage Server
  - RAID-based disk array

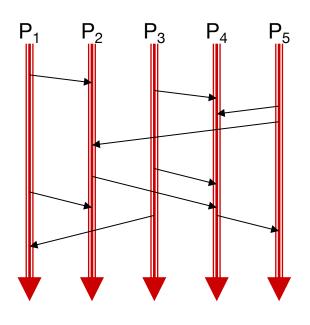
# **Recall: Typical HPC Operation**



- Long-lived interdependent processes
- Partitioning: exploit spatial locality
- Hold all program data in memory (no disk access)
- High bandwidth communication
- Strengths
  - High utilization of resources
  - Effective for many scientific applications
- Weaknesses
  - Requires careful tuning of application to resources
  - Intolerant of any variability

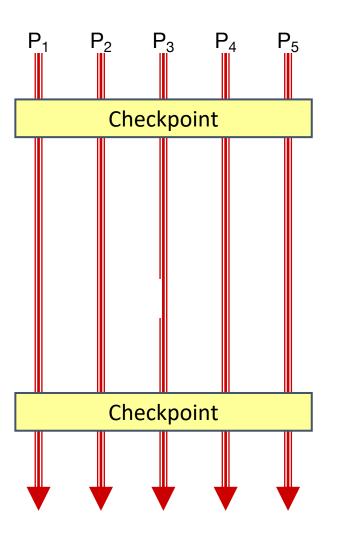


## **HPC Fault Tolerance**



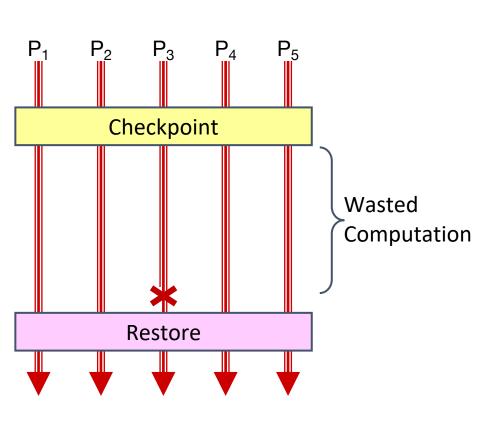
- Tightly coupled processes
  - Failure of one processes prevents all others from progressing
- How to ensure correct execution in presence of failures?

# **HPC Fault Tolerance**



- Tightly coupled processes
  - Failure of one processes prevents all others from progressing
- How to ensure correct execution in presence of failures?
- Checkpointing
  - Periodically save system state of all processes
  - Stored in reliable storage that can withstand targeted failure
  - Roll back to error-free state in case of failure

# **HPC Fault Tolerance**



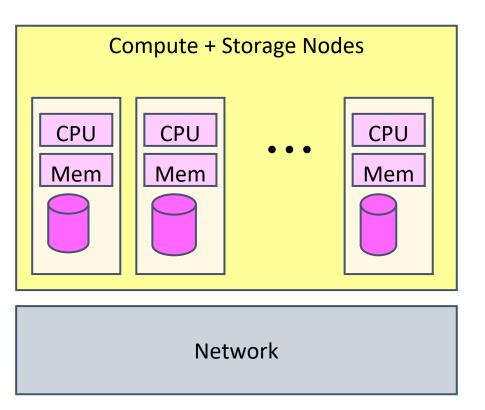
- Rollback upon failure
  - Restore state to that of last checkpoint
  - All intervening computation wasted
- Design decisions
  - Asynchronous or synchronous?
  - How often to checkpoint?
  - What data to checkpoint?
  - Who checkpoints: application or system?
- Significant I/O traffic
- Very sensitive to number of failing components

## **Cluster Computing**

- 1. High-performance computing (HPC)
  - Message Passing Interface (MPI)

- 2. Cluster computing
  - MapReduce

# **Typical Cluster Computing**



- Off-the-shelf servers
  - Collocation of compute and storage
  - Medium-performance processors
  - Modest memory
  - A few disks
- Network
  - Conventional Ethernet switches
  - 10s Gb/s

## **Oceans of Data, Skinny Pipes**

- 10 Terabytes
  - Easy to store
  - Hard to move

Disks	MB / s	Time
Seagate HDDs	~100s	> Few hours
Networks	MB / s	Time
Gigabit Ethernet	< 125	> 23 hours
10GE	< 1,200	> 2.4 hours
100GE	< 12,000	15 minutes

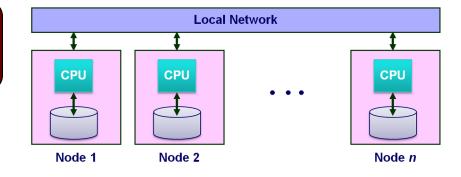
## **Data-Intensive System Challenge**

How to process 10 TB in a few minutes?

