CS 525
Advanced Distributed Systems
Spring 2015

Indranil Gupta (Indy)

Lecture 3

Cloud Computing (Contd.)

January 27, 2015

What is MapReduce?

- Terms are borrowed from Functional Language (e.g., Lisp)

Sum of squares:

- (map square ‘(1 2 3 4))
  - Output: (1 4 9 16)

  [processes each record sequentially and independently]

- (reduce + ‘(1 4 9 16))
  - (+ 16 (+ 9 (+ 4 1)))
  - Output: 30

  [processes set of all records in batches]

- Let’s consider a sample application: Wordcount
  - You are given a huge dataset (e.g., Wikipedia dump or all of Shakespeare’s works) and asked to list the count for each of the words in each of the documents therein

Map

- Process individual records to generate intermediate key/value pairs.

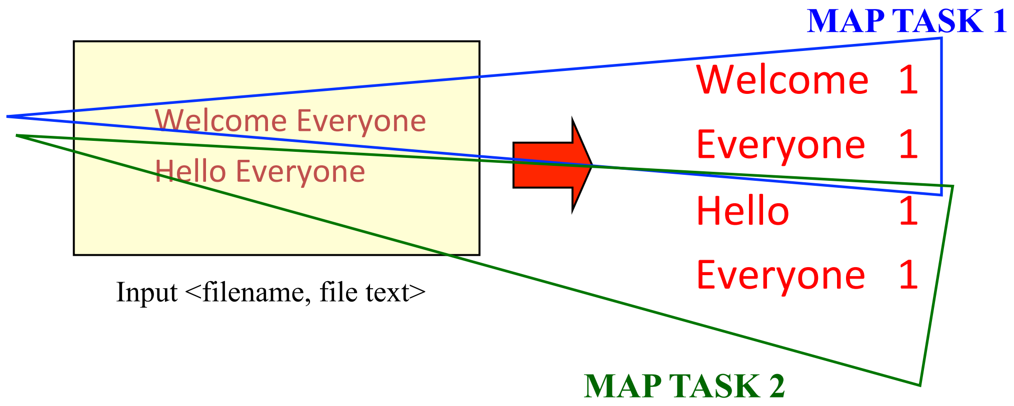
Welcome Everyone
Hello Everyone

Input <filename, file text>

Key	Value
Welcome	1
Everyone	1
Hello	1
Everyone	1

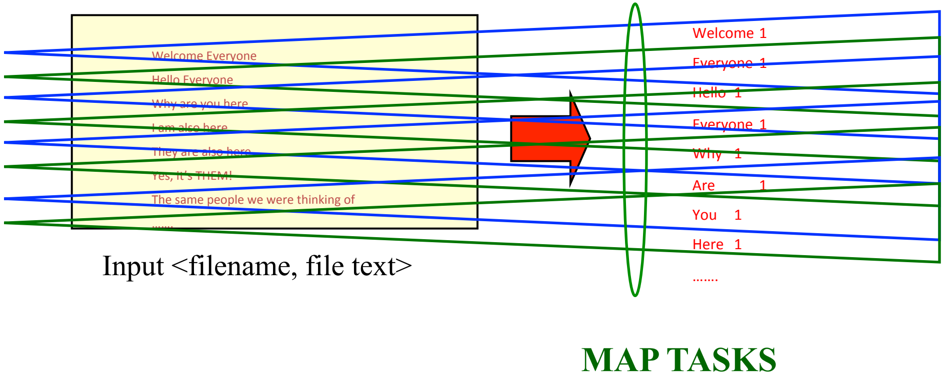
Map

- Parallelly Process individual records to generate intermediate key/value pairs.



Map

- Parallelly Process a large number of individual records to generate intermediate key/value pairs.



Reduce

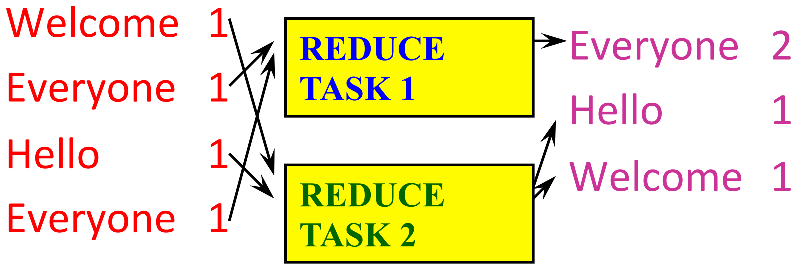
- Reduce processes and merges all intermediate values associated <u>per key</u>

Welcome	1
Everyone	1
Hello	1
Everyone	1

Key	Value
Everyone	2
Hello	1
Welcome	1

Reduce

- Each key assigned to one Reduce
- Parallelly Processes and merges all intermediate values by partitioning keys



- Popular: *Hash partitioning, i.e.,* key is assigned to reduce # = hash(key)%number of reduce servers

Hadoop Code - Map

```
public static class MapClass extends MapReduceBase     implements
Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one =
    new IntWritable(1);
  private Text word = new Text();

  public void map( LongWritable key, Text value,
       OutputCollector<Text, IntWritable> output, Reporter reporter)
    throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
     word.set(itr.nextToken());
     output.collect(word, one);
    }
  }
} // Source: http://developer.yahoo.com/hadoop/tutorial/module4.html#wordcount
```

