Performance Considerations

Lecture BigData Analytics

Julian M. Kunkel

julian.kunkel@googlemail.com

University of Hamburg / German Climate Computing Center (DKRZ)

18-12-2015

Outline

1 Overview

2 Hardware

3 Assessing Performance

4 Benchmarks

5 Summary

Goals

- Goal (user perspective): minimal time to solution
  - Solution = workflow from data ingestion, programming to analysis results
  - Programmer/User productivity is important
- Goal (system perspective): cheap total cost of ownership
  - Simple deployment and easy management
  - Cheap hardware
  - Good utilization of (hardware) resources means less hardware

$\Rightarrow$ In this lecture, we focus on the processing of a workflow

Processing Steps

1. Ingesting data into our big data environment
2. Processing the workflow with (multiple) Hive/Pig/... queries
   - Low runtime is crucial for repeated data analysis and data exploration
   - Important factor for the productivity of data scientists
   - Multiple steps/different tools can be involved in a complex workflow

     We consider only the execution of one job with any tool
3. Post-processing of output with (external) tools to produce insight

Performance Factors Influencing Processing Time

- Startup phase
  - Distribution of necessary files/scripts
  - Allocating resources/containers
  - Starting the scripts and loading dependencies
  - Usually fixed costs
- Job execution: computing the product
  - Costs for computation and necessary communication & I/O depend on
    - Job complexity
    - Software architecture of the big data solution
    - Hardware performance and cluster architecture
- Cleanup phase
  - Teardown containers, free resources
  - Usually fixed costs

BigData Cluster Characteristics

- Usually commodity components
- Cheap (on-board) interconnect, node-local storage
- Communication (bisection) bandwidth between different racks is low

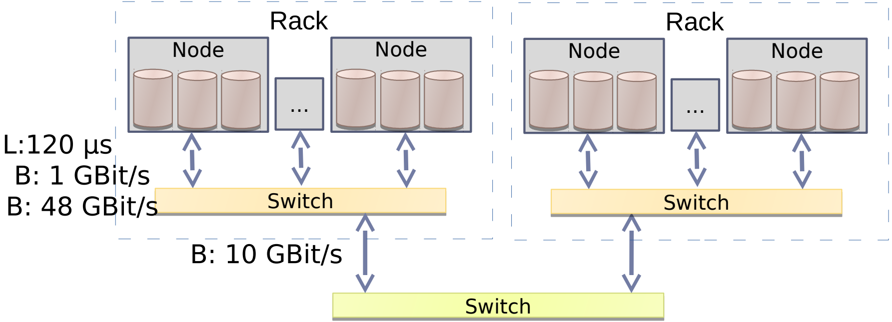


Figure: Architecture of a typical BigData cluster

HPC Cluster Characteristics

- High-end components
- Extra fast interconnect, global/shared storage with dedicated servers
- Switches provide high bisection bandwidth

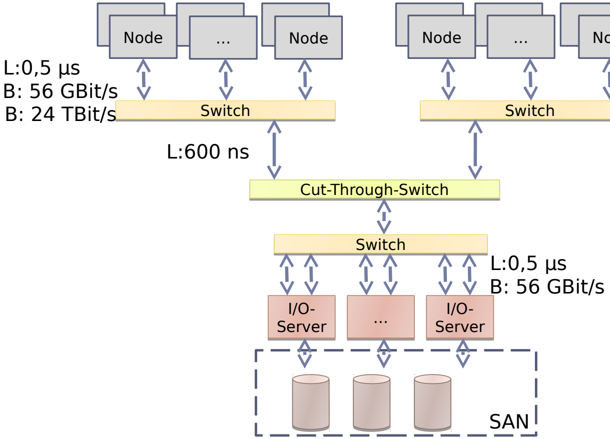


Figure: Architecture of a typical HPC cluster.

Hardware Performance

Computation

- CPU performance (frequency · cores · sockets)
  - e.g. 2.5 GHz · 12 cores · 2 sockets = 60 Gcycles/s
  - The number of cycles per operation depend on the instruction stream
- Memory (throughput · channels); e.g. 25.6 GB/s per DDR4 DIMM ·3

Communication via the network

- Throughput e.g. 125 MiB/s with Gigabit Ethernet
- Latency e.g. 0.1 ms with Gigabit Ethernet

