MapReduce for Data Intensive Scientific Analyses

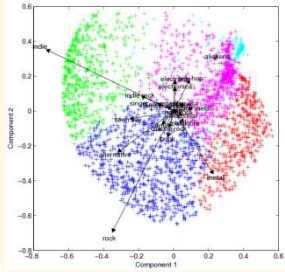
Jaliya Ekanayake
Shrideep Pallickara
Geoffrey Fox
Department of Computer Science
Indiana University Bloomington, IN, 47405

Presentation Outline

- Introduction
- MapReduce and the Current Implementations
- Current Limitations
- Our Solution
- Evaluation and the Results
- Future Work and Conclusion

Data/Compute Intensive Applications

- Computation and data intensive applications are increasingly prevalent
- The data volumes are already in peta-scale
  - High Energy Physics (HEP)
    - Large Hadron Collider (LHC) - Tens of Petabytes of data annually
  - Astronomy
    - Large Synoptic Survey Telescope -Nightly rate of 20 Terabytes
  - Information Retrieval
    - Google, MSN, Yahoo, Wal-Mart etc..
- Many compute intensive applications and domains
  - HEP, Astronomy, chemistry, biology, and seismology etc..
  - Clustering
    - Kmeans, Deterministic Annealing, Pair-wise clustering etc…
  - Multi Dimensional Scaling (MDS) for visualizing high dimensional data

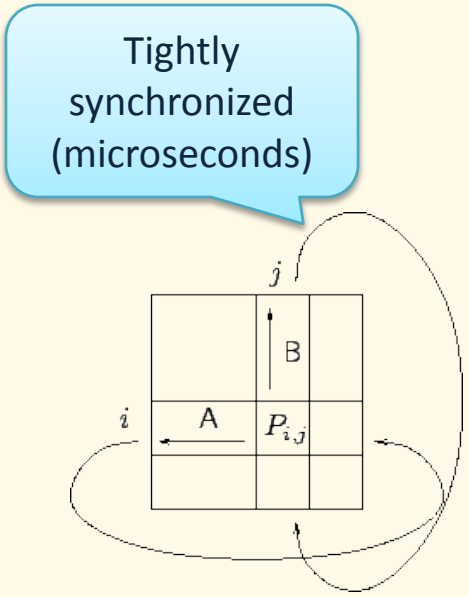


Composable Applications

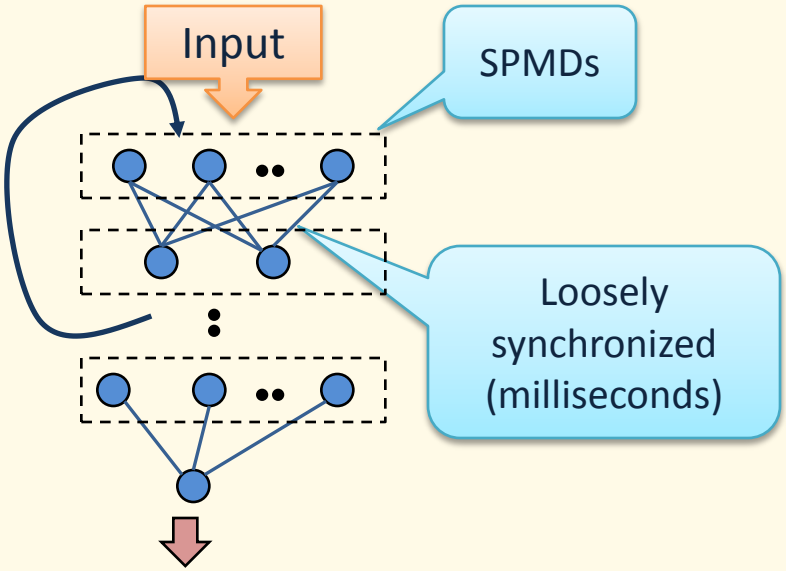
- **How do we support these large scale applications**?
  - Efficient parallel/concurrent algorithms and implementation techniques

- **Some key observations**
  - Most of these applications are:
    - A Single Program Multiple Data (SPMD) program
    - or a collection of SPMDs
  - Exhibits the ***composable*** property
    - Processing can be split into small sub computations
    - The partial-results of these computations are merged after some post-processing
    - Loosely synchronized (Can withstand communication latencies typically experienced over wide area networks)
    - Distinct from the closely coupled parallel applications and totally decoupled applications
  - With large volumes of data and higher computation requirements, even closely coupled parallel applications can withstand higher communication latencies**?**

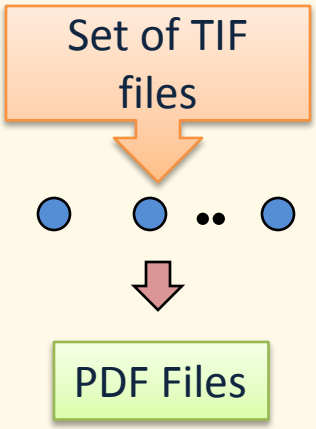
The Composable Class of Applications



Cannon's Algorithm for matrix multiplication – tightly coupled application



Composable application



Totally decoupled application

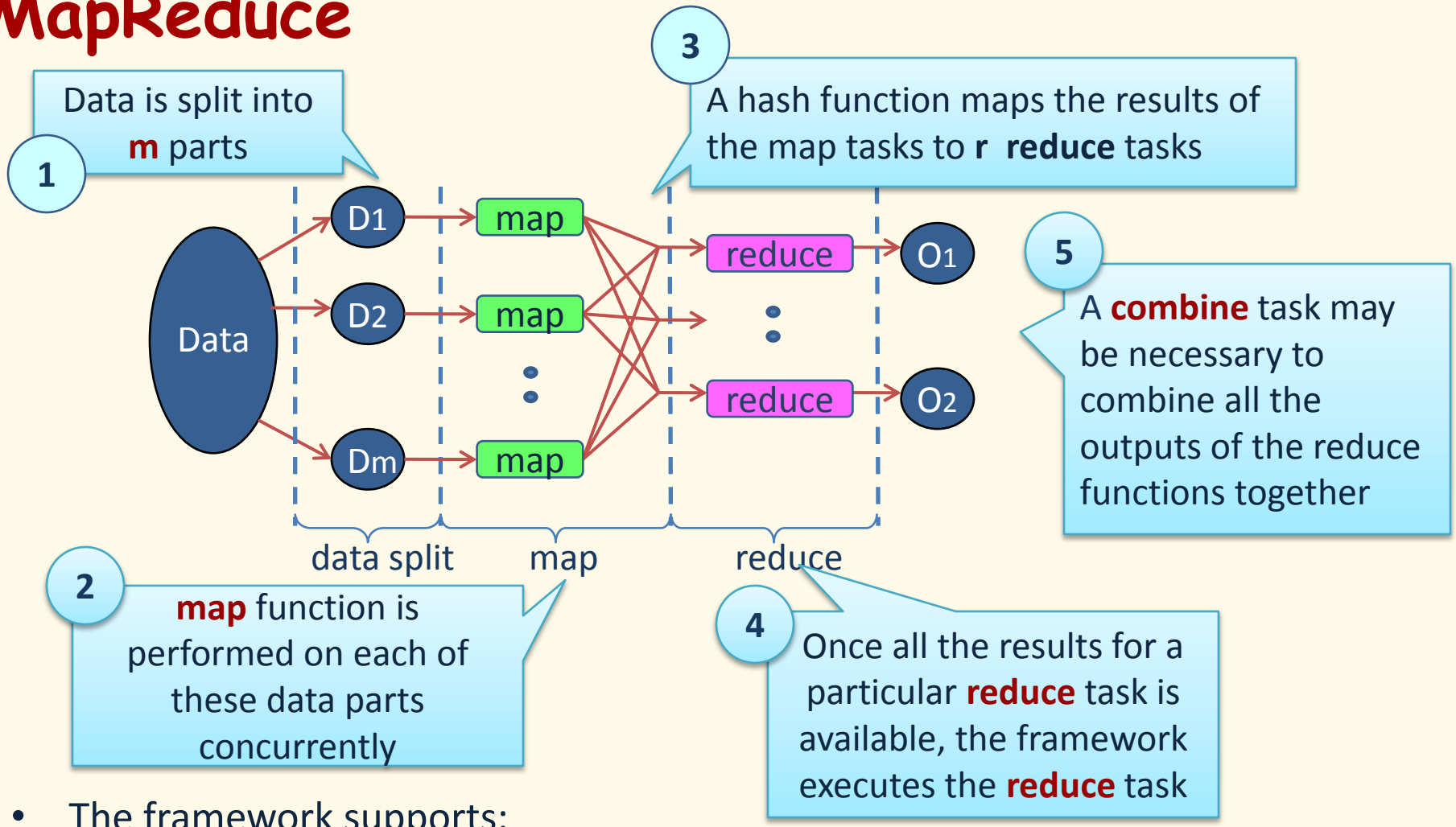
Composable class can be implemented in high-level programming models such as MapReduce and Dryad

