ECE 574 – Cluster Computing
Lecture 22

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

11am Barrows 133

12 April 2024

Announcements

- Second Midterm will be Tuesday the 16th (Tuesday)
  Cumulative, though concentrating on recent material
- Project Status Report (see next)

Status Report

- Provide some “related work” for your project. Do a literature search and find examples of other people who have done similar research. (Since this is a grad class)
  - Good skill to have for writing papers.
  - As they say, a month in the lab replaced by few hours in library
  - It’s OK if you find something exactly the same. Reproducing work is important and under-utilized in academia.

  - Even for masters degree it's OK if similar work is out there
  - PhD they expect you to have something unique
- I'd prefer if the references you find are books or academic papers, but (especially for programming projects) blog or website references are OK
  NOTE: don't pay for ACM/IEEE/etc papers. You can get them for free through UMaine library website
- Send report by e-mail, only one submission per group.
- State in one sentence a summary of your project
- Describe the hardware and software that you will be

using.

- Have you made any progress on running code in such a setup
- The related work. Two references are fine for a single-person project, at least four if you have more than one person in your group.
- Will you be willing to volunteer to present early (Thursday 18th/23rd/25th)

Midterm Review

- Can have 1 page of notes (1 side, 8.5"x11")
- Cumulative, but concentrating on stuff since last exam
- Performance, Speedup, Parallel efficiency, Scaling
- Shared Memory vs Distributed Systems
- Tradeoffs. Given code and hardware, would you use
  - MPI – distributed systems, large problems
  - OpenMP – shared memory, fit on one node / CPU
  - CUDA – when you have an NVIDIA gpu
  - OpenCL – when you have a non-NVIDIA gpu

  - pthreads – would you ever use if you didn't have to?
- OpenMP: dynamic vs static scheduling
- MPI, especially its limitations
- GPGPU/CUDA/OpenCL: read code, know the tradeoffs (overhead of copying memory around)
- BigData: sizes involved, distributed filesystems
- Reliability. Causes of errors. Tradeoffs of Checkpointing
- Energy, Energy Delay, Time, Performance

Big Data

- Until now compute or network bound systems
- What if we want to do a lot of disk/IO? Big Data?
- There are often trends in Computer Research
  - Takes a while to trickle down to UMaine, funny watching how administration will launch an initiative for the new "hot topic" after it has already peaked and is on the downswing elsewhere
  - Big Data was the previous hot thing, and sure enough UMaine just finalized its effort for it

- The current big thing is AI. I thought that had peaked too, but this time the Chat-GPT stuff came in at the last minute as a bit of a curve-ball so maybe it will stick around longer

Can you name big data datasets?

- Physics (particle accelerators)
- Astronomy
- Genomics
- AI Training
- Web search
- Social media
- Streaming (youtue, netflix)

Big Data Examples

- Where is Data Used a lot?
  - Google
  - Worldwide LHC Computing Grid (WLCG) as of 2012
    25 petabytes of data/year (petabyte=1000 terabytes)
    300 GByte/s of data incoming
- Big Data is about big data sets (Terabytes?) and trying to get useful information out of them. Data Mining.
- "Big Data: Astronomical or Genomical": PLos Biology, 7 July 2015. (as per IEEE Spectrum

December 2015). Twitter: 1-17PB/year. Astronomy, 1,000PB/year, YouTube 1,000-2,000 PB/year, Genomics 2,000-40,000PB/year

Big Data Challenges

- Data capture
- Data storage
- Data analysis / Visualization
- Data search / Querying
- Data sharing
- Data transfer
- Updating
- Information privacy (?) afterthought?

the Big Data Vs

- Volume – quantity, terabytes? Petabytes?
- Variety – more than just lists of numbers
- Velocity – speed the data is generated
- Veracity – GIGO (garbage in/garbage out)
- Value – Usefulness
- Variability –

Big Data

- A buzzword?
- How big is big?
- Terabytes?
- Too big for one machine?
- In general if fits in RAM $(< 32GB)$ or fits on disk $(< 10TB)$ you are better off just using a database or similar
- Once it won't fit on one machine, and you want to use a cluster, things get complicated.

