Department of Computer Science

University of Reading

Performance of Distributed and Parallel Systems

Julian Kunkel

2019-10-28

Learning Objectives

- Describing relevant performance factors
- Listing peak performance of relevant components
- Assessing/Judging observed application performance

Outline

1 Overview

2 Hardware

3 Assessing Performance

4 Summary

Goals

University of Reading

- In the context of this lecture, we assume the **goal of a system** is data processing
- Goal (user perspective): Minimal time to solution
  - For Big Data: Workflow from data ingestion, programming, results analysis
  - For Science: Workflow until scientific insight/paper
  - Programmer/User productivity is important
- Goal (system perspective): cheap total cost of ownership
  - Simple deployment and easy management
  - Cheap hardware
  - Good utilisation of (hardware) resources means less hardware

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

- Other "performance" alike aspects:
  - Productivity of users (user-friendliness)

Processing Steps

University of Reading

1. Preparing input data
   - Big Data: Ingesting data into our big data environment
   - HPC: Preparing data for being read on a supercomputer
2. **Processing** a workflow consisting of multiple steps/queries
   - It is a relevant factor for the productivity in data science
   - Low runtime is crucial for repeated analysis and interactive exploration
   - 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
   - Typical strategy of scientists: HPC/Big Data workflow – data transfer – local analysis
   - Best: return a final product from the workflow
   - For exploratory/novel research, the result is unknown, and may require a long period of manual analysis

Performance Factors Influencing Processing Time

University of Reading

- Startup phase
  - Distribution of necessary files/scripts
  - Allocating resources/containers
  - Starting the scripts and loading dependencies
  - Usually fixed costs (in the order of seconds to spawn MR/TEZ job, also for HPC jobs!)
- **Job execution**: computing the product
  - Costs for computation and necessary communication and I/O depending on
    - Job complexity
    - Software architecture of the big data solution
    - Hardware performance and cluster architecture
- Cleanup phase
  - Teardown compute environment, free resources
  - Usually fixed costs (in the order of seconds)

Outline

1 Overview

2 Hardware
- Big Data Clusters
- HPC Clusters
- Software

3 Assessing Performance

4 Summary

Big Data Cluster Characteristics

University of Reading

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

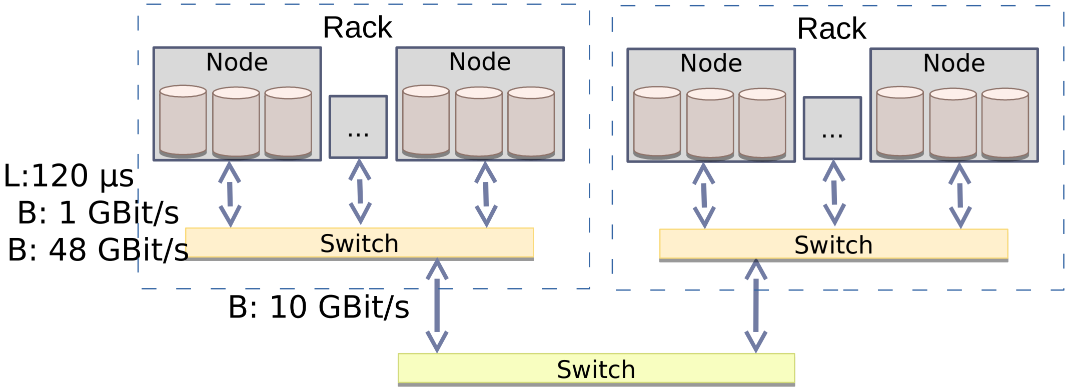


Figure: Architecture of a typical big data cluster

HPC Cluster Characteristics

University of Reading

- High-end components
- Extra fast interconnect, global/shared storage with dedicated servers
- Network provides high (near-full) bisection bandwidth. Various topologies are possible.

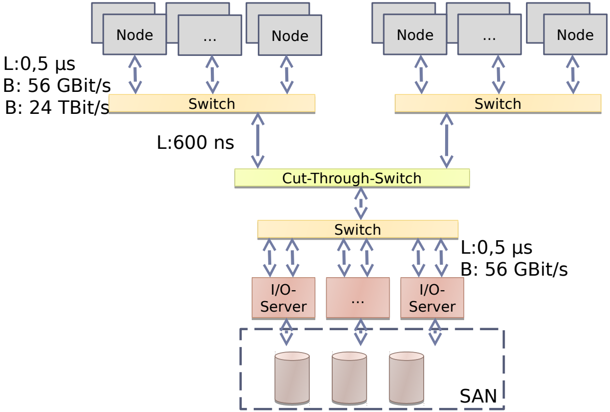


Figure: Architecture of a typical HPC cluster (here fat-tree network topology)

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

⇒ Access data consecutively and not randomly

⇒ Performance depends on the I/O granularity

- E.g.: 150 MiB/s with 10 MiB blocks

Hardware-Aware Strategies for Software Solutions

- Java is suboptimal: 1.2x - 2x of cycles needed than in C[1]
- Utilise different hardware components concurrently
  - Pipeline computation, I/O, and communication
  - At best hide two of them $\Rightarrow$ 3x speedup vs sequential
  - 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 (or process 10x data)
- Allow monitoring of components to see their utilisation
- Avoid I/O, if possible (keep data in memory)
- Avoid communication, if possible

Examples for exploiting locality in SQL/data-flow languages

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

Outline

1 Overview

2 Hardware

3 Assessing Performance
- Approach
- Errors Causing Recomputation
- Assessing Performance
- Assessing Compute and Storage Workflow

4 Summary

Basic Approach

Question

Is the observed performance acceptable?