MapReduce

"MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key."

MapReduce: Simplified Data Processing on Large Clusters
Jeffrey Dean and Sanjay Ghemawat

MapReduce



- The framework supports:
  - Splitting of data
  - Passing the output of map functions to reduce functions
  - Sorting the inputs to the reduce function based on the intermediate keys
  - Quality of services

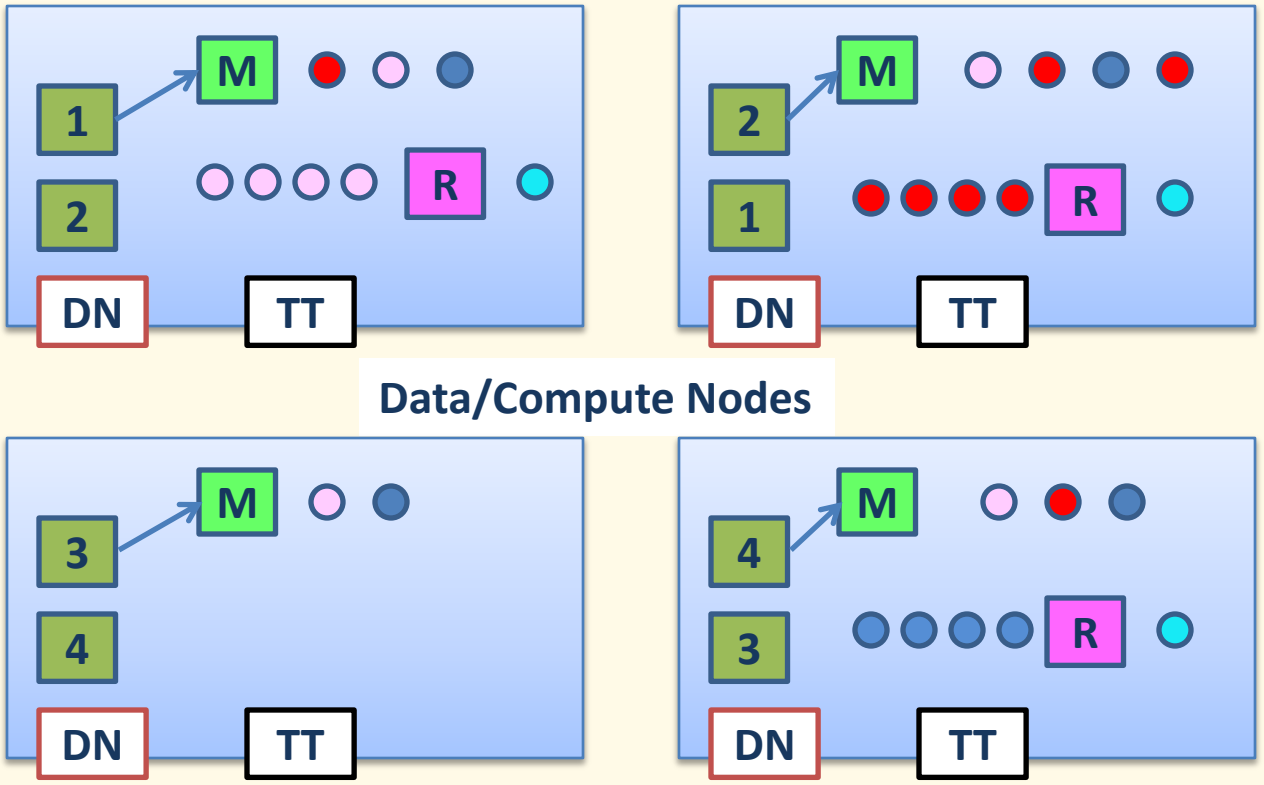
Hadoop Example: Word Count

E.g. Word Count

```
map(String key, String value):
// key: document name
// value: document contents
```

```
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
```

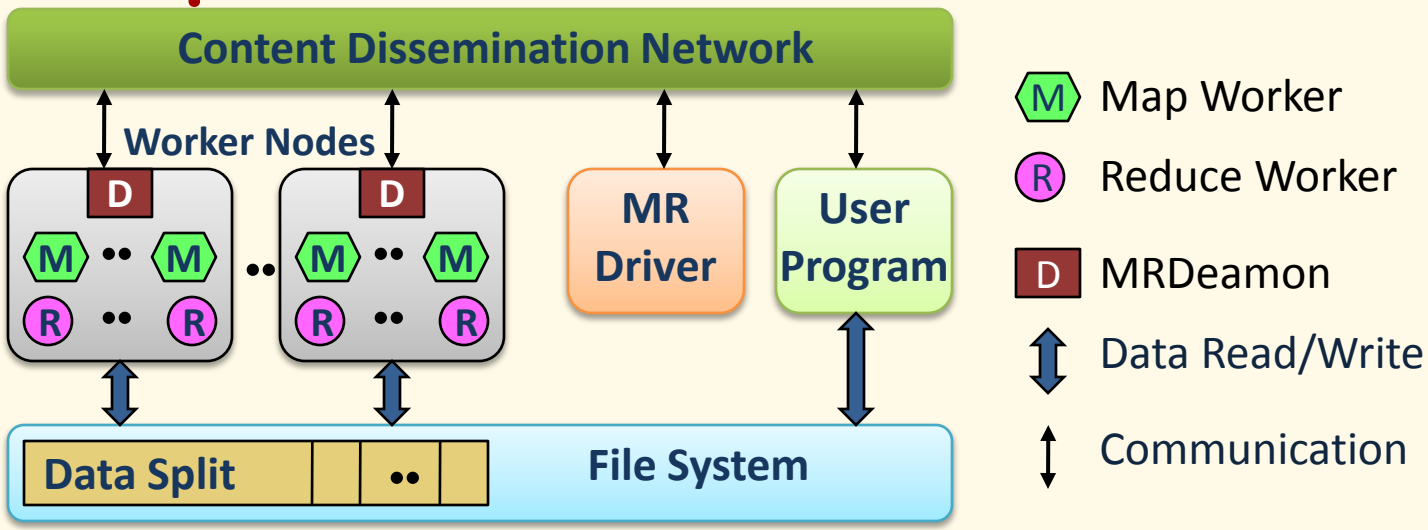
- Task Trackers Execute Map tasks
- Output of map tasks are written to local files
- Retrieve map results via HTTP
- Sort the outputs
- Execute reduce tasks