Hadoop Code - Reduce

```
public static class ReduceClass extends MapReduceBase  implements
Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(
      Text key,
      Iterator<IntWritable> values,
      OutputCollector<Text, IntWritable> output,
      Reporter reporter)
     throws IOException {
      int sum = 0;
      while (values.hasNext()) {
        sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));
  }
} // Source: http://developer.yahoo.com/hadoop/tutorial/module4.html#wordcount
```

Hadoop Code - Driver

```
// Tells Hadoop how to run your Map-Reduce job
public void run (String inputPath, String outputPath)
      throws Exception {
  // The job. WordCount contains MapClass and Reduce.
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName(”mywordcount");
  // The keys are words
  (strings) conf.setOutputKeyClass(Text.class);
  // The values are counts (ints)
  conf.setOutputValueClass(IntWritable.class);
  conf.setMapperClass(MapClass.class);
  conf.setReducerClass(ReduceClass.class);
  FileInputFormat.addInputPath(
      conf, newPath(inputPath));
  FileOutputFormat.setOutputPath(
      conf, new Path(outputPath));
  JobClient.runJob(conf);
} // Source: http://developer.yahoo.com/hadoop/tutorial/module4.html#wordcount
```

Some Applications of MapReduce

Distributed Grep:

- Input: large set of files
- Output: lines that match pattern

- Map – *Emits a line if it matches the supplied pattern*
- Reduce – *Copies the intermediate data to output*

Some Applications of MapReduce (2)

Reverse Web-Link Graph

- Input: Web graph: tuples (a, b) where (page a → page b)
- Output: For each page, list of pages that link *to* it

- Map – *process web log and for each input <source, target>, it outputs <target, source>*
- Reduce - *emits <target, list(source)>*

Some Applications of MapReduce (3)

Count of URL access frequency

- Input: Log of accessed URLs, e.g., from proxy server
- Output: For each URL, % of total accesses for that URL

- Map – *Process web log and outputs <URL, 1>*
- Multiple Reducers - *Emits <URL, URL_count>*

(So far, like Wordcount. But still need %)

- Chain another MapReduce job after above one
- Map – *Processes <URL, URL_count>* and outputs *<1, (<URL, URL_count>)>*
- 1 Reducer – Sums up *URL_count's* to calculate overall_count.

  Emits multiple <URL, URL_count/overall_count>

Some Applications of MapReduce (4)

Map task's output is sorted (e.g., quicksort)

Reduce task's input is sorted (e.g., mergesort)

Sort

- Input: Series of (key, value) pairs
- Output: Sorted <value>s

- Map – *<key, value> → <value, _> (identity)*
- Reducer – *<key, value> → <key, value> (identity)*
- Partitioning function – partition keys across reducers based on ranges (can't use hashing!)
  - Take data distribution into account to balance reducer tasks

Programming MapReduce

Externally: For user

1. Write a Map program (short), write a Reduce program (short)
2. Specify number of Maps and Reduces (parallelism level)
3. Submit job; wait for result
4. Need to know very little about parallel/distributed programming!

Internally: For the Paradigm and Scheduler

1. Parallelize Map
2. Transfer data from Map to Reduce
3. Parallelize Reduce
4. Implement Storage for Map input, Map output, Reduce input, and Reduce output

(Ensure that no Reduce starts before all Maps are finished. That is, ensure the ***barrier*** between the Map phase and Reduce phase)

