cloudera

Thinking at Scale

Overview

- Characteristics of large-scale computation
- Pitfalls of distributed systems
- Designing for scalability

By the numbers…

- Max data in memory: 32 GB
- Max data per computer: 12 TB
- Data processed by Google every month: 400 PB … in 2007
- Average job size: 180 GB
- Time that would take to read sequentially off a single drive: 45 minutes

What does this mean?

- We can process data **very quickly** but we can read/write it **very slowly**
- Solution: parallel reads

- 1 HDD = 75 MB/sec
- 1000 HDDs = 75 GB/sec
  - Now you're talking!

Sharing is Slow

- Grid computing: not new
  - MPI, PVM, Condor…
- Grid focus: distribute the **workload**
  - NetApp filer or other SAN drives many compute nodes
- Modern focus: distribute the **data**
  - Reading 100 GB off a single filer would leave nodes starved – just store data locally

Sharing is Tricky

- Exchanging data requires synchronization
  - Deadlock becomes a problem
- Finite bandwidth is available
  - Distributed systems can “drown themselves”
  - Failovers can cause cascading failure
- Temporal dependencies are complicated
  - Difficult to reason about partial restarts

Ken Arnold, CORBA designer:

"Failure is the defining difference between distributed and local programming"

Reliability Demands

- Support partial failure
  - Total system must support graceful decline in application performance rather than a full halt

Reliability Demands

- Data Recoverability
  - If components fail, their workload must be picked up by still-functioning units

Reliability Demands

- Individual Recoverability
  - Nodes that fail and restart must be able to rejoin the group activity without a full group restart

Reliability Demands

- Consistency
  - Concurrent operations or partial internal failures should not cause externally visible nondeterminism

Reliability Demands

- Scalability
  - Adding increased load to a system should not cause outright failure, but a graceful decline
  - Increasing resources should support a proportional increase in load capacity

A Radical Way Out…

- Nodes talk to each other as little as possible – maybe never
  - “Shared nothing” architecture
- Programmer should not explicitly be allowed to communicate between nodes
- Data is spread throughout machines in advance, computation happens where it’s stored.

Motivations for MapReduce

- Data processing: > 1 TB
- Massively parallel (hundreds or thousands of CPUs)
- Must be easy to use
  - High-level applications written in MapReduce
  - Programmers don’t worry about socket(), etc.

Locality

- Master program divvies up tasks based on location of data: tries to have map tasks on same machine as physical file data, or at least same rack
- Map task inputs are divided into 64—128 MB blocks: same size as filesystem chunks
  - Process components of a single file in parallel

Fault Tolerance

- Tasks designed for independence
- Master detects worker failures
- Master re-executes tasks that fail while in progress
- Restarting one task does not require communication with other tasks
- Data is replicated to increase availability, durability

Optimizations

- No reduce can start until map is complete:
  - A single slow disk controller can rate-limit the whole process
- Master redundantly executes “slow-moving” map tasks; uses results of first copy to finish

Conclusions

- Computing with big datasets is a fundamentally different challenge than doing “big compute” over a small dataset
- New ways of thinking about problems needed
  - New tools provide means to capture this
  - MapReduce, HDFS, etc. can help

cloudera