Current Limitations

- The MapReduce programming model could be applied to most composable applications but;
  - Current MapReduce model and the runtimes focus on “Single Step” MapReduce computations only
  - Intermediate data is stored and accessed via file systems
  - Inefficient for the iterative computations to which the MapReduce technique could be applied
- No performance model to compare with other high-level or low-level parallel runtimes

CGL-MapReduce

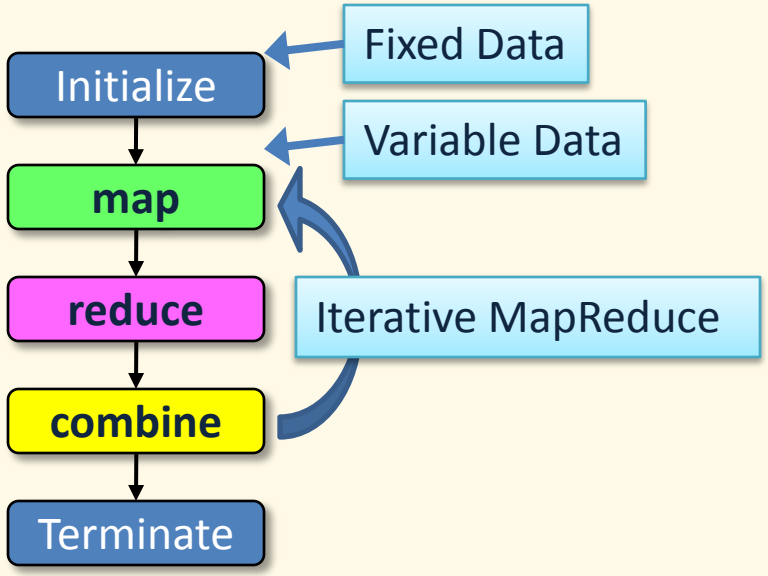


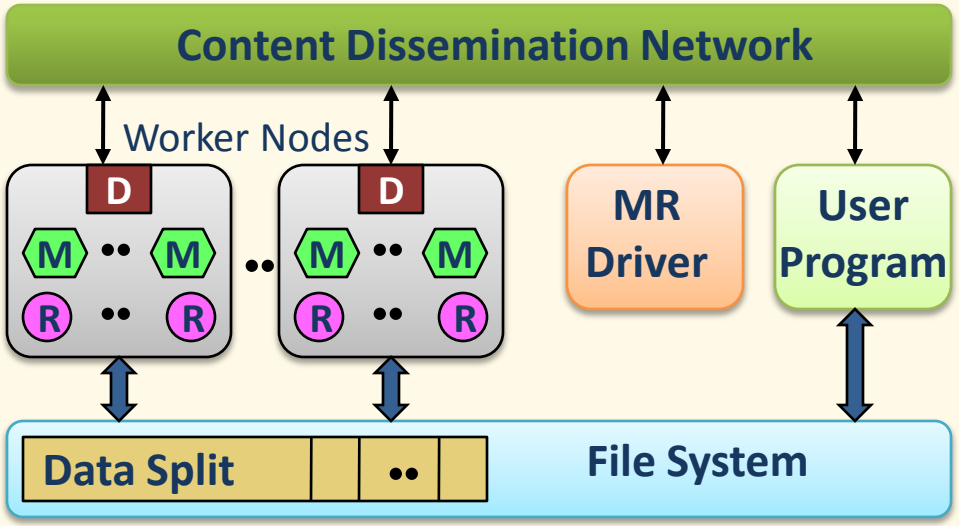
Architecture of CGL-MapReduce

- A streaming based MapReduce runtime implemented in Java
- All the communications(control/intermediate results) are routed via a content dissemination network
- Intermediate results are directly transferred from the map tasks to the reduce tasks – **eliminates local files**
- MRDriver
  - Maintains the state of the system
  - Controls the execution of map/reduce tasks
- User Program is the **composer** of MapReduce computations
- Support both **single step** and **iterative** MapReduce computations

CGL-MapReduce – The Flow of Execution

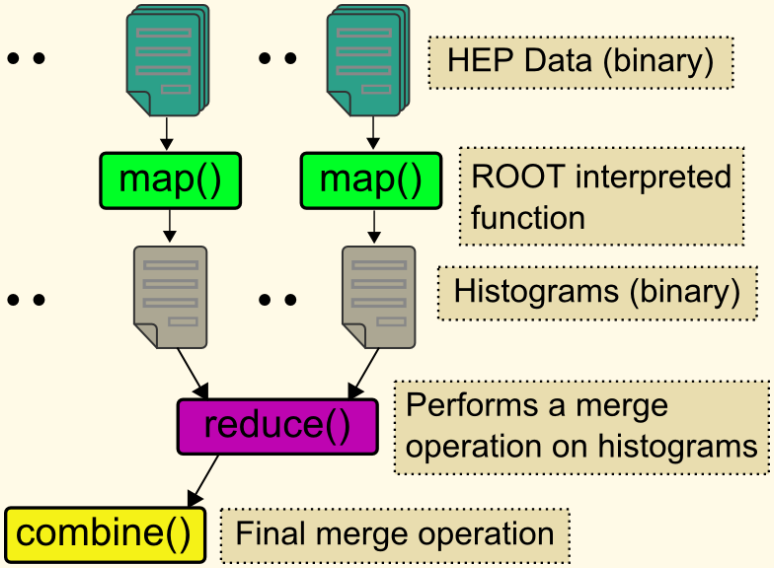
1. **Initialization**
   - Start the map/reduce workers
   - Configure both map/reduce tasks (for configurations/fixed data)
2. **Map**
   - Execute map tasks passing <key, value> pairs
3. **Reduce**
   - Execute reduce tasks passing <key, List<values>>
4. **Combine**
   - Combine the outputs of all the reduce tasks
5. **Termination**
   - Terminate the map/reduce workers