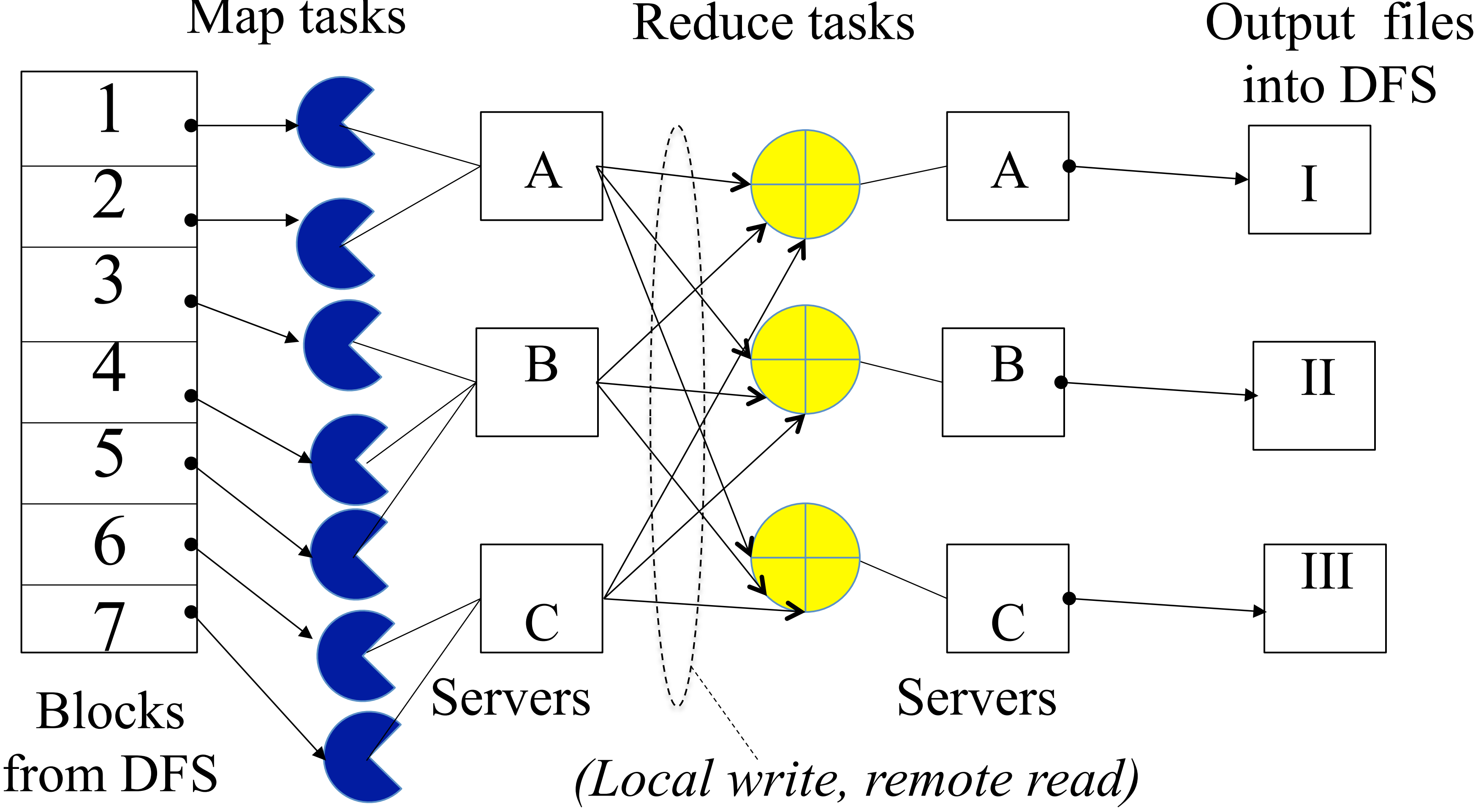
INSIDE MAPREDUCE

For the cloud:

1. Parallelize Map: easy! each map task is independent of the other!
   - All Map output records with same key assigned to same Reduce
2. Transfer data from Map to Reduce:
   - All Map output records with same key assigned to same Reduce task
   - use partitioning function, e.g., hash(key)%number of reducers
3. Parallelize Reduce: easy! each reduce task is independent of the other!
4. Implement Storage for Map input, Map output, Reduce input, and Reduce output
   - Map input: from distributed file system
   - Map output: to local disk (at Map node); uses local file system
   - Reduce input: from (multiple) remote disks; uses local file systems
   - Reduce output: to distributed file system

local file system = Linux FS, etc.

distributed file system = GFS (Google File System), HDFS (Hadoop Distributed File System)

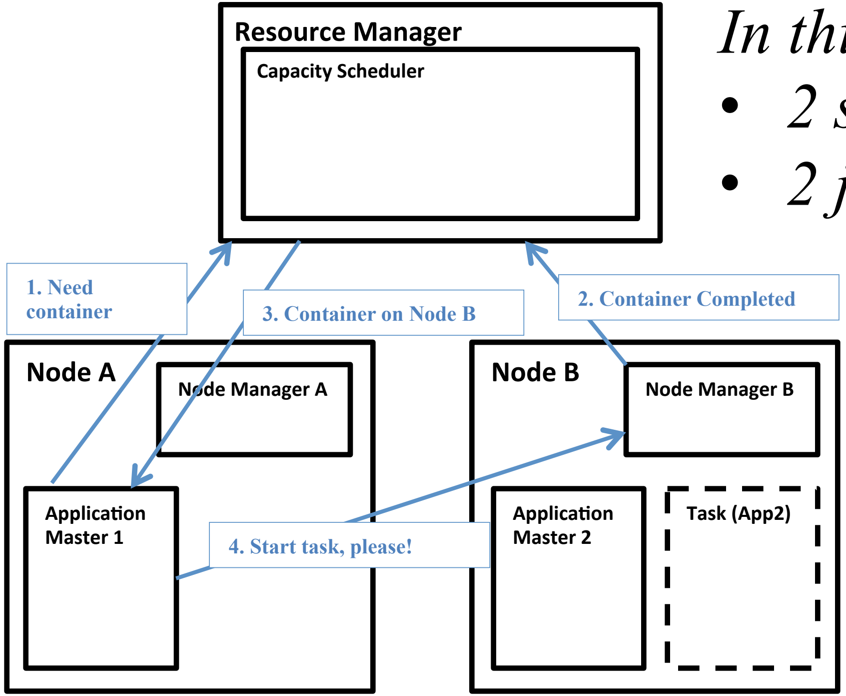


Resource Manager (assigns maps and reduces to servers)

The YARN Scheduler

- Used in Hadoop 2.x +
- YARN = Yet Another Resource Negotiator
- Treats each server as a collection of *containers*
  - Container = fixed CPU + fixed memory
- Has 3 main components
  - Global *Resource Manager (RM)*
    - Scheduling
  - Per-server *Node Manager (NM)*
    - Daemon and server-specific functions
  - Per-application (job) *Application Master (AM)*
    - Container negotiation with RM and NMs
    - Detecting task failures of that job

YARN: How a job gets a container



In this figure

- *2 servers (A, B)*
- *2 jobs (1, 2)*

Fault Tolerance

- Server Failure
  - NM heartbeats to RM
    - If server fails, RM lets all affected AMs know, and AMs take action
  - NM keeps track of each task running at its server
    - If task fails while in-progress, mark the task as idle and restart it
  - AM heartbeats to RM
    - On failure, RM restarts AM, which then syncs up with its running tasks
- RM Failure
  - Use old checkpoints and bring up secondary RM
- Heartbeats also used to piggyback container requests
  - Avoids extra messages

SLOW SERVERS

Slow tasks are called Stragglers

- The slowest task slows the entire job down (why?)
- Due to Bad Disk, Network Bandwidth, CPU, or Memory
- Keep track of "progress" of each task (% done)
- Perform proactive backup (replicated) execution of straggler task: task considered done when first replica complete. Called Speculative Execution.

