Spark op Cartesius

Matthijs Moed Kees de Jong

April 1, 2019

1 Introductie

Het doel van dit project is de evaluatie van Spark op Cartesius op twee vlakken: 1.) performance, en 2.) functionaliteit.

1.1 Performance

Er zijn een aantal benchmarking-toolsets voor Spark, die de performance testen op het gebied van IO en rekenkracht. Naast de standaard toolsets simuleren we een onderzoeks-usecase met text mining op een deel van de Common Crawl-dataset.

De volgende benchmarks zullen meegenomen in de performance-evaluatie:

Naam	**Doel**
Spark TeraSort	IO
`spark-bench`	CPU performance (SparkPi) Large SQL queries Spark ML
CommonCrawl	Onderzoeks-usecase (mix van IO & CPU)

De invloed van complexiteit van de datasets zal geëvalueerd worden. Hierbij gaat het om de grootte van datasets, bijvoorbeeld sorteervolume van TeraSort en tabelgrootte voor `spark-bench` SQL-queries.

1.2 Tuning

Alle benchmarks zullen op zowel Hathi als Cartesius worden uitgevoerd. Door de verschillen in disk access, networkconfiguratie en node-setup zullen alle benchmarks getuned moeten worden voor de infrastructuur. Hierbij moet gelet worden op de volgende aspecten:

1. #CPU's/geheugen voor driver/worker/executor
2. #executors/workers per node
3. #nodes
4. Op Cartesius: striping

1.3 SparkRDMA

Naast Spark standalone op Cartesius, evalueren we ook het effect van RDMA op performance. Dit kan door middel van SparkRDMA.

1.4 Scaling

Afhankelijk van de workload wordt er een bepaald deel van de hardwarecapaciteit benut. Hierbij gaat het voornamelijk om efficiënt CPU-gebruik, maar ook geheugengebruik. Met Spark metric is het mogelijk om na te gaan hoe efficiënt capaciteit benut wordt.

1.5 Deliverables

Als uitkomst van de performance-evaluatie komt er een rapport met de volgende elementen:

1. Voor alle benchmarks:
   (a) Effect van het aantal executors op performance
   (b) Efficiëntie van executors wat betreft CPU- & geheugengebruik
2. TeraSort
   (a) Effect van datasetgrootte (100GB, 500GB, 1TB) op tijd
   (b) Effect van beschikbaar geheugen op tijd (testen op `thin` en `fat` nodes)
   (c) Effect SparkRDMA op tijd
3. `spark-bench` SparkPI: effect complexiteit (aantal getallen achter de komma van π) op tijd
4. `spark-bench` SparkML: effect grootte/complexiteit dataset op trainingstijd
5. `spark-bench` SQL: effect grootte tabellen en type operaties op querytijd
6. CommonCrawl
   (a) Effect grootte data op tijd
   (b) Effect SparkRDMA op tijd

1.6 Functionaliteit

Bepaalde Spark-functionaliteit moet makkelijk toegankelijk zijn voor gebruikers. Daarbij gaat het om het volgende:

1. Een basis-setup Spark 2.x werkend op Cartesius
2. Toegang tot Spark master UI
3. Toegang tot Spark driver UI
4. Toegang tot driver/executors logs
5. Een makkelijkere manier om applicaties te submitten (zonder direct SLURM aan te sturen), en ad-hoc Spark clusters af te breken
6. Mogelijkheid tot gebruik Spark in Jupyter Notebooks

1.6.1 Mogelijke oplossingen

1. Custom oplossing met wat scripting
2. Spark op SLURM met sparkhpc

1.7 Deliverables

1. Basis-setup van Spark 2.x op Cartesius
2. Mogelijkheden inventariseren voor toegang UIs
3. Mogelijkheden inventariseren voor toegang tot driver/executor logs
4. Plan maken voor makkelijker opzetten ad-hoc Spark-clusters op Cartesius
5. Evaluatie van integratie Jupyter Notebooks met Spark