- Distribute data over 100+ disks
  - Assuming uniform data partitioning

Key idea: partition compute tasks and run where data is stored

- Compute using 100+ processors
  - Without having to move data
- System Requirements
  - Lots of processors with co-located disks
  - Nodes located in close proximity
    - Within reach of fast, local-area network



## **How To Program A Cluster?**

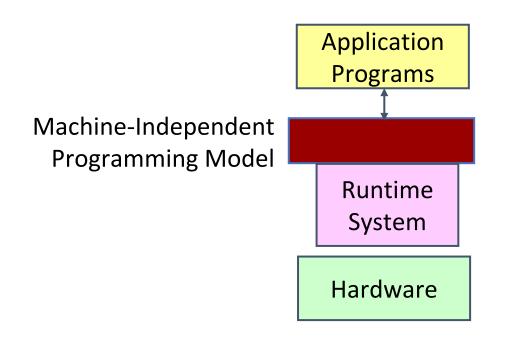
#### Example:

Many text files (e.g. logfiles, crawled webpages,..) Stored in DFS on thousands of machines (GFS) Assume you have access to all those machines

How do you find the frequency of words, such as , "440", "error", "p4" ?

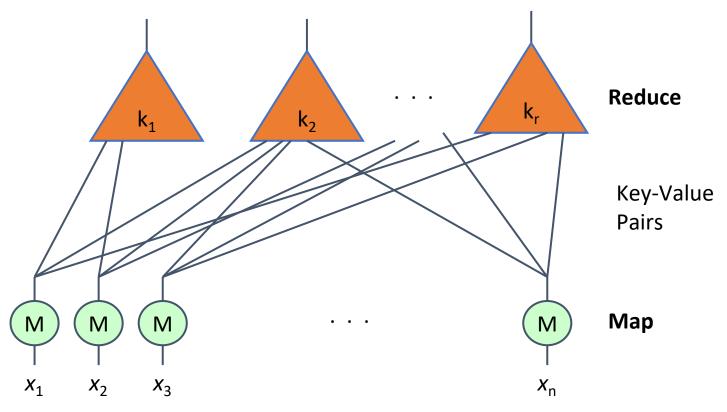
What do you do if tasks run for > 1 week? e.g., machines fail, get rebooted What do you do if a variant of this task comes up?

# **Cluster Programming Model**



- Application programs written in terms of high-level data operations
- Runtime system controls scheduling, load balancing, fault-tolerance
- This is idealized: no perfect cluster programming model, in practice
- One popular model: MapReduce

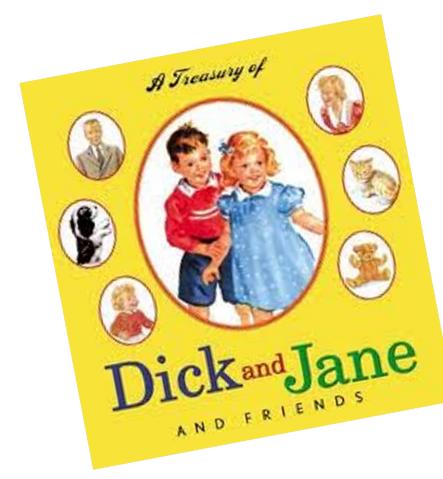
## **MapReduce Cluster Model**



- Map: Map computation across many objects
  - Runtime schedules "mappers" so as to minimize data movement
- Reduce: Aggregation of results

## **Example MapReduce**

- Calculate word frequency of a set of documents
- Example: children book in basic English



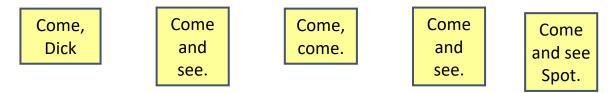


Come, Dick. Come and see. Come, come. Come and see. Come and see Spot.

