Algorithmic approaches to big data

Matthew J. Graham, CD^3, Caltech

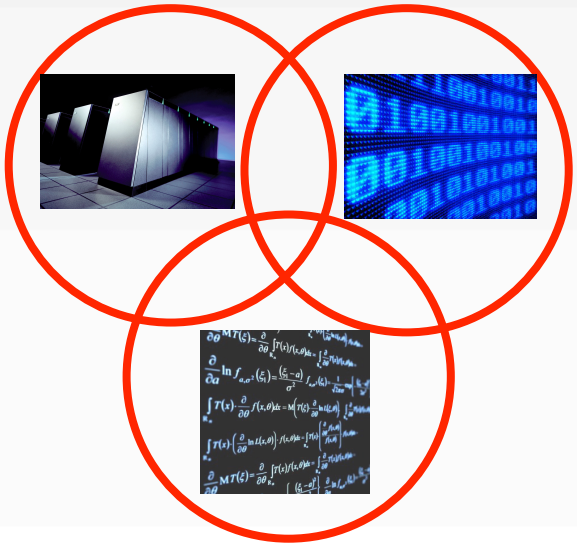
Divide-and-conquer

- Many (most) computations (in astronomy) are pleasantly/ embarrassingly parallelizable
- The key is:

$$y = \sum f(x)$$

``reduce'' ``map''

- Realizable as:



Where is the data?

- "Beyond 300TB is difficult" – Alex Szalay
- Assume networking sufficient to cope !!!
- Distributed file systems
  - Lustre
  - HDFS: inspired by Google FS
- Data store (beyond RDBMS)
  - Qserv:
    - MySQL + Xrootd, shared-nothing (LSST)
  - SciDB
    - Column-oriented db; arrays rather than tables; maintains ACID
  - NoSQL
    - Largely optimized key-value stores; not ACID; web scale transactions
  - NewSQL (H-Store, Google Spanner)
    - Relational model + SQL; sharding middle layer

Hardware

- "This is a different paradigm than solving PDEs"
- The Cloud
  - On demand computing – deploy what you want when you need it
  - Not necessarily cost effective
- GPU – your own personal supercomputer:
  - 192 cores per multiprocessor * 13 = 2496 (high-end, e.g. Kepler)
  - Each core can run 16 threads (~40k threads/GPU)
  - Threads are lightweight so can easily launch ~billion threads/sec
  - Favours brute force approach rather than ported smart algorithms
  - Memory issues (getting worse)

The MapReduce paradigm

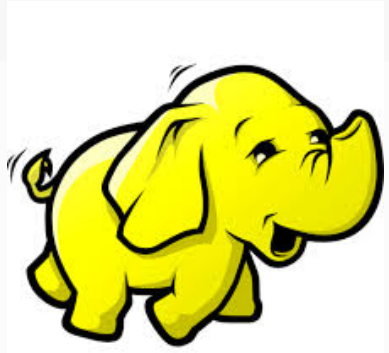
- Primary choice for fault-tolerant and massively parallel data crunching
- Large-scale data processing hammer of choice
- Invented by Google fellows
- Based on functional programming map() and reduce() functions
- Reliable processing even though machines die
- En-large parallelization – thousands of machines for tera/ petasort – 1PB in ~~6.8 hrs~~ 234 min. (Spark)

MapReduce in astronomy

- Image Coaddition Pipeline (Wiley et al. 2011)
  - Evaluated image coaddition of 100000 SDSS images using Hadoop
  - Five possible methods of implementation with progressive improvements
  - Intend to develop full petascale data-reduction pipeline for LSST (?)
- Berkeley Transient Classification Pipeline (Starr et al. 2010)
  - Make probabilistic statements about transients making use of their light curves the event occurs on the sky ("context") particularly with minimal data from survey of interest
  - Resampled ("noisified") well-sampled well-classified sources with precomputed candences, models for observing depths, sky brightness, etc. + generate classifiers for different PTF cadences
  - Uses Java classifiers from Weka direct with Hadoop; Python code with Hadoop Streaming; Cascading package; plan to use Mahout and Hive
- Large Survey Database (AAS 217 poster)
  - $>10^9$ rows, >1 TB data store for PS1 data analysis
  - Used by PTF
  - In-house MapReduce system

One stop solution: Hadoop

- HDFS: distributed file system
- HBase: column-based db (webtable)
- Hive: Pseudo-RDB with SQL (HiveQL)
- Pig: Scripting language, abstracts MapReduce (a la SQL)
- Zookeeper: Coordination service
- Whirr: Running cloud services
- Cascading: Pipes and filters
- Sqoop: RDB interface
- Mahout: ML/DM library



MapReduce for big analytics?

