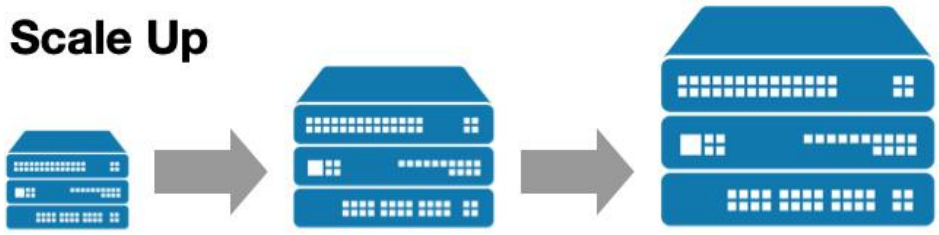
3. laboratorijska vježba

Sadržaj

Spark

Prema autorima softvera, Apache Spark, ili kako ćemo ga u nastavku zvati samo Spark, je "višejezični analitički motor za obradu velikih količina podataka". Jedna kratka i točna, ali suhoparna definicija. Kako biste si lakše dočarali Spark, shvatite ga kao tvornicu podataka - sirovina, odnosno podatci - su ulazi, a modeli, razni uvidi, vizualizacije i slično su rezultati - izlazi. Kao što tvornica često povećava kapacitet povećanjem prostora, Spark može obraditi sve veću količinu podataka proširivanjem (scaling out - horizontalno skaliranje: preko više manjih računala - Slika 1) umjesto nadograđivanjem (scaling up - vertikalno skaliranje: dodavanje više resursa poput CPU-a, RAM-a i prostora na disku na jedno računalo - Slika 2). Umjesto uobičajenog dobivanja veće vrijednosti za više novca prilikom kupnje veće količine istog artikla RAM zapravo košta više kada se kupuje veća količina za jedan artikl - tako je 128 GB RAM-a skuplje od dva puta po 64 GB RAM-a. Odnosno, umjesto da potrošite tisuće eura kako biste smjestili skup podataka, oslanjat ćete se na više računala dijeleći pritom zadatke među njima.

Slika 1



Slika 2

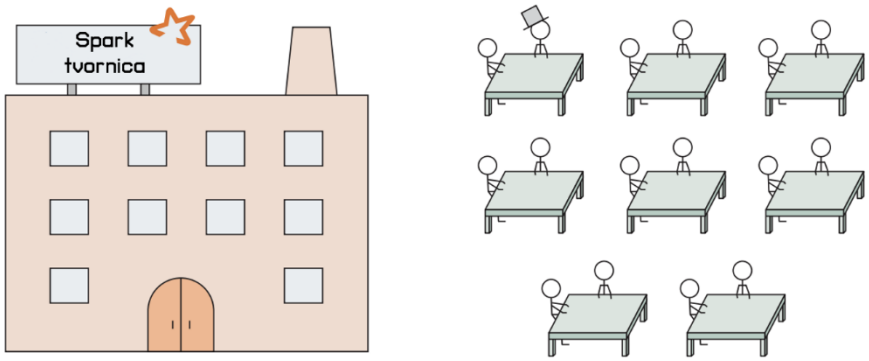
Jedno računalo može ponekad pasti ili se ponašati nepredvidljivo. Ako umjesto jednog imate stotinu računala, vjerojatnost da će barem jedno od njih otkazati je mnogo veća. Zbog toga Spark ima puno karika za upravljanje, skaliranje i nadgledanje kako biste se mogli usredotočiti na ono što želite - rad s podatcima. Upravo je to jedna od ključnih stvari o Sparku: dobar je alat zbog onoga što s njim možete uraditi, ali posebno zbog

onoga što s njim ne morate raditi. Spark pruža moćni API s kojim vam se čini kao da radite s kohezivnim izvorom podataka dok u pozadini naporno radi na optimiziranju vašeg programa kako bi iskoristio svu dostupnu snagu. Ne morate biti stručnjak za distribuirano računarstvo; samo trebate biti upoznati s jezikom koji ćete koristiti za izgradnju svog programa.