# **Example MapReduce**

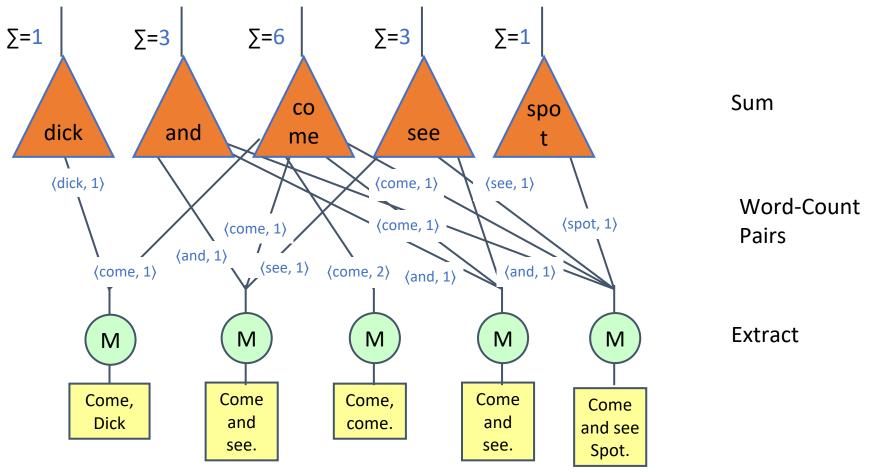


Come, Dick. Come and see. Come, come. Come and see. Come and see Spot.

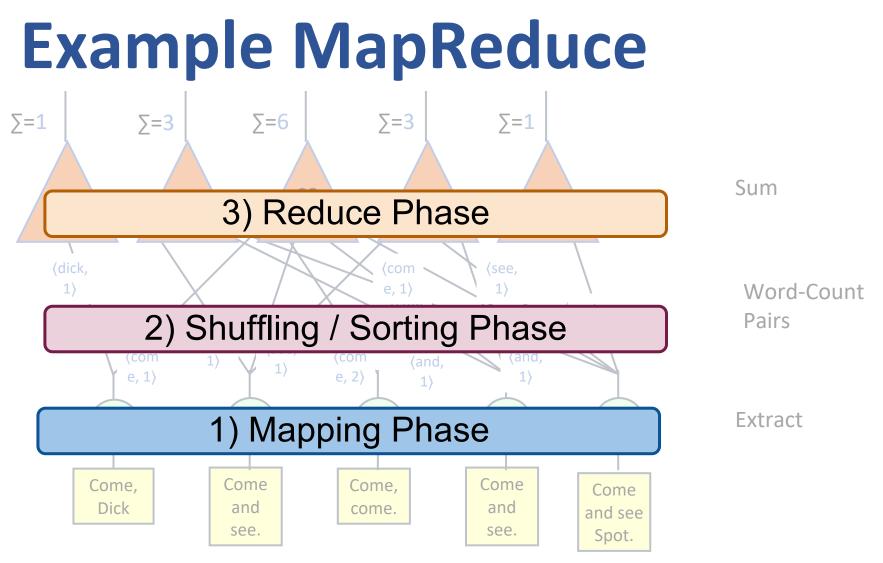


• Calculate word frequency of set of documents

## **Example MapReduce**



- Map: generate (word, count) pairs for all words in document
- Reduce: sum word counts across documents



- Map: generate (word, count) pairs for all words in document
- Reduce: sum word counts across documents

## **MapReduce Implementation**

- Built on Top of Cluster Filesystem
  - Provides global naming
  - Reliability via replication (3 replicas of every chunk)
- Breaks work into tasks
  - Typically #tasks >> #processors
  - Master schedules tasks on workers dynamically
- Net effect
  - Input: Set of files in reliable file system
  - Output: Set of files in reliable file system

## **Real-World Challenges**

Fault Tolerance

- Reliable file system is not enough
- Workers can fail even if input files available
- Map-Reduce solution
  - Detect failed worker (Heartbeat mechanism)
  - Reschedule failed task

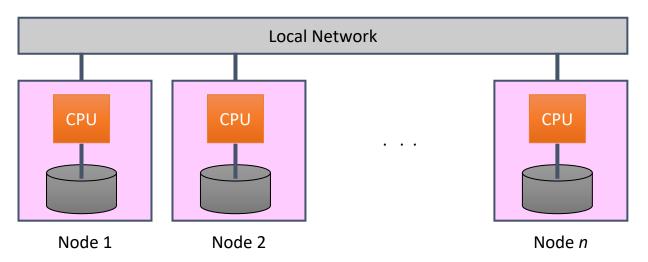
Stragglers

- Tasks that take a long time to execute
  - Might be bugs, flaky/slow hardware (e.g., disk I/O), poor partitioning, etc.
- Map-Reduce solution:
  - When done with most tasks, reschedule any remaining executing tasks
  - Keep track of redundant executions
  - Significantly reduces overall run time