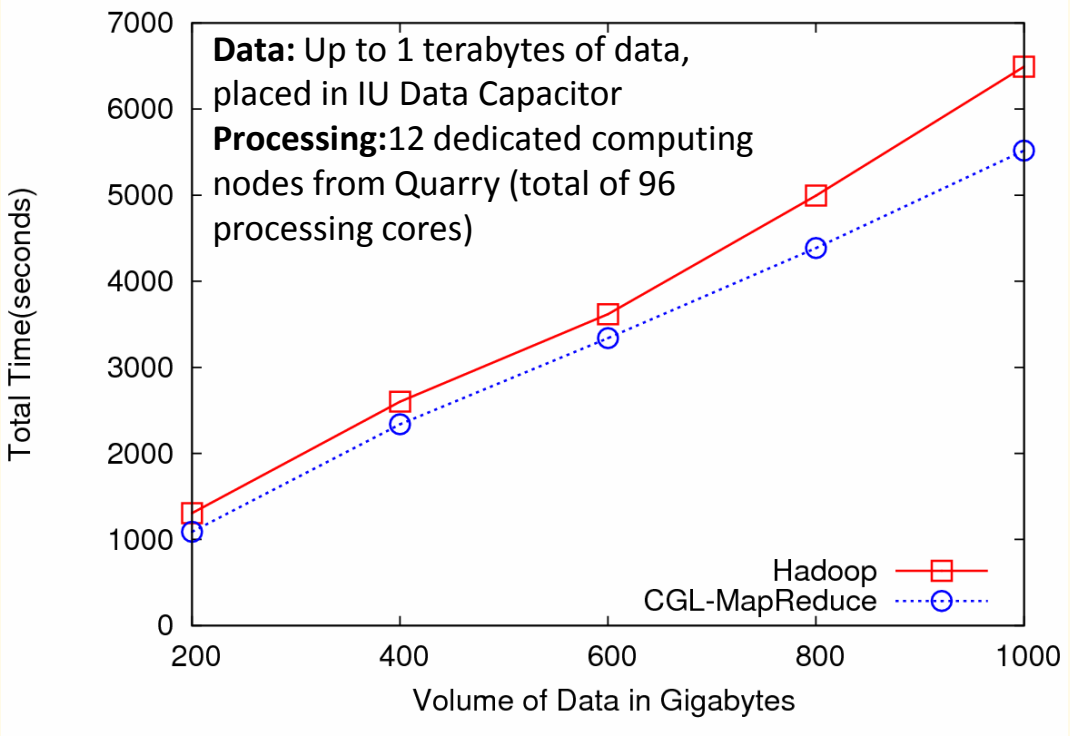


CGL-MapReduce, the flow of execution

HEP Data Analysis



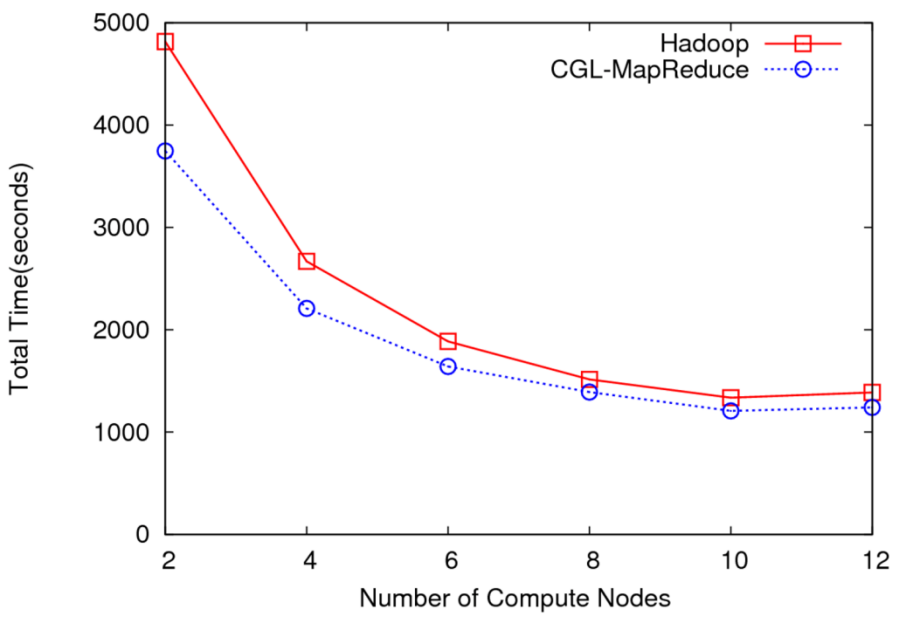
MapReduce for HEP data analysis



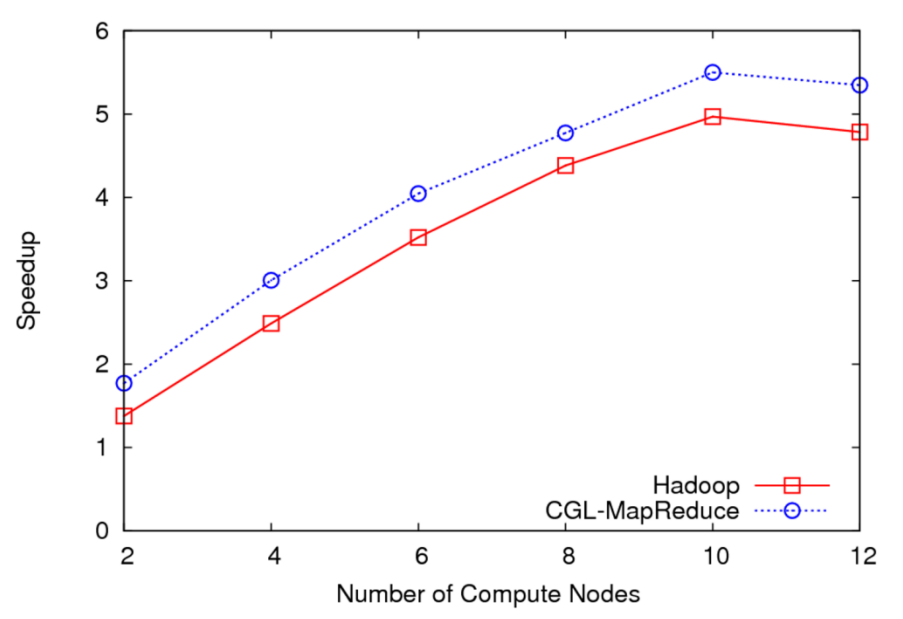
HEP data analysis, execution time vs. the volume of data (fixed compute resources)

- Hadoop and CGL-MapReduce both show similar performance
- The amount of data accessed in each analysis is extremely large
- Performance is limited by the I/O bandwidth
- The overhead induced by the MapReduce implementations has negligible effect on the overall computation