Input/output devices

- HDD mechanical parts (head, rotation) lead to expensive seek

$\Rightarrow$ Access data consecutively and not randomly

$\Rightarrow$ Performance depends on the I/O granularity, e.g. 150 MiB/s

Hardware-Aware Strategies for Software Solutions

- Use Java: 1.2 - 2x more cycles needed than C
- Utilize different hardware components concurrently
  - Pipeline computation, I/O and communication
  - At best hide two of them $\Rightarrow$ 3x speedup
  - Avoid barriers (waiting for the slowest component)
- Balance and distribute workload among all available servers
  - Linear scalability is vital (and not the programming language)
  - Add 10x servers, achieve 10x performance
- Avoid I/O if possible (keep data in memory)
- Avoid communication if possible
- Allow monitoring of components to see their utilization

Examples for Pig/Hive

- Foreach, Filter are node-local operations
- Sort, group, join need communication

Basic Approach

Question

Is the observed performance acceptable?

Basic approach

Start with a simple model

1. Measure time for the execution of your workload
2. Quantify the workload with some metrics
   - e.g. amount of tuples or data processed, computational operations needed
   - e.g. you may use the statistis output for each Hadoop job
3. Compute *wt*, the workload you process per time
4. Compare wt with your expectations of the system

Refine the model as needed e.g. include details about intermediate steps

Updating Batch Views

Scenario

- While performing a batch update new data is captured
- We have to keep up with the data generation
- How long is the processing time? Should we upgrade hardware?

A simple model [11]

- T Runtime of the batch update
- O Overhead for startup/cooldown of the batch (independent to size)
- A Time data is captured in the system that is processed per batch
- P Additional processing time per time unit of data
  - e.g. add 30 minutes processing for 60 minutes of data = 0.5
- Runtime of the workflow: $T = O + P \cdot A$
- Equilibrium with incoming data: $A = T \Rightarrow T = \frac{O}{1-P}$

Updating Batch Views: Model Equilibrium

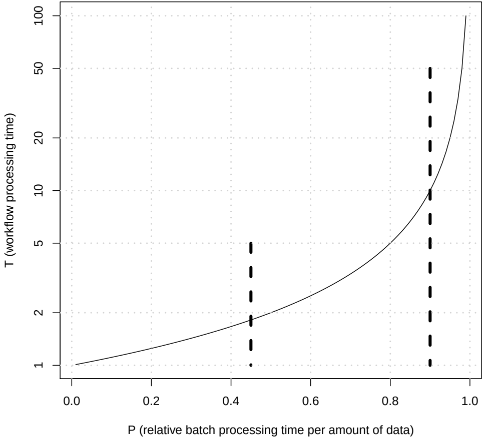
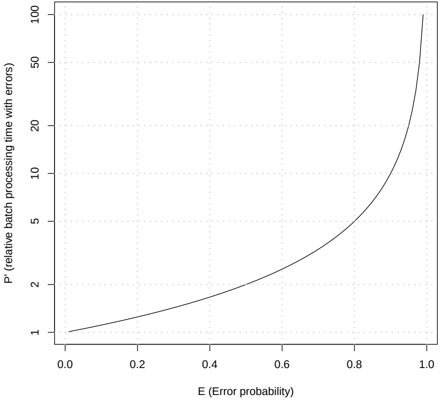


Figure: Time for updating a batch view with a variable proc. time for const. overhead

- Assume constant overhead of 1 time unit
- Processing time increases significantly with $P \rightarrow 1$
  - e.g. processing 90% of time takes 5 times longer than with 45%
  - Requires about twice the hardware resources to half P
- Buying more hardware is efficient for $P > 0.5$

Errors while Processing [11]

- Error probability $E < 1$ increases the processing time
- A rerun of a job may fail again
- Processing time with errors: $P' = (E + E^2 + ...) \cdot P' = P/(1 - E)$



- Familiar graph
- With 50% chance of errors, twice the processing time

Figure: Processing time & error probability

Daytona GraySort

- Sort at least 100 TB data in files into an output file
  - Generates 500 TB of disk I/O and 200 TB network I/O [12]
  - Drawback: Benchmark is not very compute intense
- Records: 10 byte key, 90 byte data
- Performance Metric: sort rate (TBs/minute)

	Hadoop MR Record	**Spark Record**	**Spark 1 PB**
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

Figure: Source: [12]

Assessing Performance

Hadoop

- 102.5 TB in 4,328 seconds [13]
- Hardware: 2100 nodes, dual 2.3Ghz 6cores, 64 GB memory, 12 HDDs
- Sort rate: 23.6 GB/s = 11 MB/s per Node $\Rightarrow$ 1 MB/s per HDD
- Clearly this is suboptimal!