PySpark = Python + Spark

PySpark pruža ulaznu točku za Python u računalnom modelu Sparka. Dok je sam Spark napisan u Scali, autori su se potrudili pružiti koherentno sučelje između jezika, istovremeno održavajući specifičnosti svakog jezika gdje je prikladno. Stoga će biti poprilično jednostavno programeru koji radi sa Sparkom u Scali pročitati vaš PySpark program, kao i za nekom drugom Python programeru koji još nije uvelike upoznat sa Sparkom.

Objasnimo ukratko kako Spark obrađuje program. Važno je imati znanje o tome kako se Spark postavlja, kako upravlja podatcima i kako optimizira upite - ako to razumijete moći ćete manipulirati sustavom, poboljšavati svoj kod i brzo shvaćati kada nešto ne radi onako kako želite. Ako zadržimo analogiju tvornice, možete zamisliti da klaster računala na kojima Spark radi predstavlja građevinu. Ako pogledate sliku 3, vidjet ćete dvije različite interpretacije tvornice podataka. S lijeve strane vidimo kako ona izgleda izvana: jedinica u koju ulaze razni projekti, a izlaze rezultati - kao takva će se većinu vremena i vama činiti. Međutim, iznutra izgleda više kao desna slika: skup radnih klupa kojima su dodijeljeni zaposlenici.



Slika 3

Spark klaster

Tvornica započinje s radom po prijemu zadatka koji se u svijetu Sparka naziva *driver program* (ili kraće samo driver). Zamislite driver kao voditelja kata - vi mu dajete popis koraka i prepuštate ga da se pozabavi time. To ne znači da se odmah počinje s procesiranjem - prije toga, klaster treba isplanirati kapacitet koji će alocirati za program. Driver je odgovoran za kompletiranje zadanog zadatka te zahtijeva resurse prema potrebi.

Entitet ili program koji se brine za to se zove upravitelj klastera (engl. *cluster manager*). U našoj tvornici upravitelj klastera pregledava radne stolove s raspoloživim prostorom i osigurava onoliko prostora koliko je potrebno, a zatim upošljava zaposlenike kako bi popunio kapacitet. U Sparku upravitelj pregledava strojeve s raspoloživim računalnim resursima i osigurava što je potrebno prije pokretanja potrebnog broja izvršitelja nad njima. Bilo kakve upute o kapacitetu (računala i izvršitelji) kodiraju se u *Spark-Contextu* koji predstavlja vezu s našim Spark klasterom. Ako upute ne spominju određeni kapacitet, upravitelj klastera će alocirati zadani kapacitet koju propisuje Spark instalacija.

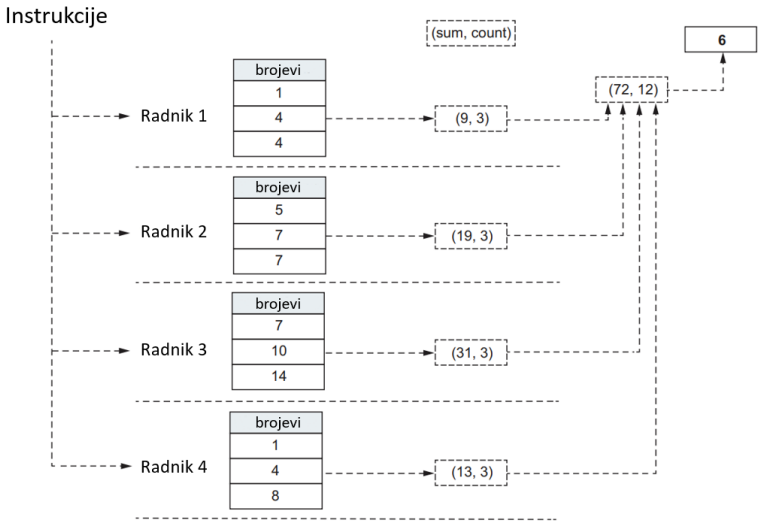
Radne klupe su poput računala u Sparkovom klasteru: njihov broj je određen. Zaposlenici se u Spark literaturi nazivaju izvršiteljima (engl. *executors*): oni obavljaju stvarni rad na strojevima/čvorovima - skup računalnih/memorijskih resursa se u Sparku naziva *worker nodes* (ili kraće samo workers). Driver/voditelj kata uzima vaše instrukcije u kodu i prevodi ih u Spark korake te ih zatim obrađuje preko *workera*. Driver također određuje koji worker/klupa ima koji dio podataka i osigurava da pritom ne izgubite neke važne dijelove.