Big Data Optimizations

- Key idea is to *move computation to the data*, rather than vice-versa

Big data challenges/astronomy

- `https://www.computerworld.com/article/2972251/massive-telescope-array-aims-for-black-hole-gets-gu`

  `html`
- Black hole “picture”
- From radio-wave interferometry
- Telescopes scattered all over world, including Antarctica
- Hard drives fail on mountain tops! (not enough air) use helium-filled ones instead
- Over 5 days, each telescope collected 900TB of data
- 1000-2000 hard drives, about 9PB

- How data sent? Hard-drives shipped to Massachusetts
- Had to wait for spring in Antarctica to ship out those
- 800 core cluster to analyze

Types of Storage

- Hard Disks
  - spinning rust – can be slow latency wise
  - SSD – faster, why?
  - Traditional vs Advanced features

    Shingled (SMR) Disks

    Perpendicular (PMR) Disks

    `https://www.youtube.com/watch?v=xb_PyKuI7II`

    Helium

    Caches

- Other flash / SD cards
- Memristors/Phase-Change/Optane/XPoint Non-volatile RAM
- Tape – robot tape libraries, etc
- Optical – CD/DVD/Blueray

RAID

- Redundant Array of (Independent / Inexpensive) Disks
- Patterson Gibson and Katz 1987: replace expensive mainframe disks with arrays of relatively cheap desktop drives
- RAID0: striping, spreading across multiple disks, can increase performance, increases size of disk, bad things happen if one drive fails
- RAID1: mirroring – same thing written to both drives can increase performance as either drive can answer

request

- RAID2: hamming code, each bit on separate drive. Not really used.
- RAID3: byte-level striping with parity. not common
- RAID4: block-level striping with dedicated parity.
- RAID5: block-level striping with distributed parity.
  can handle failure of single disk, can rebuild based on parity.
  Not recommended, as you have to read entirety of all other disks to rebuild, likely to fail other disks if all of same vintage

- RAID6: block-level striping with double parity. Recommended
- Hybrid: RAID10 = RAID1 + RAIDO
- Software vs Hardware
- Some filesystems include RAID like behavior: ZFS, GPFS, brfs, xfs

RAID5/6 further notes

- raid5, you need at least three drives. Data is striped across first two drives, third drive gets the bytes XOR with each other
- If a drive fails, you can xor the data from the two remaining drives to get the missing data
- This does waste some space. Also rebuilding can be stressful on remaining drives and cause them to fail
- RAID6 has an extra drive for redundancy, but reduces capacity

- Modern systems often do RAID10 (RAID1+0) which is striping plus full backup. Wastes half the space, but doesn't require XOR

Non-RAID

- nearline storage – not offline but also not fully online
- MAID – Massive Array of Idle Drives
  - Write once, read occasionally.
  - Data that you want to save, but really don't access often so taking seconds to recall is OK.
  - What kind of data? Backups? Old Facebook pictures?
  - Old science data?

Traditional Ways to manage Data

Databases / RDBMs

- Machines that store large amounts of data, often optimized for fast retrieval
- Databases / Relational Database Management System
- Relational databases: store rows of data, with a key. Each field has attribute.

  `Item, Name, Price, Color, Rating`

SQL (structured query language)

- Standard for querying relational database (this was a past “hot” computing topic)
-
```
SELECT *
FROM  Book
WHERE price > 100.00
ORDER BY title;
```

- Consistency?

SQL Challenges

- Can have parallel and distributed databases too. It's more difficult with SQL
  - Replication – task runs, making sure all the various copies are kept in sync
  - Duplication – there is a master, and all the others are copies of the master. Users may only change master
- Main memory database – machines with 100TB of RAM?

NoSQL Databases

- Scale-out architecture
  Can increase performance by adding nodes (rather than by upgrading single machine)
- Can store unstructured data (json, binary, text, sparse)
  Doesn't have to map to rows
- Can spread out on other machines, into cloud

Cluster Filesystem

- Filesystem shared by multiple machines/nodes
- Can be centralized or distributed

Shared-disk Filesystem

- Shared-disk filesystem – shares disk at block level
- SGI CXFS
  IBM GPFS
  Oracle Cluster Filesystem (OCFS)
  RedHat GFS
  Many Others
- RedHat GFS2
  Distributed Lock Manager

DAS – Direct Attached Storage

- typically how you hookup a hard-drive
- No network involved
- Relative low latency, but not distributed.
- Can be used as cache
- Can be exported for use as part of distributed filesystems

SAN – Storage Area Network

- (Don't confuse with NAS – network attached storage)
- A network that provides block-level access to data over a network and it appears to machines the same as local storage
- SAN often uses fibrechannel (fibre optics) but can also be over Ethernet

NAS – network attached storage

- Like a hard-drive you plug into the Ethernet but serves files (not disk blocks) usually by SMBFS (windows sharing), NFS, or similar
- NAS: appears to machine as a fileserver, SAN appears as a disk

Network Storage Concerns

- QoS – quality of service. Limit bandwidth or latency so one user can't overly impact the rest
- Deduplication

Cluster Storage

- Client-server vs Distributed
- Multiple servers?
- Distributed metadata – metadata spread out, not on one server
- Striping data across servers.
- Failover – if network splits, can you keep writing to files
- Disconnected mode.