	One iteration	Multiple iterations	Not good for MR
Clustering		k-means	
Classification	Naïve Bayes, kNN	Gaussian mixture	SVM, HMM
Graphs		PageRank, connected component	
Information retrieval	Inverted index		

- Single pass
- Keys uniformly distributed

- Small shared information synchronized across iterations
- Multiple passes
- Intermediate states are small

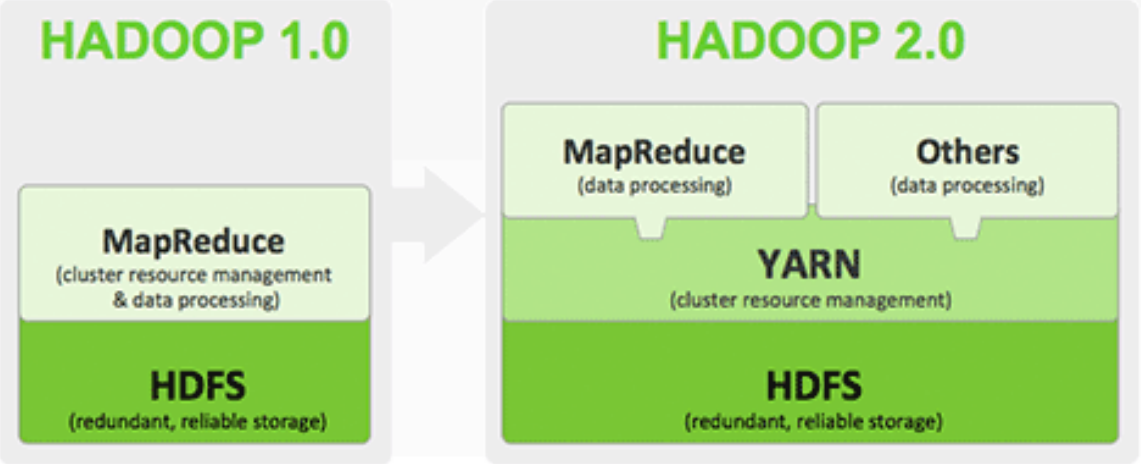
- Large shared information
- Lot of fine-grained synchronization

Alternates to MapReduce (NoHadoop)

- Percolator
  - Incrementally update massive data set continuously
- Apache Hama
  - Implementation of BSP (Bulk Synchronous Parallel)
  - Alternate to MPI, smaller API, impossibility of deadlocks, evaluate computational cost of an algorithm as function of machine parameters
- Pregel:
  - Very large graphs (billions of nodes, trillions of edges)
  - Uses BSP
  - Computations are applied at each node until
  - Cross-matched catalogs (GAIA, LSST, SKA)