Možete uočiti kako se jedan radnik izdvaja iz mase radnika - u tvornici podatka on je upravitelj radnog područja te raspodjeljuje resurse prema potrebi kako bi se dovršili zadatci. U Sparkovim terminima ga nazivamo glavnim čvorom (engl. master). Master sjedi na jednoj od radnih klupa/strojeva, ali može sjediti i na odvojenom stroju ovisno o upravitelju klastera i načinu deploymenta.

Kao primjer uzmimo CSV datoteku čiji je sadržaj ispisan u prozoru ispod - nad njom ćemo izvršiti zadatak izračuna aritmetičke sredine iz vrijednosti sadržanih u stupcu brojevi. Pretpostavimo da naš primjerak Sparka ima četiri izvršitelja i da svaki radi na svom radnom čvoru. Obrada podataka će biti podjednako raspodijeljena između četiri izvršitelja: svaki izvršitelj će raditi s malim dijelom DataFramea kojim će upravljati.

```
brojevi
1
4
4
5
7
7
7
10
14
1
4
8
```

Slika 4 prikazuje jedan od načina na koji PySpark može procesirati prosjek iz stupca brojevi u našem malom DataFrameu. U konkretnom slučaju izračuna srednje vrijednosti svaki radnik nezavisno računa zbroj i brojnost prije nego što proslijedi rezultat (ne sve podatke) jednom radniku koji će isprocesirati agregaciju u jedan broj - srednju vrijednost.



Svaki radnik ima uzorak podataka i obavlja međukorak kako bi dobio zbroj i brojnost svake particije DataFramea.

Podatci iz prethodnog koraka se prosljeđuju na daljnju redukciju jednom radniku.

Na kraju dobivamo željeni odgovor. Spark učinkovito skriva složenost efikasnog distribuiranja računanja preko čvorova.

Slika 4

Lijena evaluacija u Sparku

Kroz ovaj odjeljak ćete upoznati jedan od osnovnih aspekata Sparka: njegove mogućnosti lijene evaluacije. Svaka instrukcija koju predate Sparku može se klasificirati u dvije kategorije: akcije i transformacije. Akcije (actions) u Sparku predstavljaju ono što se u drugim programskim jezicima smatra za I/O. Najčešće akcije uključuju:

- show - ispis informacija na zaslon
- write - zapisivanje i pohrana podataka
- count - broj zapisa

Transofrmacije (transformations) u Sparku su više manje sve ostalo - neki primjeri uključuju:

- dodavanje stupca tablici
- izvođenje agregacije prema određenim ključevima
- izračunavanje sažetka statistike
- treniranje modela strojnog učenja

Mogli biste se zapitati zašto se radi distinkcija. Kada razmišljate o komputaciji podataka kao programer jedino vas zanima komputacija koja vodi k nekoj akciji - komunicirate s rezultatima akcije jer je to nešto što želite vidjeti. Spark sa svojim modelom lijenog izračuna će ovo odvesti u ekstreme i izbjeći izvođenje rada s podatcima sve dok akcija ne pokrene računski lanac. Prije toga, driver program će spremiti vaše instrukcije.