HEP Data Analysis Scalability and Speedup



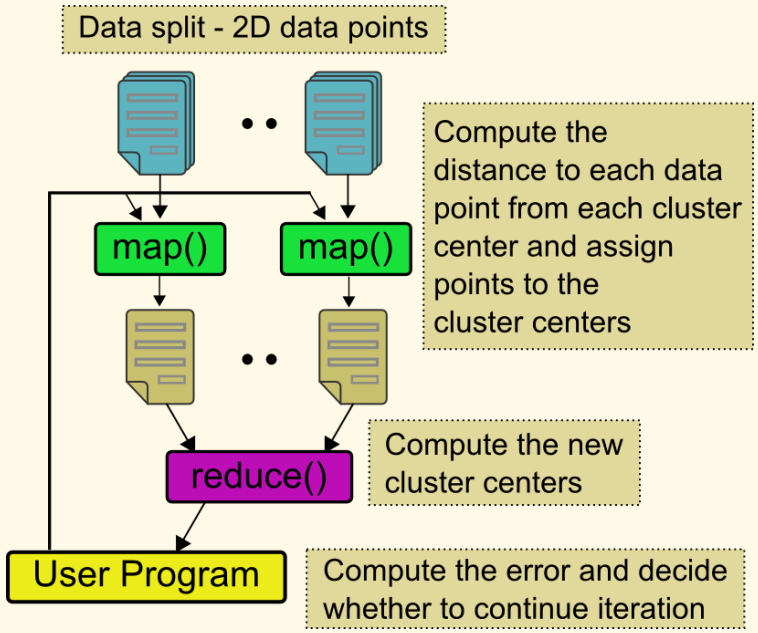
Execution time vs. the number of compute nodes (fixed data)



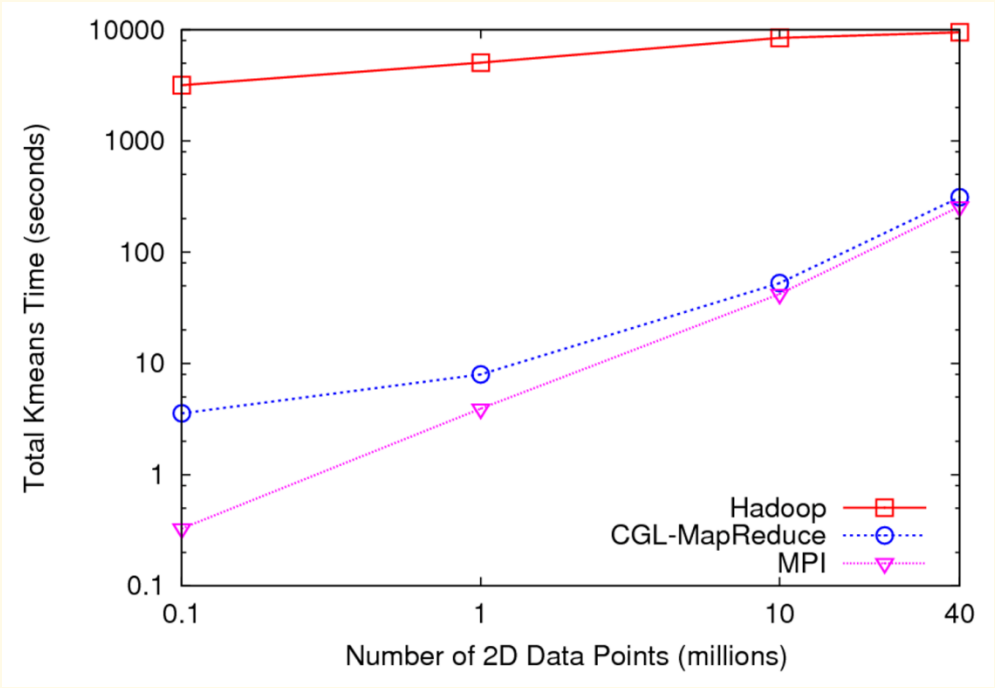
Speedup for 100GB of HEP data

- 100 GB of data
- One core of each node is used (Performance is limited by the I/O bandwidth)
- Speedup = MapReduce Time / Sequential Time
- Speed gain diminish after a certain number of parallel processing units (after around 10 units)

Kmeans Clustering



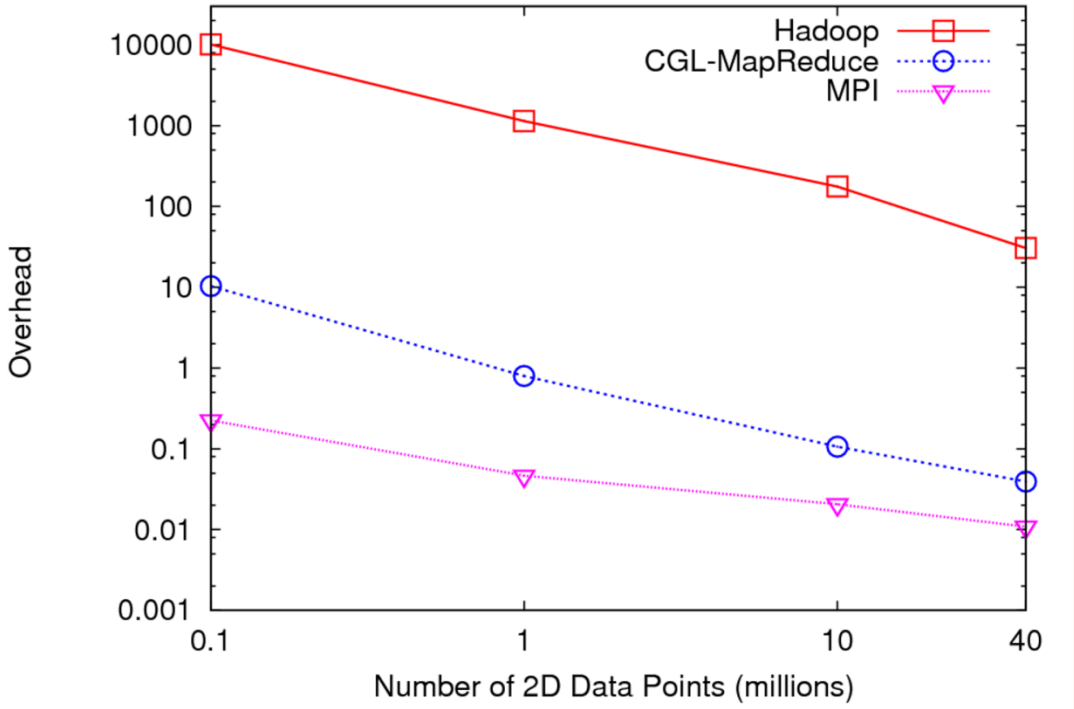
MapReduce for Kmeans Clustering



Kmeans Clustering, execution time vs. the number of 2D data points (Both axes are in log scale)

- All three implementations perform the same Kmeans clustering algorithm
- Each test is performed using 5 compute nodes (Total of 40 processor cores)
- CGL-MapReduce shows a performance close to the MPI implementation
- Hadoop's high execution time is due to:
  - Lack of support for iterative MapReduce computation
  - Overhead associated with the file system based communication

Overheads of Different Runtimes



Overhead f(P)= [P T(P)–T(1)]/T(1)