# Hadoop Project

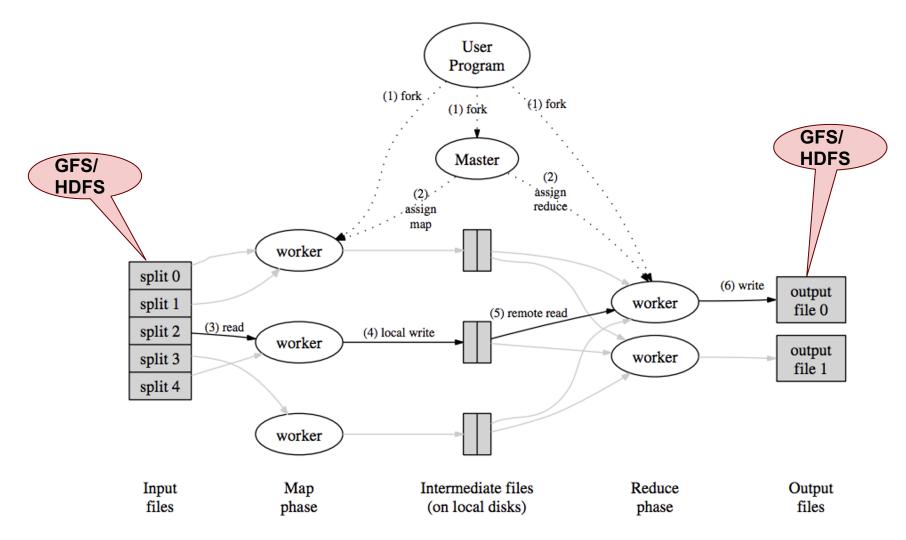


• Colocate compute and storage: HDFS + MapReduce



- HDFS Fault Tolerance (3 copies of file)
- "Locality-preserving" compute job placement priority order
  - 1) On same node as HDFS chunk
  - 2) On same rack as HDFS chunk
  - 3) Anywhere else (access over HDFS network)
- MapReduce programming environment

## **MapReduce Execution**

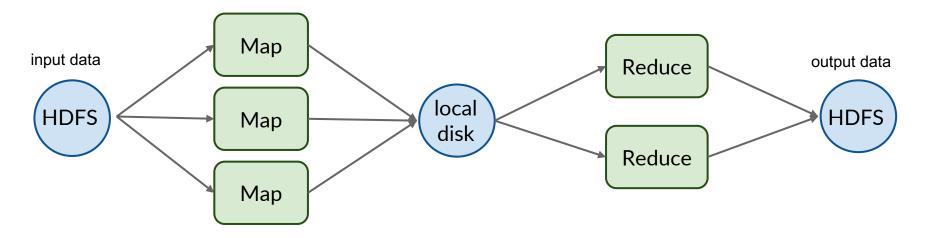


Dean & Ghemawat: "MapReduce: Simplified Data Processing on Large Clusters", OSDI 2004

15-440/640 Carnegie Mellon University

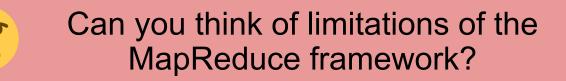
### **Cluster Computing**

#### MapReduce (Hadoop) Framework:



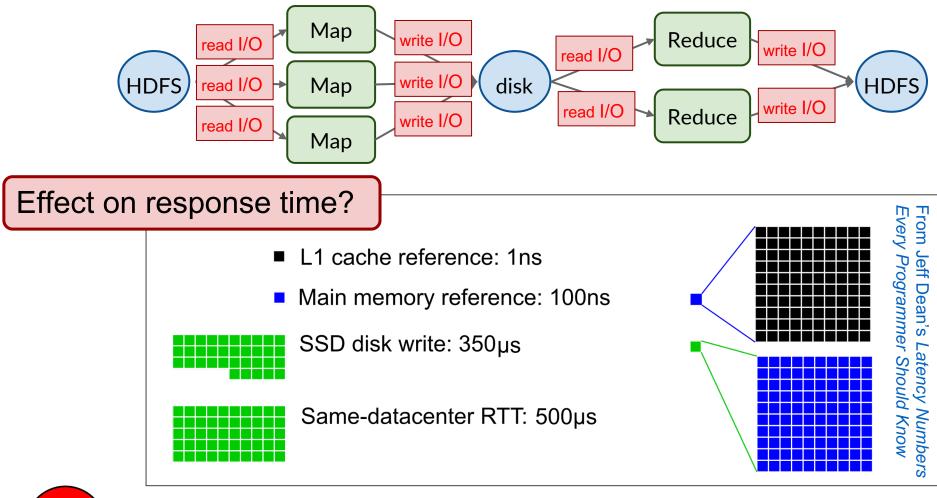
Key features: fault tolerance and high throughput