Basic Approach

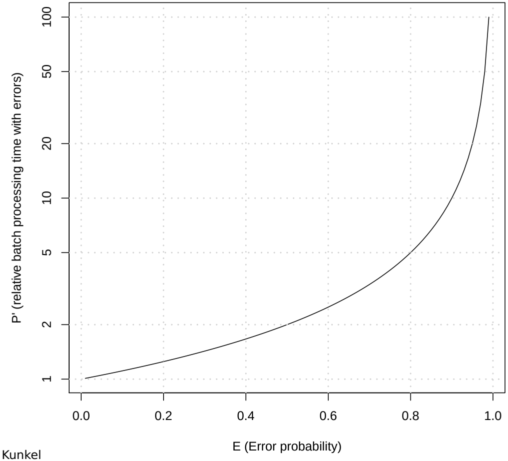
Start with a simple model

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 statistics output for each Hadoop job
3. Compute w_t, the workload you process per time
4. Compare w_t with your expectations of the system

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

Errors Increase Processing Time [11]

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



- With 50% chance of errors, twice the processing time
- With 90% chance, 10x

Outline

1 Overview

2 Hardware

3 Assessing Performance

4 Summary

Groupwork: Assessing Performance

Task: Aggregating 10Mi integers with 1 thread

- Vendor-reported performance from [14] indicates improvements

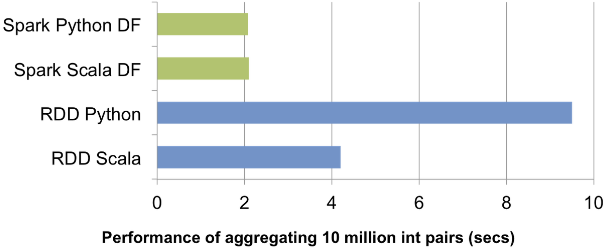


Figure: Source: Reference [14]

- These are the advancements when using Spark for the computation
- Can we trust in such numbers? Are these numbers good?
- Discuss these numbers with your neighbour (Time: 3 minutes)

Assessing Performance of In-Memory Computing

Measured performance numbers and theoretic considerations

- Spark [14]: 160 MB/s, 500 cycles per operation[2]
  - Invoking external programming languages is even more expensive!
- Python (raw): 0.44s = 727 MB/s, 123 cycles per operation
- Numpy: 0.014s = 22.8 GB/s, 4 cycles per operation (memory BW limit)
- One line to measure the performance in Python using Numpy:

```
timeit.timeit(stmt="np.sum(d)", setup="import numpy as np; d =
    ↪ np.array(range(1,10*1000*1000))", number=1)
```

- Hence, the big data solution is 125x slower in this example as anticipated!

Groupwork: Comparing Pig and Hive Big Data Solutions

University of Reading

Benchmark by IBM [16], similar to Apache Benchmark

- Tests several operations, data set increases 10x in size
  - Set 1: 772 KB; 2: 6.4 MB; 3: 63 MB; 4: 628 MB; 5: 6.2 GB; 6: 62 GB
- Five data/compute nodes, configured to run eight 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

	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: **Pig and Hive time**. Source: B. Jakobus (modified), "Table 2: Averaged performance" [16]

Assessing performance

- How could we model performance here?
- How would you judge the runtime here?
- Time: 2 minutes

Assessing Compute and Storage Workflow

- Daytona GraySort: Sort at least 100 TB data in files into an output file
  - Generates 500 TB of disk I/O and 200 TB of network I/O [12]
  - Drawback: Benchmark is not very compute intense
- Data record: 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: Reference [12]

Assessing Performance of In-Memory Computing

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

University of Reading

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 receiver
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

Summary

- Goal (user-perspective): Optimise the time-to-solution
- Runtime of queries/scripts is the main contributor
- Computation in big data clusters is usually over-dimensioned
- Understanding a few HW throughputs help to assess the performance
- Linear scalability of the architecture is the crucial performance factor
- Basic performance analysis
  1. Estimate the workload
  2. Compute the workload throughput per node
  3. Compare with hardware capabilities
- Error model predicts runtime if jobs must be restarted
- GreySort with Spark utilises I/O, communication is good
- Computation even with Spark is much slower than Python
- Different big data solutions exhibit different performance behaviours

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