Locality

- Locality
  - Since cloud has hierarchical topology (e.g., racks)
  - GFS/HDFS stores 3 replicas of each of chunks (e.g., 64 MB in size)
    - Maybe on different racks, e.g., 2 on a rack, 1 on a different rack
  - Mapreduce attempts to schedule a map task on
    - a machine that contains a replica of corresponding input data, or failing that,
    - on the same rack as a machine containing the input, or failing that,
    - Anywhere

Mapreduce: Summary

- Mapreduce uses parallelization + aggregation to schedule applications across clusters
- Need to deal with failure
- Plenty of ongoing research work in scheduling and fault-tolerance for Mapreduce and Hadoop

10 Challenges [Above the Clouds]

(Index: Performance Data-related Scalability Logisitical)

- Availability of Service: Use Multiple Cloud Providers; Use Elasticity; Prevent DDOS
- Data Lock-In: Standardize APIs; Enable Surge Computing
- Data Confidentiality and Auditability: Deploy Encryption, VLANs, Firewalls, Geographical Data Storage
- Data Transfer Bottlenecks: Data Backup/Archival; Higher BW Switches; New Cloud Topologies; FedExing Disks
- Performance Unpredictability: QoS; Improved VM Support; Flash Memory; Schedule VMs
- Scalable Storage: Invent Scalable Store
- Bugs in Large Distributed Systems: Invent Debuggers; Real-time debugging; predictable pre-run-time debugging
- Scaling Quickly: Invent Good Auto-Scalers; Snapshots for Conservation
- Reputation Fate Sharing
- Software Licensing: Pay-for-use licenses; Bulk use sales

A more Bottom-Up View of Open Research Directions

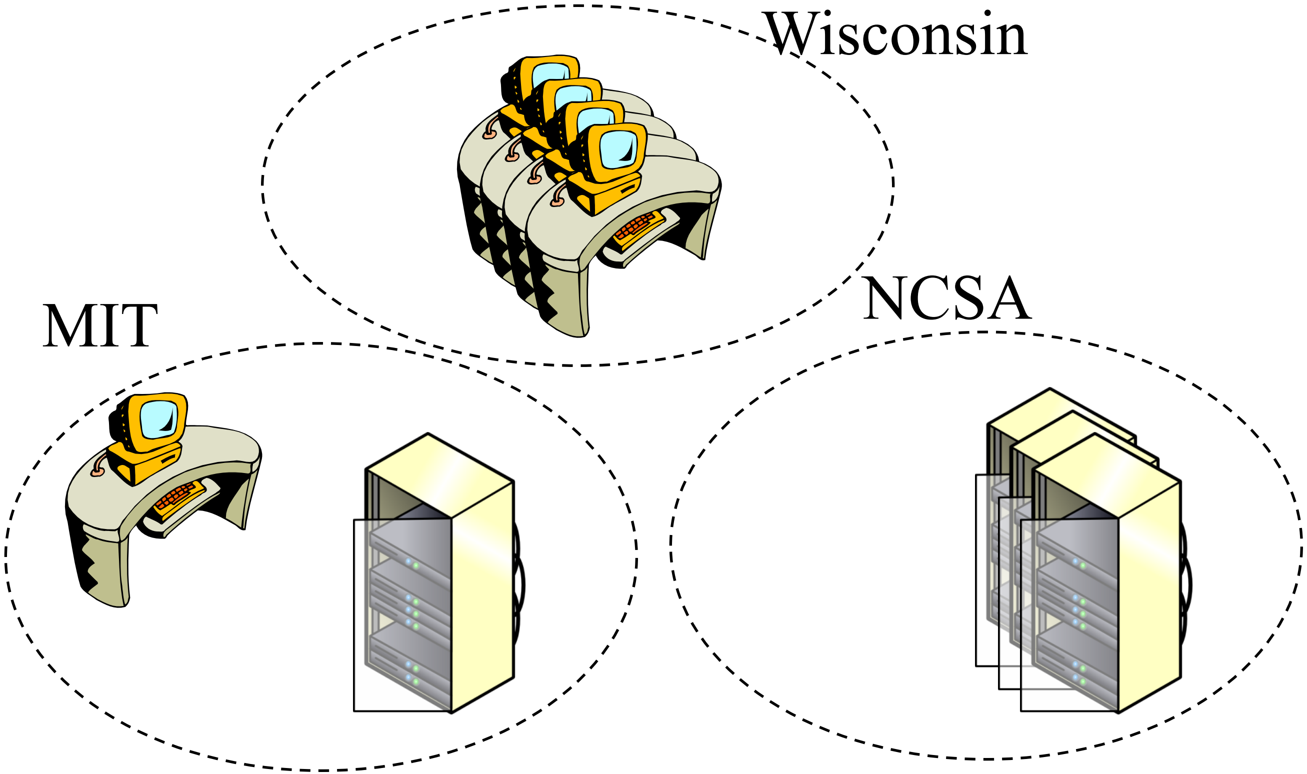
Myriad interesting problems that acknowledge the characteristics that make today's cloud computing unique: massive scale + on-demand + data-intensive + new programmability + and infrastructure- and application-specific details.