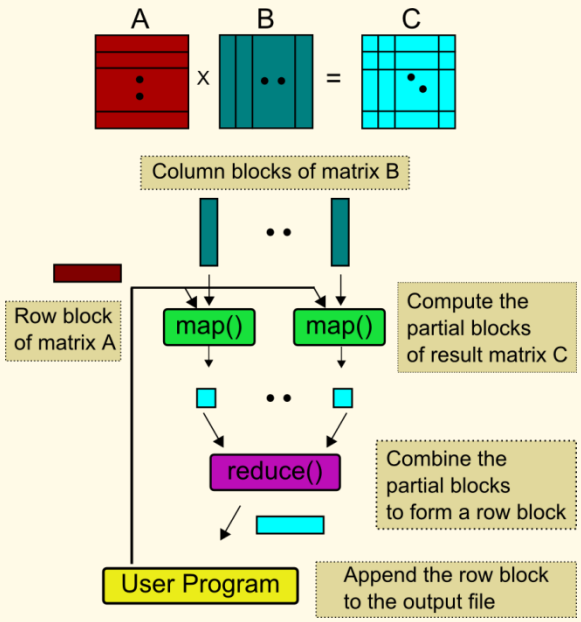
P *- The number of hardware processing units*

T(P) *– The time as a function of P*

T(1) *– The time when a sequential program is used (P=1)*

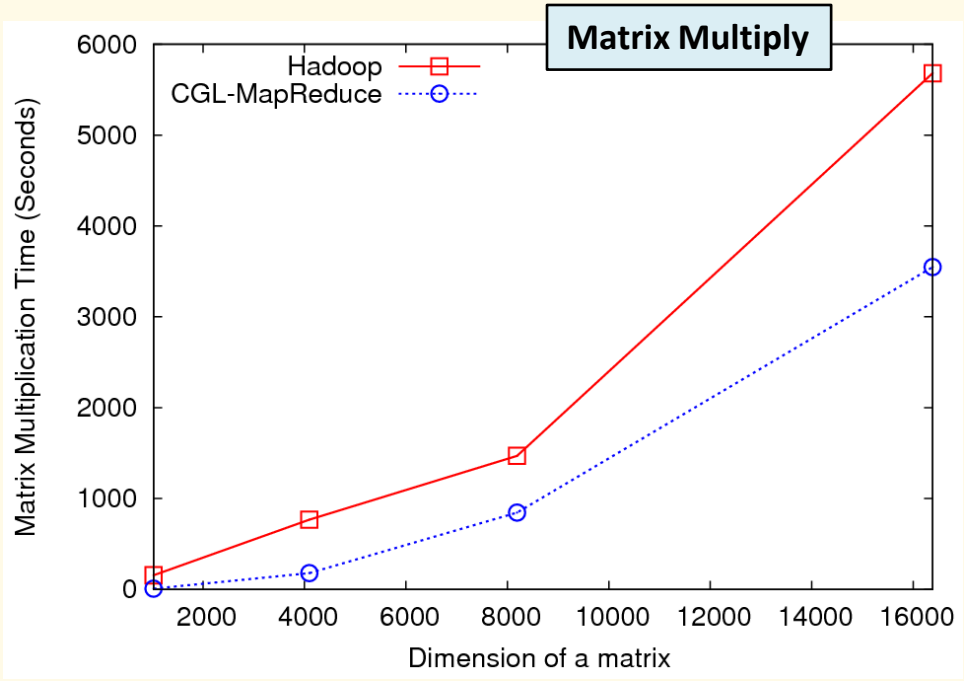
- Overhead diminishes with the amount of computation
- Loosely synchronous MapReduce (CGL-MapReduce) also shows overheads close to MPI for sufficiently large problems
- Hadoop's higher overheads may limit its use for these types(iterative MapReduce) of computations