Non-Distributed Network Filesystems

- NFS, SMBFS, Netware

Distributed Filesystem Architectures

From *A Taxonomy and Survey on Distributed File Systems*

Can be any combination of the following

- Client Server – like NFS
- Cluster Based – single master with chunk servers
- Symmetric – peer to peer, all machines host some data with key-based lookup
- Asymmetric – separate metadata servers
- Parallel – data striped across multiple servers

Stateless

Can you reboot server w/o client noticing? Lower overhead if server stateless because the server doesn't have to track every open file in the system

Synchronization/File Locking

- Multiple users writing to same file?
- Always synchronous?
- Same problems with consistency that you have with caches/memory

Consistency and Replication

- Checksums to validate the data
- Caching – if you cache state of filesystem locally, how do you notice if other nodes have updated a file?

Failure Modes

Security

Distributed Filesystems

- Follow a network protocol, do not share block level access to disk
- Transparency is important. User doesn't need to know how it works underneath.
- Ceph, GFS, GlusterFS, Lustre, PVFS

Lustre

- "Linux Cluster"
- Complex ownership history
- Old article: `http://lwn.net/Articles/63536/`
- Used by many of the top 500 computers
  As of 2020, 77 of top 100 (rest IBM Spectrum scale)
  Frontier has 700TB 5GB/s Lustre filesystem
- Can handle tens of thousands of clients, tens of petabytes of data across hundreds of servers, and TB/s of I/O.
- One or more metadata servers: keep track of what files

exist, metadata, etc, locking, can load balance.

- One or more object storage servers
  Boxes of bits accessed by unique tag
- File can be "striped" across multiple storage servers and stream the file data in parallel
- Failure recovery. If node crashes, other nodes remember what it missed while down and help it recover to the proper state
- Distributed Locking
- Fast networking. Use RDMA when available.

Big Data Tools

- There are various
- Hadoop was one of the more popular

Hadoop

- A distributed filesystem (HDFS)
- A way to run map-reduce jobs

Hadoop

- Apache
- Distributed Processing and Distributed Storage on commodity clusters
- Java based
- Data spread throughout nodes
  Large data sets split up and spread throughout the cluster
- Unlike traditional HPC clusters, code sent *to the nodes* that have data of interest, rather than taking data over

network to running code.

- HADOOP common – libraries
- HADOOP YARN – thread scheduling
- Hadoop Distributed File System – HDFS
- Hadoop MapReduce – processing algorithm
- Originally developed at Yahoo by Cutting and Cafarella. Named after toy elephant.
- Many users. As of 2012 Facebook had 100PB of data, said it grew at 0.5PB/day

Hadoop Distributed Filesystem

`https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`

- Keeps working in face of hardware failures
- Streaming data access – optimize for bandwidth, not latency
  Relaxes some POSIX assumptions
- Large data sizes – optimized for files of gigabytes to terabytes
- Write-once-read-many – assumption is the data isn't being actively written.

- “Moving computation easier than moving data”
- blocksize and replication factor per-file
- Rack-aware filesystem
- “location awareness” Tries to spread code out multiple copies distributed physically
- Data spread throughout nodes. Default replication value of 3, duplicated three times, twice on same rack and once on different
- Namenode plus cluster of datanodes
- Namenode tracks filenames and locations, keeps entire map in memory

- Datanode stores data. Uses local computer's underlying filesystem. Just blocks of data, makes directories as appropriate but doesn't necessarily have any relationship to the files as seen from within HDFS.
- Communication is over TCP

HDFS Fault Handling

- Datanodes send heartbeats to namenode. When datanodes go missing, marked as dead, no new I/O sent to them. If any files fall below replication level they can be replicated on remaining nodes
- Rebalancing – if disk availability changes files might be moved around
- Integrity – checksums on files to detect corruption
- Namenode is a single point of failure. Keeps the edit log and fsimage, only syncs at startup

Data Organization

- Data broken up into chunks, default 64MB
- Creating a file does not necessarily allocate a chunk; it is cached locally and only sent out once enough data has accumulated to fill a block
- Replication pipeline: once file created starts being sent in smaller chunks (4kb) and it gets forwarded 1 to 2 to 3 in a pipeline until file in all places.
- Deleting a file does not delete right away, moved to /trash After configurable time gets deleted from trash

and the blocks are marked as free. It can take a while for this to all happen, deletes do not free up space immediately.

- Not a full POSIX filesystem. Writes are slow, and you can't write to an existing file.