- Monitoring: of systems&applications; single site and multi-site
- Storage: massive scale; global storage; for specific apps or classes
- Failures: what is their effect, what is their frequency, how do we achieve fault-tolerance?
- Scheduling: Moving tasks to data, dealing with federation
- Communication bottleneck: within applications, within a site
- Locality: within clouds, or across them
- Cloud Topologies: non-hierarchical, other hierarchical
- Security: of data, of users, of applications, confidentiality, integrity
- Availability of Data
- Seamless Scalability: of applications, of clouds, of data, of everything
- Geo-distributed clouds: Inter-cloud/multi-cloud computations
- Second Generation of Other Programming Models? Beyond MapReduce! Storm, GraphLab, Hama
- Pricing Models, SLAs, Fairness
- Green cloud computing
- Stream processing

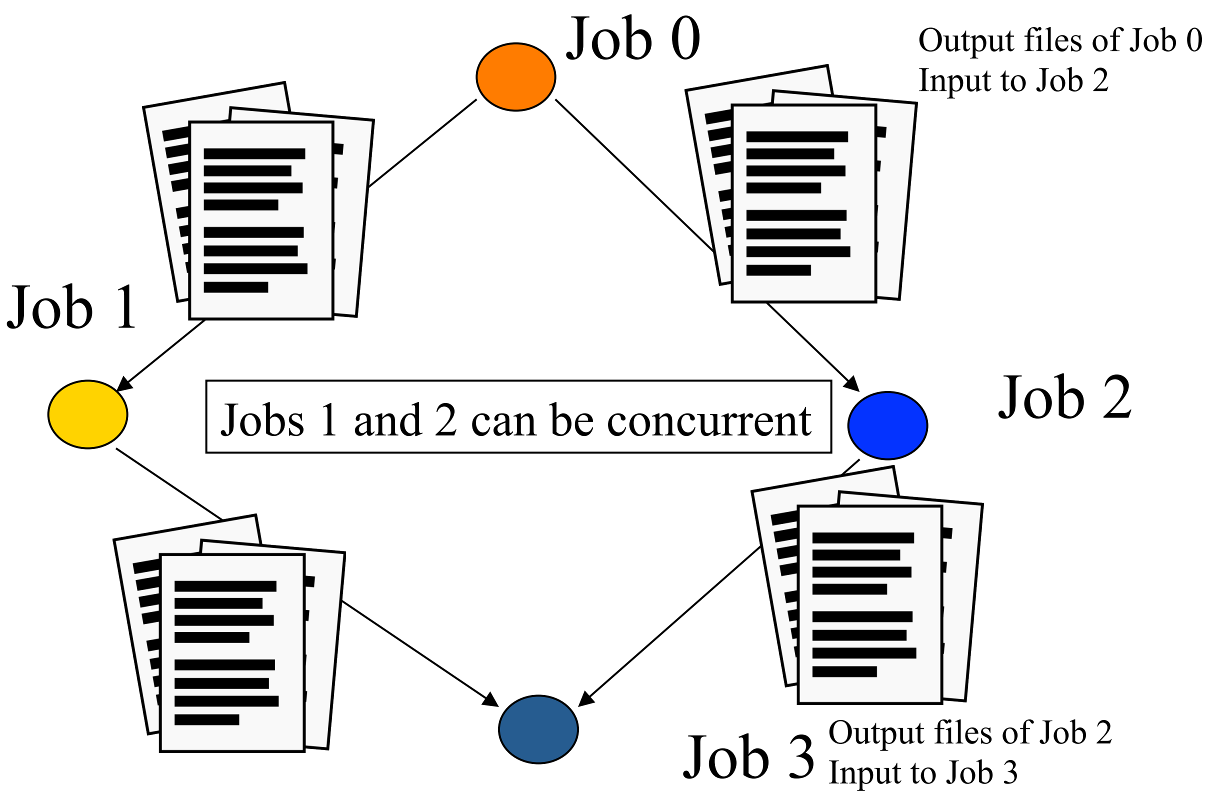
Example: Rapid Atmospheric Modeling System, ColoState U

- Hurricane Georges, 17 days in Sept 1998
  - "RAMS modeled the mesoscale convective complex that dropped so much rain, in good agreement with recorded data"
  - Used 5 km spacing instead of the usual 10 km
  - Ran on 256+ processors
- Computation-intenstive computing (or HPC = High Performance Computing)
- *Can one run such a program without access to a supercomputer?*