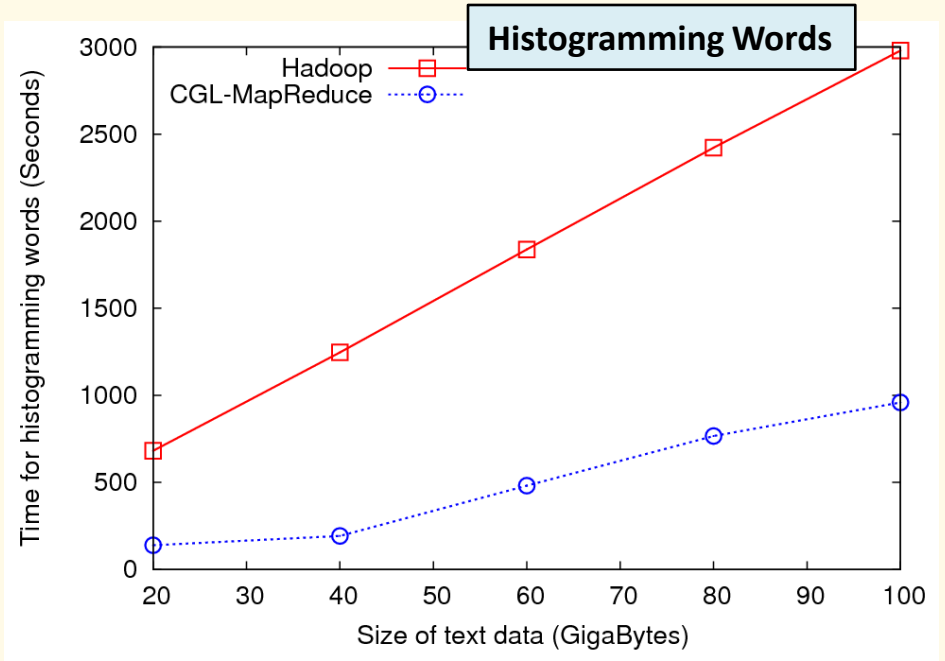
More Applications



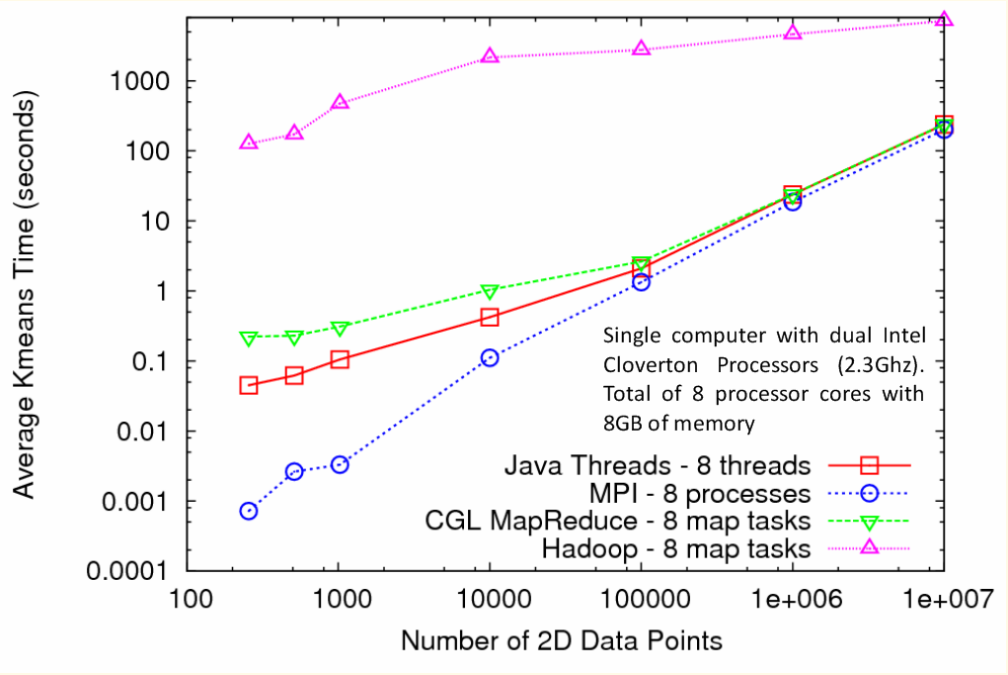
MapReduce for Matrix Multiplication

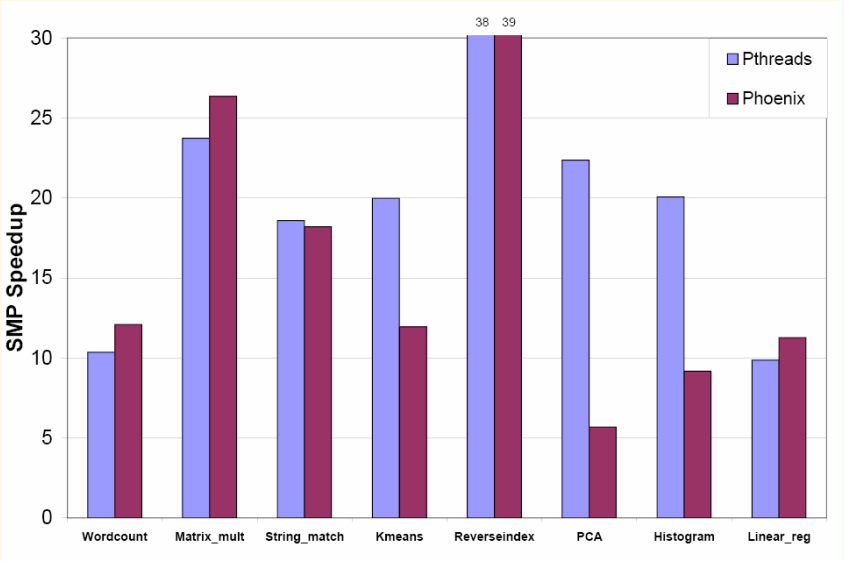
- Matrix multiplication -> iterative algorithm
- Histogramming words -> simple MapReduce application
- Streaming approach provide better performance in both applications





Multicore and the Runtimes

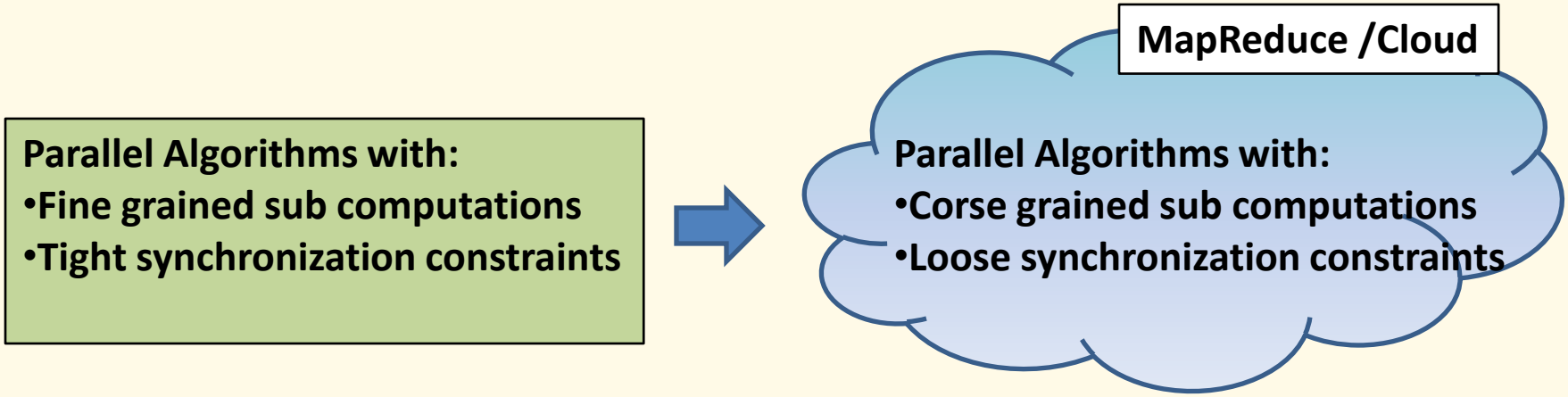




- The papers [1] and [2] evaluate the performance of MapReduce using Multicore computers
- Our results show the converging results for different runtimes
- The right hand side graph could be a snapshot of this convergence path
- Easiness to program could be a consideration
- Still, threads are faster in shared memory systems

[1] **Evaluating MapReduce for Multi-core and Multiprocessor Systems**. By C. Ranger et al.
[2] **Map-Reduce for Machine Learning on Multicore** by C. Chu et al.

Conclusions



- Given sufficiently large problems, all runtimes converge in performance
- Streaming-based map reduce implementations provide faster performance necessary for most *composable* applications
- Support for iterative MapReduce computations expands the usability of MapReduce runtimes

Future Work

- Research on different fault tolerance strategies for CGL-MapReduce and come up with a set of architectural recommendations
- Integration of a distributed file system such as HDFS
- Applicability for cloud computing environments

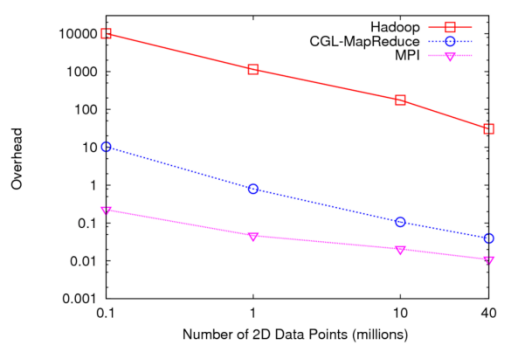
Questions?

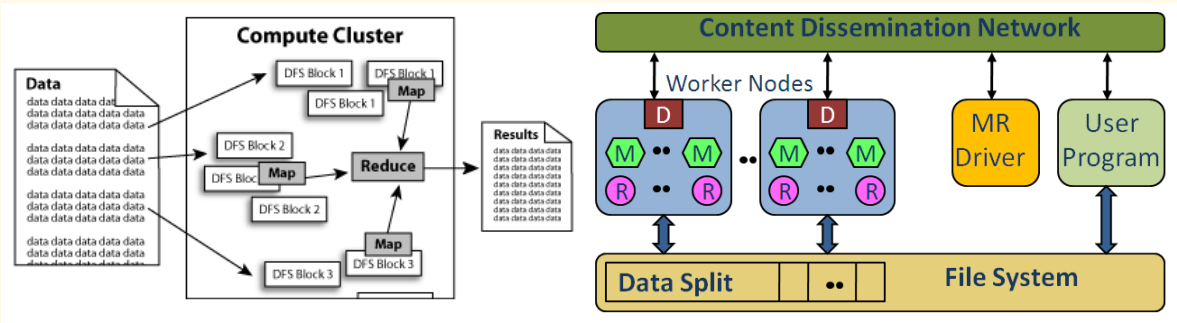
Thank You!