 $\Rightarrow$  Simplified data analysis on large, unreliable clusters



## Limitations of MapReduce I

Store input/output after every step on disk

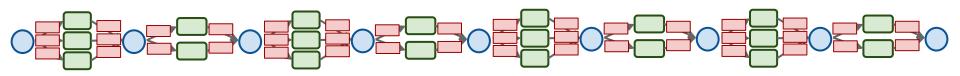


I/O penalty makes interactive data analysis impossible

15-440/640 Carnegie Mellon University

### Limitations of MapReduce II

Many applications require iterating MapReduce steps

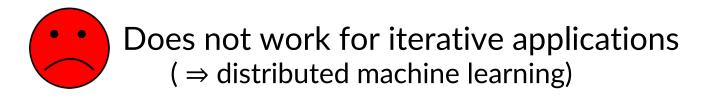


Each iteration steps is small.

But: we need many iterations

 $\Rightarrow$  90% spent on I/O to disks and over network

 $\Rightarrow$  10% spent computing actual results



### Limitations of MapReduce III

MapReduce abstraction not expressive enough

Explosion of specialized analytics systems

 Streaming analytics, Iterative ML algorithms, Graph/social data

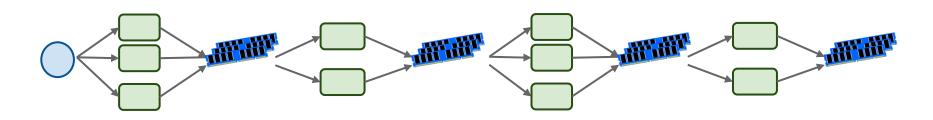


# 15-440/640 Distributed Systems

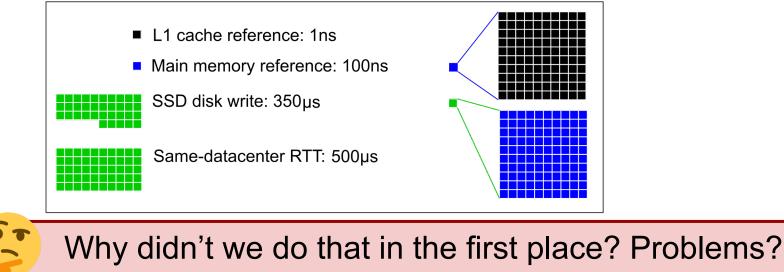
- Finish up distributed computation (MPI & MapReduce)
- In-memory cluster compute (Spark)
- Distributed ML

### Apache Spark: In-Memory Computation

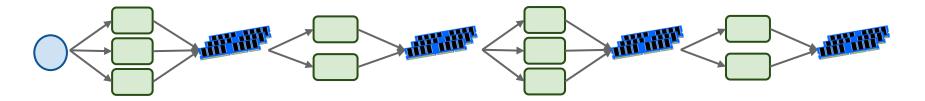
Key idea: keep and share data sets in main memory



#### Much faster response time (in practice: 10x-100x)



### In-memory computation and data-sharing



How to build **fault-tolerant** and **efficient** system?

Fault tolerance techniques from lectures so far?

- Logging each operation to node-local persistent storage
- Replicating data across nodes (+ persistent storage)
- Checkpointing (checkpoints need to be stored persistently)

 $\Rightarrow$  Expensive (10-100x slowdown)

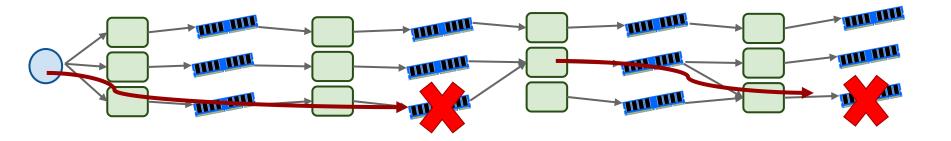
How common are fine-grained (bit-level) data updates?

### Spark Approach: RDDs and Lineage

#### **R**esilient **D**istributed **D**atasets

Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. NSDI 2012.