1.8 Benchmark results

1.8.1 SparkPi

SparkPi is een workload in `spark-bench`, en voert een numerieke approximatie uit van π. Deze workload dient als CPU-benchmark, en stuurt geen data over het netwerk of naar Lustre. Het aantal *slices* is configureerbaar. Elke slice verwijst naar één punt dat gebruikt wordt in de approximatie.
De volgende parameters/settings zijn gebruikt voor deze benchmark:

- #slices: 5.000.000
- #executors: 163
- #cores/executor: 4
- geheugen/executor: 18GB
- #nodes: 54, thin, allen op hetzelfde eiland

De totale wall-clocktijd van deze benchmark was **20 minuten en 15 seconden**. Dit is onverwacht lang voor het uitrekenen van vijf miljoen coördinaten. SparkPi blijkt voor elke slice een taak uit te sturen over het netwerk. Deze taken worden door de driver verstuurd naar de executors, waarbij executors hun resultaten terugsturen naar de driver. Elke taak moet gepland worden door de scheduler van de driver. Het aandeel van deze scheduling in de totale rekentijd van een taak neemt proportioneel toe met het aantal executors waarmee de driver communiceert.

Een steekproef middels de Spark web UI wees uit dat de daadwerkelijke rekentijd minder dan 1 milliseconde bedraagt, evenals de serialisatie- en deserialisatietijd van een taak. Echter, de scheduling-tijd van een taak bedroeg bij 2 nodes (6 executors) 8 milliseconden, en bij 54 nodes (163 executors) 80 milliseconden. Vrijwel alle tijd wordt dus ingenomen door het schedulen van taken. Om deze reden is de SparkPi-benchmark niet geschikt om een goede inschatting te maken van CPU-performance.

1.8.2 k-means

k-means is een benchmark in `spark-bench`, en voert k-means clustering uit op een gegenereerde dataset. Deze workload dient als een gecombineerde CPU- en I/O-workload waar data sequentieel en in grote hoeveelheden wordt ingelezen en verwerkt. De input- en output-data zijn een Spark DataFrame.
De parameters/settings voor deze benchmark zijn als volgt:

- Hoeveelheid input data: 181G (Parquet-formaat)
- #rijen: 1.000.000.000
- #kolommen: 24
- #clusters: 500
- Scaling factor: 1.6
- #partities: 50

https://www.overleaf.com/project/5bf53171346de316634ea033

Met deze settings is het aantal executors en de stripe count van de input en output data gevarieerd. De resultaten zijn te vinden in Tabel 1.8.2.

#executors	Stripe count	Wall-clocktijd
163	1	9m8
163	1	9m1
163	-1	10m23
76	-1	13m45

Table 1: Wall-clocktijden voor k-means clustering.

1.9 TeraSort

TeraSort is een gecombineerde CPU- en I/O-benchmark die een gegenereerde dataset sorteert. De voornaamste component in deze benchmark is de Lustre-I/O. TeraSort deelt de inputdata op in kleine (MB-range) files, waar tienduizenden kleine (kB-range), random reads in worden gedaan. Aangezien Lustre geöptimaliseerd is voor het uitvoeren van grote, sequentiële reads dient deze benchmark vooral als I/O-benchmark.

De benchmark is uitgevoerd met de volgende parameters:

- Hoeveelheid data: 100GB

Met deze settings is het aantal executors en de stripe count van de input en output data gevarieerd. De resultaten zijn te vinden in Tabel 1.9.

#executors	**Stripe count**	**Wall-clocktijd**
15	-1	18m
15	1	16m
15 (Hadoop)*	N/A	16m

Table 2: Wall-clocktijden voor TeraSort.

* = 15 executors van 9GB op Hathi Hadoop cluster.

1.9.1 TODOs

- Onderzoek effect striping op TeraSort shuffle-performance
- Real-life use case van SQL processing.joins
- CommonCrawl-voorbeeld
- Effect #executors op wall-clocktijd k-means