Big Data

(from an HPC perspective)

What is Big Data?

- Big Data is (generally) concerned with processing vast quantities of existing data.
  - Petabytes of log files, video or image processing, etc.
- HPC (generally) also uses or produces vast quantities of data. But …
  - Simulation codes (a major HPC usage) loads data for initial conditions and runs toward the future occasionally exporting its solution state for analysis.

- Which is which?
  - HPC: data is (generally) the start/end overhead of an intensive calculation.
  - BigData: data input and its process is the primary workload.

What is Big Data?

- But the HPC-BigData lines are blurring as streaming data is outpacing the ability to process it.
- AI training is another example of a blended field:
  - Training is computationally intensive
  - But the training sets are embarrassingly parallel and massive.

Hardware perspective:

BigData

- Parallelism: Loosely-coupled analytics, *linear* scalability.
- Networks: Bandwidth but not latency intensive (10G-ethernet)
- Storage: Massive distributed file systems with replication (HDFS). Each node is a member of the parallel storage device. Huge on-node RAM, too. (TB)

HPC

- Tightly-coupled algorithms, limited scalability.
- Latency and bandwidth equally important (e.g., Infiniband)
- Limited on-node storage. Huge off-node massive parallel storage devices (Lustre). Limited on-node RAM (GB).

Reed and Dongara, "Exascale Computing and Big Data," Communications of the ACM, v58, n7, 2015.

Hardware perspective: …

BigData

- They spend the $$ on commodity nodes with ample out-of-core (disk) storage and ample in-core (RAM) storage.
- Low latency is not as important since the disk load time will be the bottleneck regardless.
  - And they will try to load GB of data so microsecond aren't an issue.

HPC

- We spend the $$ on high speed CPUs, RAM, and networks.
  - Latency lowers scalability of the tightly coupled algorithms.
- Storage is (often) a secondary consideration.
  - And often comes back to bite us.

Hadoop:

- Very popular platform for BigData analytics.
  - Spark is another popular platform that uses some Hadoop tools
- Provides …
  - Massive distributed file system (HDFS)
  - MapReduce development framework for unstructured data processing
  - Linear scalability in storage and processing time
    - Add nodes to increase storage capacity linearly or reduce processing time
  - Distributed resource management

Hadoop:

- HDFS:
  - Ingest user data and replicate the data as directed by the managing NameNode.
  - User input is split into blocks (64 MB default). Each block is replicated 3x on unique DataNodes for redundancy and scheduling of parallel work.
- MapReduce:
  - A generic two-phase algorithm, nothing unique to Hadoop. (can do with MPI)
  - *Map* phase processes blocks on the DataNodes (is alive, isn't busy, has a piece of my data) … that's why data is replicated: redundancy and scalability.
  - *Reduce* processes the individual (independent) Map'd data in parallel across the DataNodes to generate 'the solution'.

MapReduce Example:

- How many times have I been mentioned in the NYT in the past 10 years?

1. Ingest all digital editions of the NYT for 10 years as searchable text.
   - Split the dataset into tiny pieces with some care to keep words (or lines or dates) together.
2. Execute the Map phase:
   - Each process on the DataNodes searches its piece of the data for my name and keeps a count of occurrences: simple string matching and counter incrementing.
3. Execute the Reduce phase:
   - Each member of the DataNodes that processed data sends its local count to the reducer using generic reduction operations. (Not unlike what you did for HW3 in the search() function.)

- And the answer is … 0.

Parallel I/O in HPC:

- Get the data into or out of the application as fast as possible.
  - The data is the most important element (of the code) but it's a bottleneck.
  - I/O is a burst in HPC but a continuous stream (or river) in BigData.
- MPI supports parallel I/O operations.
  - Each process can write to a section of a file concurrently
  - Needs an underlying parallel FS to distribute the incoming data streams.
  - Most parallel FS's use a similar approach as HDFS (eg. DataNodes + NameNodes) but use file striping (~1MB) and parity to store the file.
    - Redundancy with higher storage efficiency (80-90% v. 33% in HDFS).
    - Blocks of a file are meaningless (usually) in HPC so we will need to recreate the whole file at some point. We can't exploit the block-level parallelism like in HDFS + MapReduce.