- Limit update interface to coarse-grained operations
  - O Map, group-by, filter, sample, ...
- Efficient fault recovery using lineage
  - Small partition size  $\rightarrow$  individual operations are cheap
  - Master server tracks coarse-grained operator sequence
  - Recompute lost partitions on failure



## **RDD Consistency and Fault Recovery**

#### RDDs are **immutable** datasets

- Deterministic functions of input
   o recreate any RDD any time
- Simplifies consistency (caching, sharing, ..)
- Still need periodic RDD checkpoints
  - stored persistently on disks/ in HDFS



	Relieled Docs: object RDD   packag
	extends Sarializable with Loging
Kanifest Distributed Delayer	PTD do basis abroaction in Seath Biorements an investible antificiant effective of discourts for each a sum of a seather that each a seather that each a seather that each a sum of a seather that each a s
nd persist. In addition, and (DEs of Doubles; and org oper rigicit.	paths panils all Maril Cale analysis contains constained only on HCDs of key-value pairs, such as provedy for and joint can panels such and Calebra Calebra and the second second and Sequence Field All operations are assessed by BCD of the operation and and Sequence Field. All operations are assessed by BCD of the operation are assessed by BCD
nemally, each RDD is characte A lite of positions	
A function for computing each A last of dependencies on of Opendencies on of	
	y volue (Thio) (a) as any more the CRD is have posttanced. Notation is to consult wait split on (e.g. block hoadmon for an HDPS Re) on a Roufs is not not on these memory, about good HDD is inglement is own way of comparing tool, index, work can inglement owner HDDs (o.g. to randing data from a new texcep nyterin) by over
rese functions. Please refer to	The <u>South paper</u> for more details on PDD Internats.
earce BDD.scs Uncer Superhypes	3
Known Subclasses	
a	
ndering (Rhitsderic) by s	belaco-
nheiled (830) (1239/19)	(Sensizate) (Sensizate) anyor ary
HOLAL (Share)	Ð.
Ashiry (Public) Al	
nstance Constructors	mm(metwyer: std())((sh(c)r.sven.classta)t))
15	Construct on FCOS with just a one-to-one dependency on one parent
	MMQ_sc: garktentext, deps: Sou[Expendency[_])(Implicit arge: classTag[T])
ibstract Value Members	compute(split: Pertition, context: Tesk(entext): Iterater(T)
	Implemented by subclasses to compute a given partition.
Concrete Value Members	
def	++(other: K00(11): K00(11) Return the union of this K00 and exother one.
601	aggregate(U)(zeroislue: U)(sopp: (U, T) = U, combin: (U, U) = U)(seplecif arg8: ClassTog(U)): U
	Aggregate the elements of each partition, and then the results for all the partitions, using given combine functions and a restrict "new value". capital () - 160, 1515, Trave
	Persist this RCD with the default storage level (HDNMY_DNLY).
def	cartesian(U)(other: 00010)(inv[icit and (CostTa(U)): 000(T, U)) mean the Carroson product of this PETC and another area that is, the PETC of all pairs of elements (a, b) where a is in this card b is in other.
601	checkpoint(): Unit
	Nam to NEO to medipering. celesce(numPartitions: Int, shuffle: Boolean = false, partitionCollescer: OptionPartitionCollescer = dotion.empty(ideolacit ord: Ordering[7] = multi: NEO
	Return a new RCD that is reduced into numPartitions and the second
	estLect(U)(1: PartialVancium(1, U))(dapticit arg0: Classifie(U)): (82(U) Return an RDD that constain all matching values by opplying f.
	cellect(): Array[T] Return an anay that contains all of the elements in this RDD.
def	centext: SourkContext The erg approximate K (Source that this RSD was exceeded on.
def	count(): Long Return the number of elements in the DCD.
def	ceastApprex(timecut: Long, confidence: Double = 0.05): <u>PartialResult(BoundedDouble)</u> Approximate ventor of cound) that return a potentially incomplete result within a transco, even if not all table have finited.
def	count/pproxDistinct(relativeSD) Double = 0.05); Long
def	Datas approvinge sensor el dister cimento en la DDD. centagenerabis(s)(x(1), 10, 11, 10) Leg Datas approvinge minor el dister cimento el be DDD.
def	<pre>country(wlue()(inplicit ord: Ordering[T] = mull: Hup[T, Long]</pre>
def	Refer the court of each unique wake in the RECE as a local map of (make, courd) parts. countByvalumespersa(timesus: coup, confidence: Double = 0.40)(implicit ord: trubering[1] = mull): <u>persialmenuls[Mep[1, moundedDouble]]</u>
final def	Approximate version or country/valacity. dependenceLes :: Kerg(tergendency) 11
	Get the tot of dependencies of this RDD, skilling into account whether the RDD is checkpointed or not. distilance(1): 100111
	Return a new RCD containing the distinct elements in 2hs RCD.
def	distinct(sumPartitions: int)(implicit and: drdering[1] = null): 800[1] Return a new HCD containing the distinct elements in this HCD.
def	filter(f) (Y) = &olenn); 800(Y) Return a new RDD contains only the determine that satisfy a predicate.
def	Parallel a new race consuming our parallel and an enter a province. farat () : : : : : : : : : : : : : : : : : :
def	FLatMap[U] (1: (1) - TraversableOnce(U)) (splict sign: Classing(U)): R00[U] Return a new R00 by first applying a function to all elements of this R00, and then flattening the results.