Distributed Computing Resources



An Application Coded by a Physicist/Biologist/Meterologist



An Application Coded by a Physicist/Biologist/Meterologist

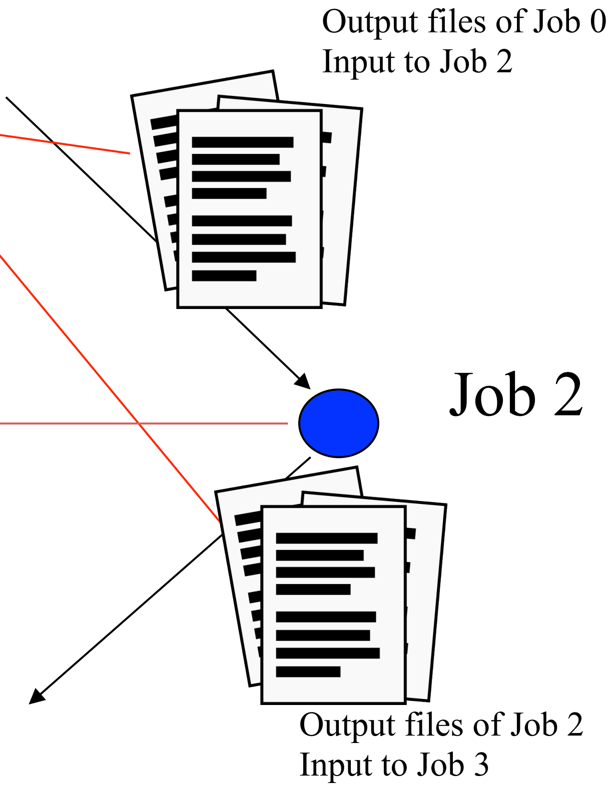
Several GBs

May take several hours/days

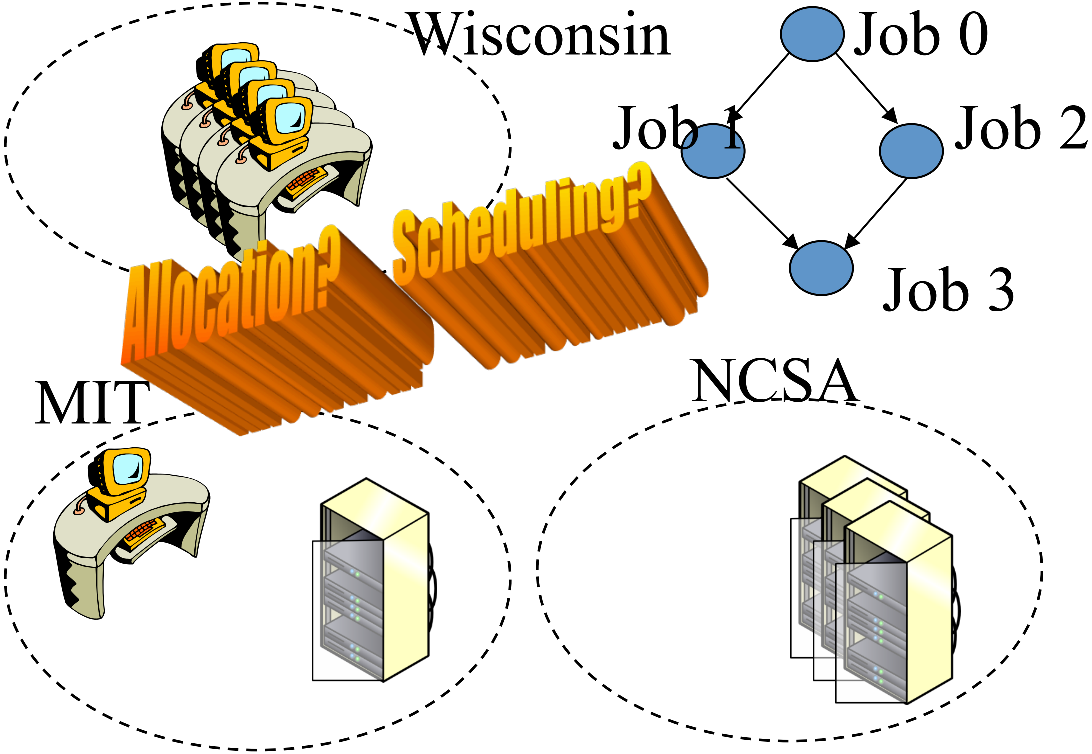
4 stages of a job

- Init
- Stage in
- Execute
- Stage out
- Publish

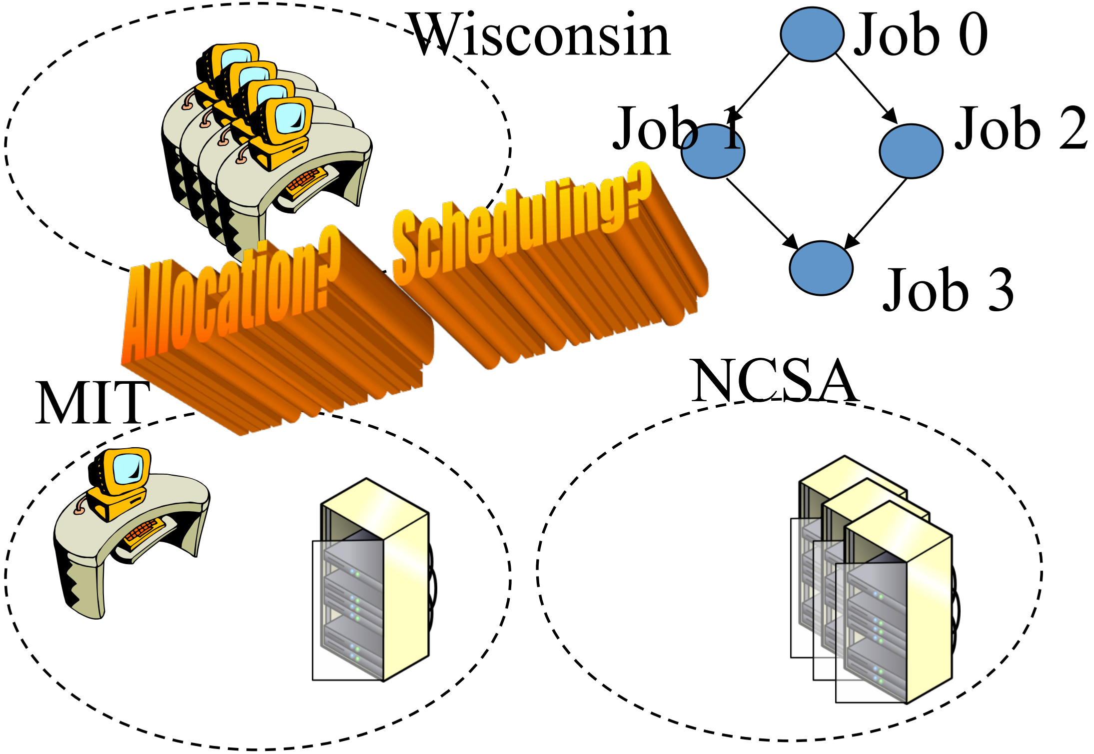
Computation Intensive,
so Massively Parallel