Apache Spark (on disk)

- 100 TB in 1,406 seconds [13]
- Hardware: 207 Amazon EC2, 2.5Ghz 32vCores, 244GB memory, 8 SSDs
- Sort rate: 71 GB/s = 344 MB/s per node
- Performance assessment
  - Network: 200 TB $\Rightarrow$ 687 MiB/s per node
    Optimal: 1.15 GB/s per Node, but we cannot hide (all) communication
  - I/O: 500 TB $\Rightarrow$ 1.7 GB/s per node = 212 MB/s per SSD
  - Compute: 17 M records/s per node = 0.5 M/s per core = 4700 cycles/record

Executing the Optimal Algorithm on Given Hardware

An Utopic Algorihm

Assume 200 nodes and random key distribution

1. Read input file once: 100 TB
2. Pipeline reading and start immediately to scatter data (key): 100 TB
3. Receiving node stores data in likely memory region: 500 GB/node
   Assume this can be pipelined with the receive
4. Output data to local files: 100 TB

Estimating optimal runtime

Per node: 500 GByte of data; I/O: keep 1.7 GB/s per node

1. Read: 294s
2. Scatter data: 434s $\Rightarrow$ reading can be hidden
3. One read/write in memory (2 sockets, 3 channels): 6s
4. Write local file region: 294s

Total runtime: $434 + 294 = 728 \Rightarrow 8.2$ T/min

In-Memory Computing

Aggregating 10 M integers with 1 thread

- Spark [14]: 160 MB/s, 500 cycles per operation (should use all threads)
- Raw Python: 0.44s = 727 MB/s, 123 cycles per operation
- Numpy: 0.014s = 22.8 GB/s, 4 cycles per operation

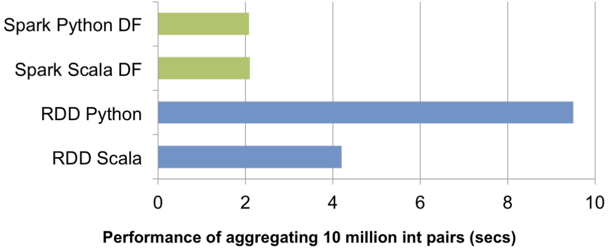


Figure: Source: [14]

⇒ External programming languages in Spark are even more expensive!

Comparing Pig & Hive

Benchmark by IBM [16], similar to Apache Benchmark

- Tests several operations, data set increases 10x in size
  - 1: 772 KB, 2: 6.4 MB, 3: 63 MB, 4: 628 MB, 5: 6.2 GB, 6: 62 GB
- 5 data/compute nodes, configured to run 8 reduce and 11 map tasks

	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
Arithmetic	32	36	49	83	423	3900
Filter 10%	32	34	44	66	295	2640
Filter 90%	33	32	37	53	197	1657
Group	49	53	69	105	497	4394
Join	49	50	78	150	1045	10258

Figure: Pig. Source: B. Jakobus, "Table 1: Averaged performance" [16]

	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
Arithmetic	32	37	72	300	2633	27821
Filter 10%	32	53	59	209	1672	18222
Filter 90%	31	32	36	69	331	3320
Group	48	47	46	53	141	1233
Join	48	56	10	517	4388	-
Distinct	48	53	72	109	-	-

Figure: Hive. Source: B. Jakobus, "Table 2: Averaged performance" [16]

Summary

- Goal (user-perspective): optimize the time-to-solution
- Runtime of queries/scripts is the main contributor
- Compute in big data clusters is usually overdimensioned
- Understanding a few hw throughputs helps assessing performance
- Linear scalability of the architecture is the crucial performance factor
- Basic performance analysis
  1. Estimate the workload/s
  2. Compare with hardware capabilities
- Model for batch update predicts benefit of upgrades
- Error model predicts runtime if jobs must be restarted
- GreySort with Spark utilizes I/O, communication well
- Computation even with Spark is much slower than Python
- Big data solutions exhibit different performance behaviors

Bibliography

10 Wikipedia

11 Book: N. Marz, J. Warren. Big Data – Principles and best practices of scalable real-time data systems.

12 https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

13 http://sortbenchmark.org/

14 https://databricks.com/blog/2015/04/24/recent-performance-improvements-in-apache-spark-sql-python-dataframes-and-more.html

15 https://github.com/hortonworks/hive-testbench

16 http://www.ibm.com/developerworks/library/ba-pigvhive/pighivebenchmarking.pdf