- High overhead: copying data (no mutate-in-place)
- Needs lots of memory (might not be able to run your workload)

### Towards a New Unified Framework

Two Goals

1. In-memory computation and data-sharing

10-100x faster than disks or network

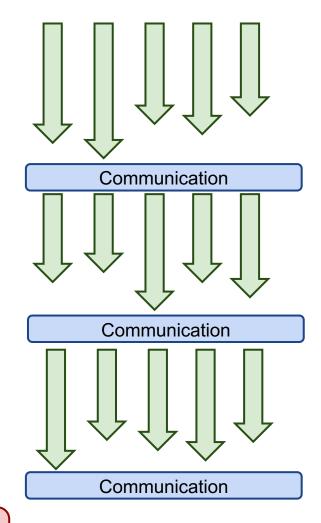
• Key problem: fault tolerance

- 2. Unified computation abstraction
  - Power of iterations ("local work + message passing")
  - Key problem: ease-of-use and generality

## **BSP computation abstraction**

- Surprising power of iterations
  - (e.g., iterative Map/Reduce)
- Explained by theory of bulk synchronous parallel (BSP) model

<u>Theorem (Leslie Valiant,1990):</u> "Any distributed system can be emulated as local work + message passing" (=BSP).



#### Spark implements BSP approximately

### Spark as a Uniform Framework

Graph processing like

GraphLab/Pregel on Spark (Bagel)

 $\Rightarrow$  "200 lines of Spark code"

Iterative MapReduce

 $\Rightarrow$  "200 lines of Spark code"

Hive SQL on Spark (Shark)

 $\Rightarrow$  "500 lines of code"



ML-lib and other distributed ML implementations



### Should You Always Use Spark?

Some examples for which Spark is not a good fit for

- Applications with fine-grained updates to shared state
- Datasets that don't fit into memory

# 15-440/640 Distributed Systems

- Finish up distributed computation (MPI & MapReduce)
- In-memory cluster compute (Spark)
- Distributed ML

## Machine Learning

#### The ML hype

Machine learning: the power and promise of computers that learn by example



#### Enabled by huge leap in parallelization



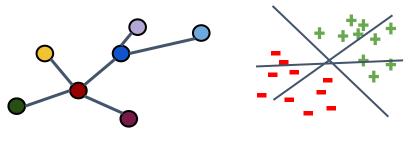


#### ML systems out scale even powerful machines (GPUs et al) => Distributed ML

ROYAL

SOCIET

### What Do ML Algorithms look like?



Page Rank

Regression

Other examples: Bayes, K-means, Neural Networks...

Common feature when computing these algorithms?

Eric Xing, Strategies & Principles for Distributed Machine Learning, Allen Al, 2016

#### Three key challenges:

1) lots of data

2) lots of parameters

3) lots of iterations

## **Distributed Machine Learning**

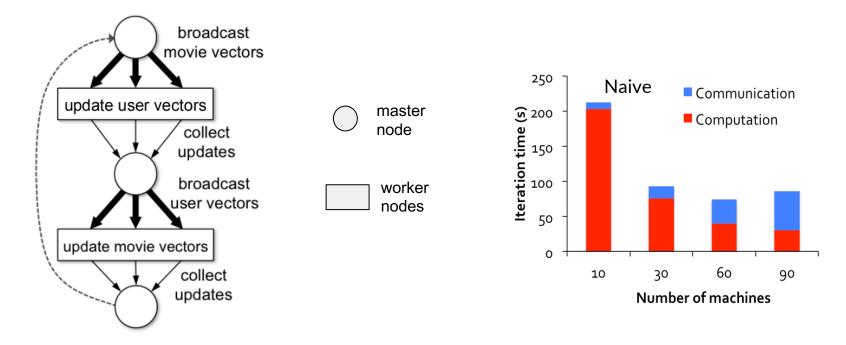
Data/model often fits into 10s-100s of nodes

Goal: more iterations / sec . deal speedup Iterations / Sec good speedup pathetic speedup

#### Machine Count

## Challenge of Communication Overhead

- Communication overhead scales badly
- E.g., for Netflix-like recommender systems