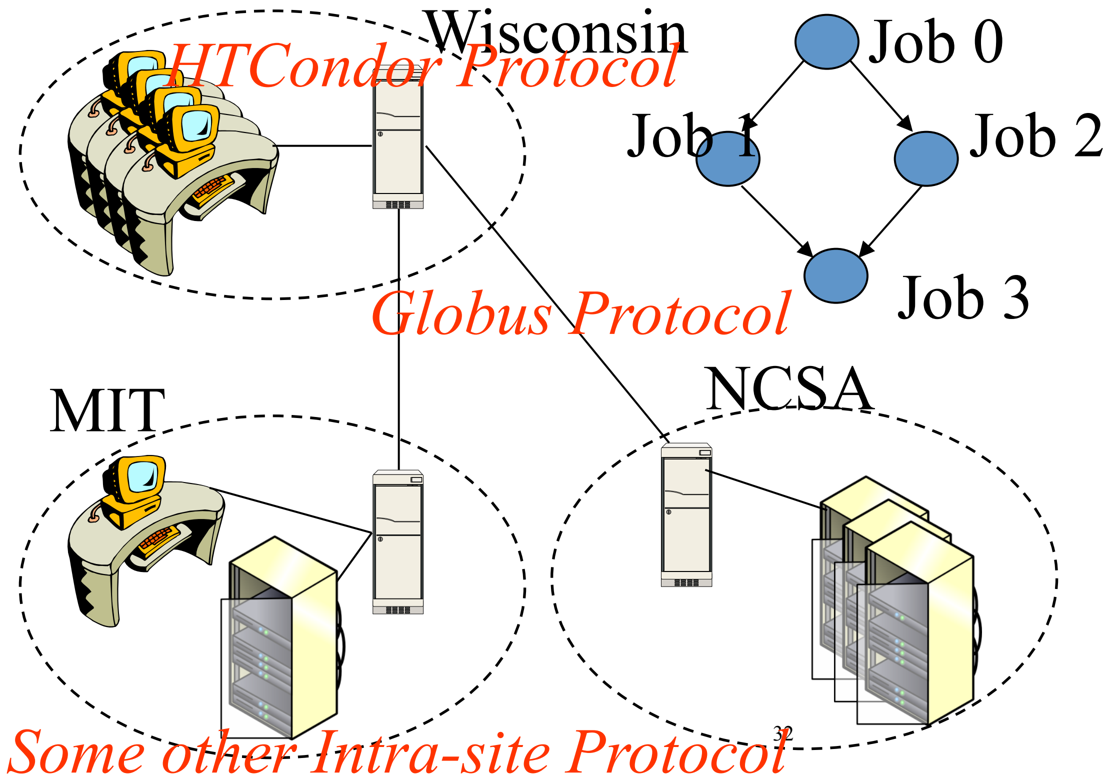
Next: Scheduling Problem



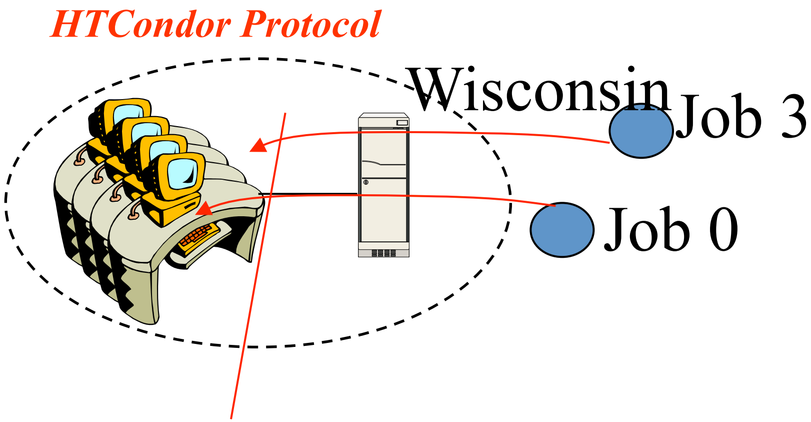
Scheduling Problem



2-level Scheduling Infrastructure



Intra-site Protocol



Internal Allocation & Scheduling
Monitoring
Distribution and Publishing of Files

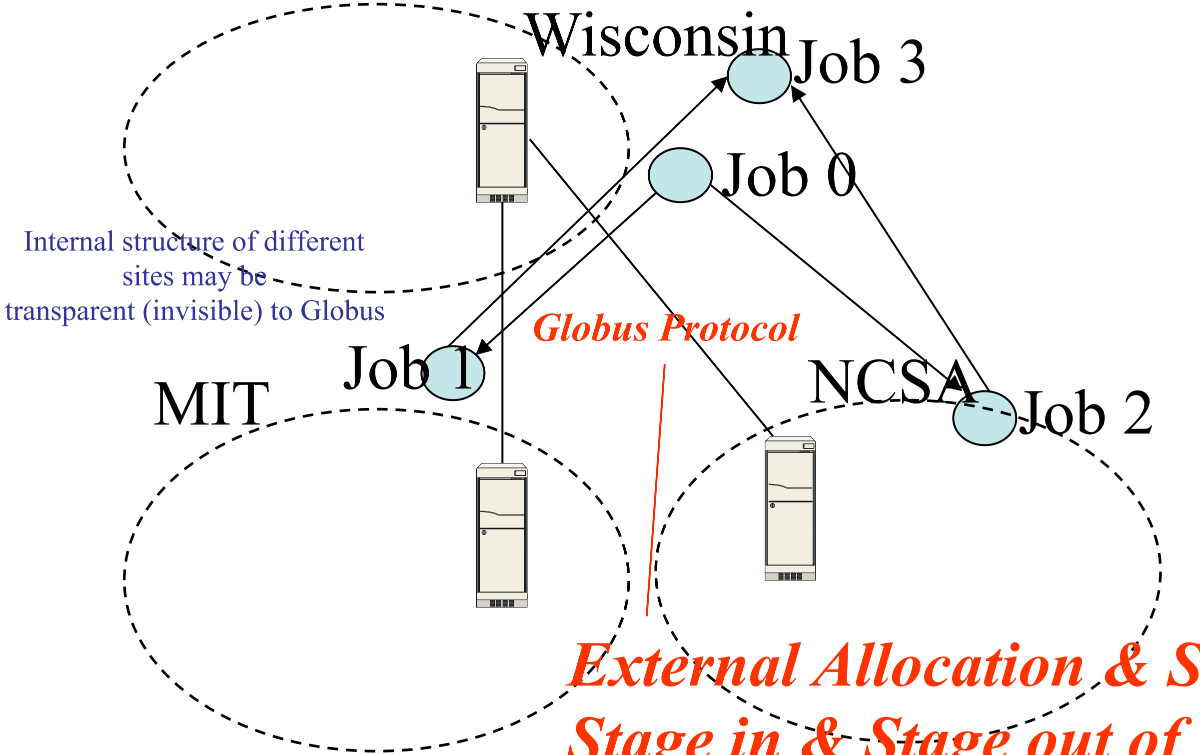
Condor (now HTCondor)

- High-throughput computing system from U. Wisconsin Madison
- Belongs to a class of Cycle-scavenging systems

Such systems

- Run on a lot of workstations
- When workstation is free, ask site's central server (or Globus) for tasks
- If user hits a keystroke or mouse click, stop task
  - Either kill task or ask server to reschedule task
- Can also run on dedicated machines

Inter-site Protocol



External Allocation & Scheduling
Stage in & Stage out of Files

Globus

- Globus Alliance involves universities, national US research labs, and some companies
- Standardized several things, especially software tools
- Separately, but related: Open Grid Forum
- Globus Alliance has developed the Globus Toolkit

http://toolkit.globus.org/toolkit/

Globus Toolkit

- Open-source
- Consists of several components
  - GridFTP: Wide-area transfer of bulk data
  - GRAM5 (Grid Resource Allocation Manager): submit, locate, cancel, and manage jobs
    - Not a scheduler
    - Globus communicates with the schedulers in intra-site protocols like HTCondor or Portable Batch System (PBS)
  - RLS (Replica Location Service): Naming service that translates from a file/dir name to a target location (or another file/dir name)
  - Libraries like XIO to provide a standard API for all Grid IO functionalities
  - Grid Security Infrastructure (GSI)

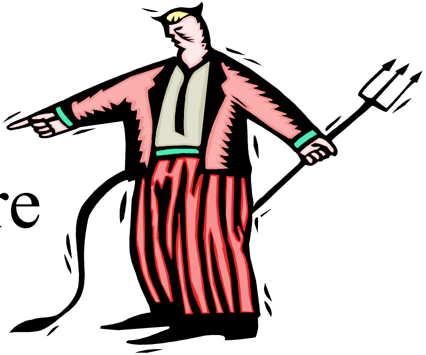