Map Reduce

- Originally popularized by Google, but not really used by them anymore (after 2014)
  Jeffrey Dean, Sanjay Ghemawat (2004) MapReduce: Simplified Data Processing on Large Clusters, Google.
- For processing large data sets in parallel on a cluster
- Similar to MPI reduce and scatter operations
- Map() – filters and sorts data into key/value pairs
  Stateless, can run in parallel
  can contain Combiner() – combines duplicates?

- Reduce() – the various worker nodes process each group in parallel.
  Shuffle() – redistribute data so all common data on same node
- Can do with single processor systems, but not any faster typically. Shines on parallel systems

Map Reduce Example

```
The quick brown fox jumped over the lazy dog.
```

MAP split by key (in this case, number of letters)

```
3: [the, fox, the, dog]
4: [over, lazy]
5: [quick, brown]
6: [jumped]
```

REDUCE each thread/node gets one of these. Reduce might simply count.

```
3: 4
4: 2
5: 2
6: 1
```

Another Map Reduce Example: Hello World

This is the example they like to use.

Map: key is the word

```
To be or not to be, that is the question.
```

```
to: [1, 1]
be: [1, 1]
or: [1]
not: [1]
that: [1]
is: [1]
```

```
the: [1]
question: [1]
```

Reduce:

```
to: 2
be: 2
or: 1
not: 1
that: 1
is: 1
the: 1
question: 1
```

Real world friends example

- http://stevekrenzel.com/finding-friends-with-ma
- https://www.tutorialspoint.com/hadoop/hadoop_
  mapreduce.htm

Why would you do things like this?

- You can see how this comes out of search research
- Have terrabytes of spidered websites you want to do a text search on? (Maybe HPC?)
- Having one thread read all terabytes over the network to central location and searching, take forever
- Instead, data spread across millions of machines
- Send code that first does a map to find out how many times HPC occurs on each file
- Then reduce down to MAX and find out which are most

relevant

Submitting a Job

- Job:
  Specify input and output on filesystem
  The jar file (java class) of the map and reduce functions
  Job configuration
- Hadoop client sends this to the scheduler

Scheduling

- Each location of system known. Try to run code on same system as data for locality, If not possible, run on one nearby.
- Small cluster has single master node, and multiple worker nodes.
- Hardware does not have to be fault tolerant; if a map/reduce fails it is simply retried again (on another machine)
- You can add/remove hardware at any time

Hadoop Update

Can set up Hadoop on single machine, even the name and data servers. Just download big chunk of Java, have Java and ssh installed.

Data Warehouse

- Enterprise Data Warehouse (EDW)
- Business gather data
- ETL: Extract, Transform, Load

Other Big Data codebases

- Google BigQuery
- Apache Spark
- Apache Storm

Google Big Query

- “serverless data warehouse”
- Petabytes of data
- “Platform as a service”
- SQL, Machine learning
- Import data as CSV, JSON, etc
- Use Google Dremel (for interactive querying of large databases)

Apache Spark

- Interface for programming clusters with data parallelism and Fault Tolerance (made at Berkeley)
- Resilient Distributed Datasets (RDD), read only multiset of data distributed over large cluster, fault tolerant
- Dataset API
- Replacement for Map Reduce / Hadoop, latency several orders of magnitude better
- Iterative algorithm can repeatedly visit
- Good for machine learning workloads

- Has a cluster manager
  - Native Spark
  - Hadoop Yarn
  - Apache Mesos
  - Kubertenes
- Uses distributed storage
  - Alluxio
  - HDFS
  - Casandra
  - Amazon S3
  - Openstack

- Kudu
- ?

More Apache Spark

- “HPC is dying and MPI is Killing it” article (2015)
- Java / Scala / Python / R
- Two components
  - Driver, converts code to multiple tasks
  - Executor: runs on nodes
- Originally ran on Hadoop Yarn, can also now via Kubertenes

Apache Spark – RDDs

- Resilient Distributed Dataset (RDD)
- Can be text, SQL, NoSQL, amazon s3 bucket
- Fault-tolerant, immutable (can't change) distributed set of objects, divided into logical partitions
- Creation/Transform/Act:
  - Create from file or bucket and parallelize with a command
  - Run a transform on it (sort of like map)
    Doesn't update current RDD, but creates new one

- Run an action on it. Count, first, max, reduce, collect

Apache Spark Example

- Install it
- Run spark-shell
- Run spark-submit

Apache Storm

- Uses clojure (Lisp/Java)
- Distributed Stream Processing
- Distributed Process Stream Data
- Pass it Directed Acyclic Graph (DAG), “spouts” and “bolts” at vertices, edges are streams
- Master nodes execute daemon, Numbys
- Worker nodes

Apache Drill

- Clone of google dremel

Apache Impala

- Massively Parallel SQL query engine
- ?

Facebook Presto

- ?