Links

- Hadoop vs. CGL-MapReduce
  - Is it fair to compare Hadoop with CGL-MapReduce ?
- DRYAD
- Fault Tolerance
- Rootlet Architecture
- Nimbus vs. Eucalyptus

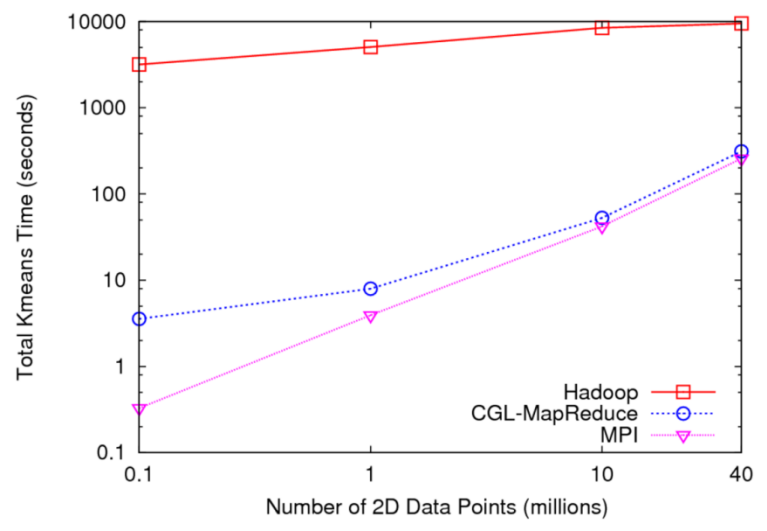
Hadoop vs. CGL-MapReduce

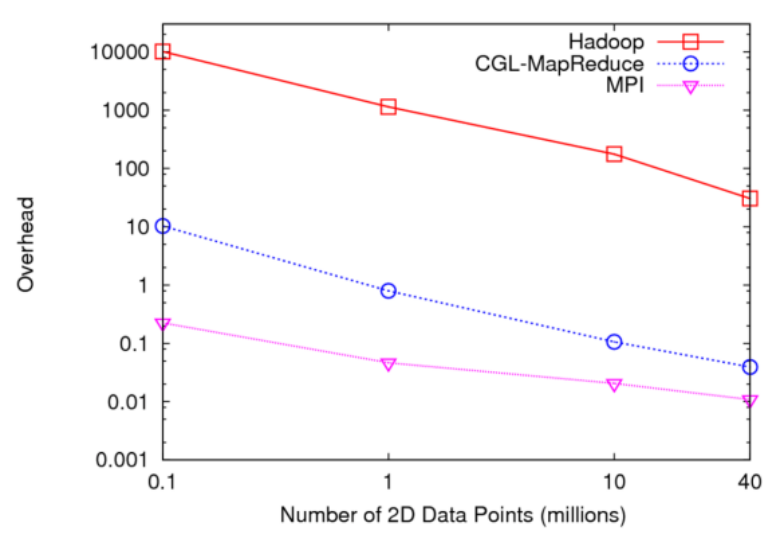




Feature	Hadoop	CGL-MapReduce
Implementation Language	Java	Java
Other Language Support	Uses Hadoop Streaming (Text Data only)	Requires a Java wrapper classes
Distributed File System	HDFS	Currently assumes a shared file system between nodes
Accessing binary data from other languages	Currently only a Java interface is available	Shared file system enables this functionality
Fault Tolerance	Support failures of nodes	Currently does not support fault tolerance
Iterative Computations	Not supported	Supports Iterative MapReduce
Daemon Initialization	Requires ssh public key access	Requires ssh public key access

Is it fair to compare Hadoop with CGL-MapReduce ?



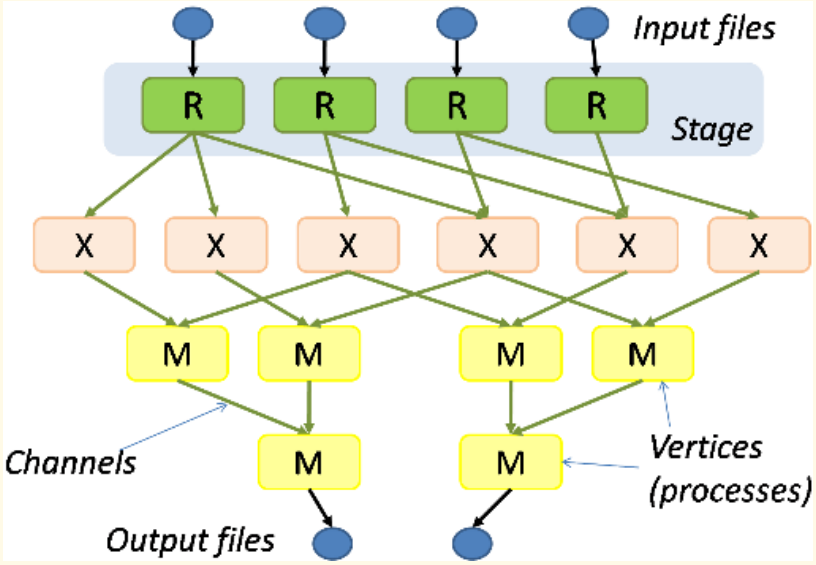


- Hadoop access data via a distributed file system
- Hadoop stores all the intermediate results in this file system to ensure fault tolerance
- Is this is the optimum strategy?
- Can we use Hadoop for only “single pass” MapReduce computations?
- Writing the intermediate results at the Reduce task will be a better strategy?
  - Considerable reduction in data from map -> reduce
  - Possibility of using duplicate reduce tasks

Fault Tolerance

- Hadoop/Google Fault Tolerance
  - Data (input, output, and intermediate) are stored in HDFS
  - HDFS uses replications
  - Task tracker is a single point of failure (Checkpointing)
  - Re-executing failed map/reduce tasks
- CGL-MapReduce
  - Integration of HDFS or similar parallel file system for input/output data
  - MRDriver is a single point of failure (Checkpointing)
  - Re-executing failed map tasks (Considerable reduction in data from map -> reduce)
  - Redundant reduce tasks
- Amazon model S3, EC2 and SQS
  - Reliable Queues

DRYAD



- The computation is structured as a directed graph
- A Dryad job is a *graph generator* which can synthesize any directed acyclic graph
- These graphs can even change during execution, in response to important events in the computation
- Dryad handles job creation and management, resource management, job monitoring and visualization, fault tolerance, re-execution, scheduling, and accounting
- **How to support iterative computations?**