Security Issues

- Important in Grids because they are *federated,* i.e., no single entity controls the entire infrastructure

- Single sign-on: collective job set should require once-only user authentication
- Mapping to local security mechanisms: some sites use Kerberos, others using Unix
- Delegation: credentials to access resources inherited by subcomputations, e.g., job 0 to job 1
- Community authorization: e.g., third-party authentication

- These are also important in clouds, but less so because clouds are typically run under a central control
- In clouds the focus is on failures, scale, on-demand nature

Discussion Points



- Cloud computing vs. Grid computing: what are the differences?
- National Lambda Rail: hot in 2000s, funding pulled in 2009
- What has happened to the Grid Computing Community?
  - See Open Cloud Consortium
  - See CCA conference
  - See Globus

Summary

- Grid computing focuses on computation-intensive computing (HPC)
- Though often federated, architecture and key concepts have a lot in common with that of clouds
- Are Grids/HPC converging towards clouds?
  - E.g., Compare OpenStack and Globus

Projects: Where to get your ideas from

- Read through papers. Read ahead! Read both main and optional papers.
- Leverage area overlaps: x was done for problem area 1, but not for problem area 2
- Look at hot areas:
  - Stream processing (Storm)
  - Graph processing (GraphLab, LFGraph)
  - Pub-sub (Kafka)
- Look at the JIRAs of these projects
  - Lots of issues listed but not being worked on

Announcements

- Please sign up for a presentation slot by this Thursday office hours
- Please fill out survey by this Thursday (link on course website)
- Next up: Peer to peer systems