Hadoop 2.0: YARN

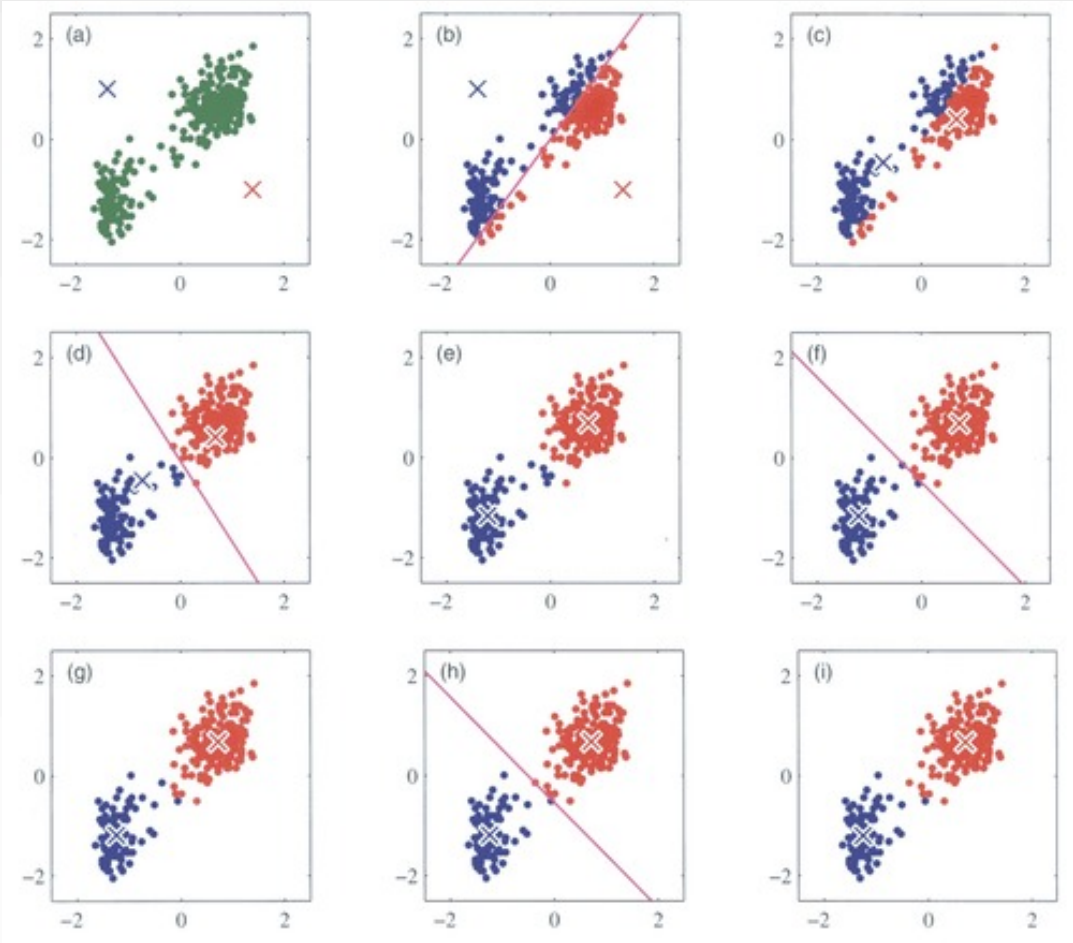
- Born of a need to enable a broader array of interaction patterns for data stored in HDFS beyond MapReduce
- New services:
  - HOYA – HBase on YARN
  - Tez – DAG framework
  - Spark – machine learning (MLib)
  - Giraph – graph processing



Canonical analysis algorithm: k-means

- Start with k cluster centers (chosen randomly or to some recipe)
- Assign each data point to its nearest cluster center
- Reevaluate the cluster centers as the “average” of the data
- Repeat until cluster centers no longer change or some other stopping criterion is met:

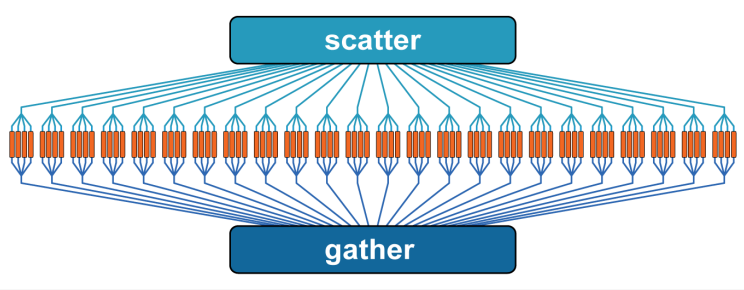
$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$



Parallelize k-means

Analysis:

- Large amounts of data that do not need to be sent around processors
- Minimum processor intercommunication
- Data set needs to be read for each iteration but each point only needs to be read by one processor



Solution:

- Map:
  - Divide data amongst processors
  - Each processor reads previous iteration's cluster centers and assigns its data to the clusters
  - Each processor then calculates new centers for its data
- Reduce:
  - True cluster centers for this iteration are weighted average of new centers from each processor

Stream k-means

Analysis:

- Data are too large to store on available resources or hold in memory
- Data are not persistent so no later processing possible
- Rough-and-ready results required for data exploration
- Time-dependent results to check convergence, data quality

Solution:

Make initial guesses for the centers $w_1, w_2, ..., w_t$

Set the counts $n_1, n_2, ..., n_t$ to zero

Loop until interrupted:

Acquire the next example, x

If w_i is closest to x:

Increment n_i

Replace w_i by $w_i + (1/n_i)*(x - w_i)$

Stochasticize k-means

Analysis:

- k-means is prone to local minima and sensitive to initial clusters
- Normally repeat several times
- Stochastic algorithm can reach (global) minimum quicker:
  - (Nominally) works with subset of the data
  - Relative position of clusters found very quickly
  - Terminal convergence slowed down by stochastic noise implied by random choice of points
  - Great learning algorithm but hopeless optimization algorithm

Solution:

- The right choice of learning rate (replace scalar with inverse Hessian of loss) gives much better convergence
- This is just the online version