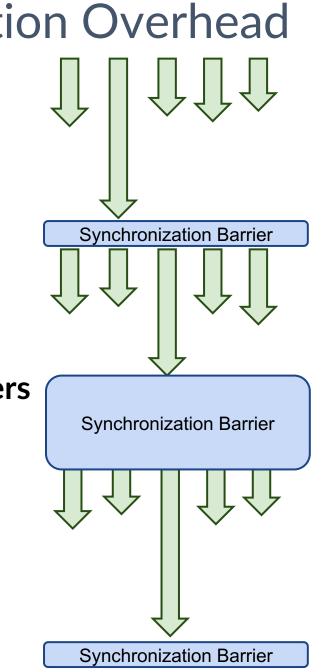


## Challenge of Synchronization Overhead

BSP model:

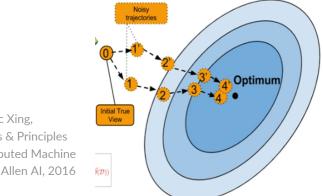
- No computation during barrier
- No communication during computation

Fundamental limitation in BSP model Constantly waiting for **stragglers** 



### **Relaxing BSP Consistency**

Idea: nodes can accept slightly stale state

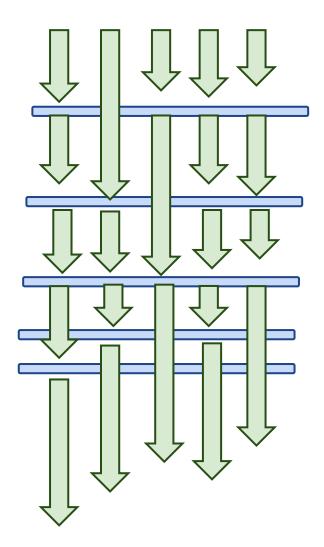


From: Eric Xing, Strategies & Principles for Distributed Machine Learning, Allen Al, 2016

ML algorithms are robust

 $\Rightarrow$  converge even with some stale state

How can we incorporate stale state into the BSP model?



### **Opposite Extreme: No Synchronization**

What if we fully remove BSP's synchronization barriers?

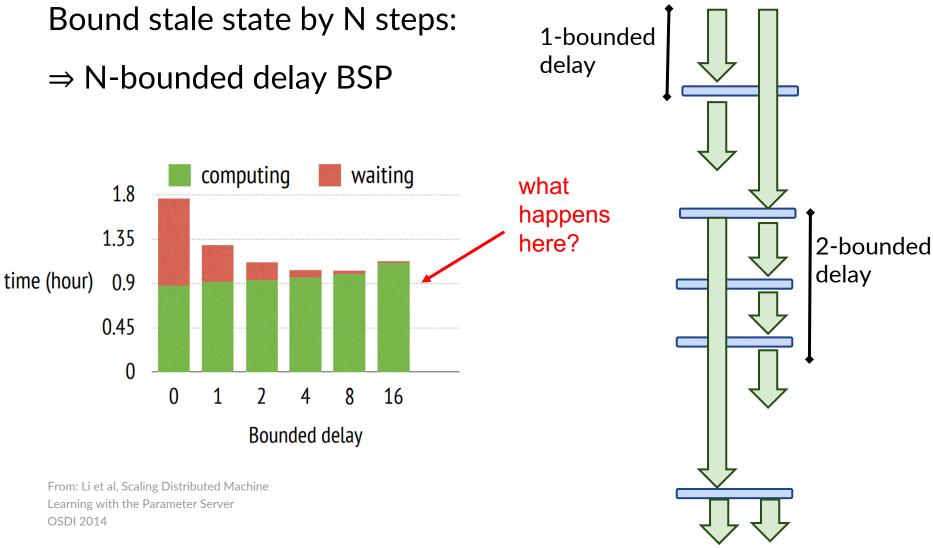
Asynchronous communication:

- no communication at all, or
- communication at any time

Observation through experiments: Iterative algorithms won't converge

XX V	

### Bounded-delay BSP for Distributed ML



### Many Challenges Remain

Trade-Off:

Stale state -> throughput (iter / sec)

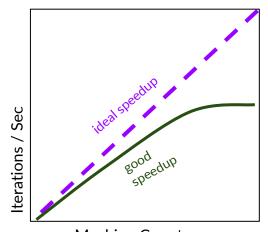
Misleading design decisions:

Higher throughput

Less progress / iteration

Many open challenges

Automatic model partitioning



Machine Count



october 12, 2017 CMU Spinoff Petuum Receives \$93M in New Round of Funding

How to schedule many parallel jobs on ML clusters

How to build a framework for interactive ML

applications

15-440/0402 am Vieroyunactive field of research