Ovakav način računanja ima brojne benefite kada radite s velikim podatcima.

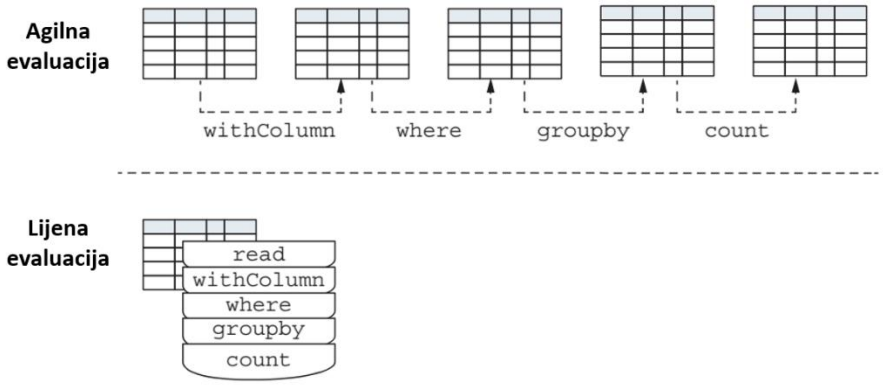
Prvo, pohranjihvanje instrukcija u memoriji zauzima puno manje prostora od spremanja međurezultata. Ako na skupu podataka izvodite veliki broj operacija i materijalizirate podatke svaki korak vrlo brzo ćete potrošiti svoj prostor za pohranu, iako vam ti međurezultati zapravo ni ne trebaju.

Drugo, posjedujući cijelu listu zadataka koje treba obaviti driver program može optimizirati rad među izvršiteljima puno učinkovitije. Može koristiti informacije tijekom izvođenja programa - kao što su na kojim čvorovima su pohranjeni koji podatci. Također, može promijeniti redoslijed, eliminirati nepotrebne transformacije, kombinirati više operacija i učinkovitije preraditi dio programa ako je potrebno.

Treće, ako jedan čvor zakaže tijekom računanja, Spark može rekreirati izgubljene podatke budući da ima predmemorirane instrukcije. Pročitat će relevantne podatke i procesirati ih do određenog dijela bez potrebe za vašom intervencijom. Tako se možete

usredotočiti na aspekt obrade podataka u vašem kodu prebacujući breme oporavka na Spark.

Lijeno računanje je temeljni aspekt Sparkovog operativnog modela i dio razloga zašto je tako brz. Većina programskih jezika, uključujući Python, R i Javu, ima agilnu evaluaciju. To znači da obrađuju instrukcije čim ih dobiju. S PySparkom tako koristite agilni jezik - Python, s lijenim frameworkom - Sparkom. Jedan aspekt koji trebate imati na umu je da Spark neće sačuvati rezultate akcija za sljedeća računanja. Ako podnesete isti program dvaput, PySpark će procesirati podatke dvaput.



Slika 5

Sažetak

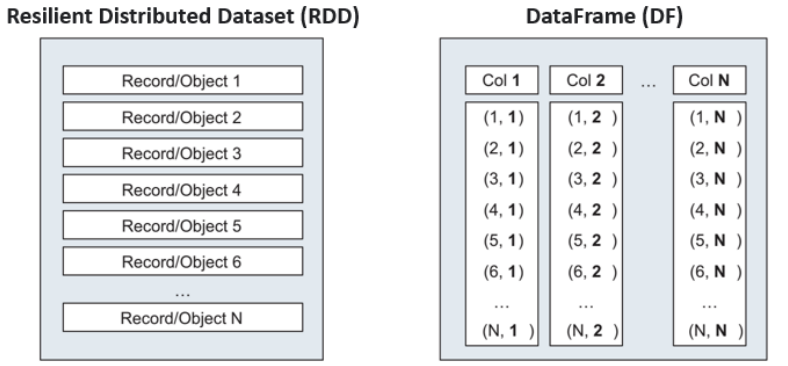
Što je upravitelj bez kompetentnih zaposlenika? Jednom kada je zaprimljen zadatak zajedno sa svojom akcijom driver počinje dodjeljivati podatke Sparkovim executorima. Executori su procesi koji izvode računanja i spremaju podatke za aplikaciju. Oni su smješteni na *worker nodeovima* - stvarnim računalima. U analogiji tvornice executor je zaposlenik koji obavlja zadatak, dok je worker node klupa gdje zaposlenici mogu raditi. Time smo završili obilazak tvornice. Rezimirajmo još jedan tipičan PySpark program:

- Prvo kodiramo svoje instrukcije u Pythonu stvarajući pritom driver program
- Kada podnesemo naš program cluster manager za nas dodjeljuje resurse za upotrebu koji će uglavnom ostati konstantni (osim kada se koristi autoskaliranje) za vrijeme izvođenja programa
- Driver uzima vaš kod i prevodi ga u Spark instrukcije. Te instrukcije mogu biti akcije ili transformacije.
- Jednom kada driver pokrene akciju optimizira cijeli lanac računanja i dijeli rad među executorima

- Executori su procesi obavljanja stvarnog rada s podatcima te se nalaze na strojevima zvanima worker nodeovi.

Strukture podataka u Sparku

Dvije glavne strukture podataka u Sparku su Resilient Distributed Dataset (RDD) i DataFrame (DF). RDD je bila jedina struktura dugo vremena. Izgleda kao distribuirana kolekcija objekata (redova). DataFrame je stroža verzija RDD-a. DF se oslanja na koncept stupaca gdje operirate na stupcima umjesto zapisima kao što je slučaj u RDD-u. Na slici 1.6. možete vidjeti vizualni sažetak te dvije strukture. DF je trenutno dominantna struktura podataka tako da će na njoj biti naglasak u nastavku vježbe. Implementacija DataFramea je slična radu u SQL-u, pa se tako čak i modul za organizaciju i manipulaciju podatcima naziva pyspark.sql.



U RDD-u svaki zapis je neovisan object te se transformacije izvode na razini zapisa. Naglasak na kolekciji, a ne na strukturi.

DF organizira zapise u stupce. Transformacije se obavljaju direktno nad tim stupcima ili na cijelom DF-u. Obično se ne pristupa zapisima horizontalno (zapis po zapis) kao u RDD-u.

Instalacija za Windows

1. Preuzmite Spark sa sljedeće poveznice https://spark.apache.org/downloads.html.
2. Dodajte raspakiranu mapu u varijable okruženja (environment variables) pod SPARK_HOME, a mapu bin u Path.
3. Preuzmite Javu - provjerite dokumentaciju za svoju verziju Sparka te sukladno tome instalirajte odgovarajuću verziju Jave (dokumentacija za Spark 3.3.2 https://spark.apache.org/docs/3.3.2/)
4. Dodajte jdk mapu u varijable okruženja pod JAVA_HOME, a mapu bin u Path.

5. Preuzmite Hadoopov bin sa sljedeće poveznice - https://github.com/cdarlint/winutils/tree/master/hadoop-3.2.2/bin
6. Mapu iznad bina dodajte u varijable okruženja pod HADOOP_HOME, a mapu bin u Path.

Povezivanje PySparka s Kafkom

Vaš prvi zadatak je dohvatiti svoje podatke s Kafke uz pomoć Sparka koristeći naredbu spark-submit u naredbenom retku - kako biste to ostvarili pronađite "Structured Streaming + Kafka Integration Guide" za svoju verziju Sparka. Svoje rezultate spremite kao CSV datoteku. Prije obavljanja ovog zadatka proučite SparkSession i DataFrame poglavlja iz Jupyter bilježnice.

Jupyter bilježnica

U nastavku vam je priložena interaktivna Jupyter bilježnica - vaš zadatak je upoznati se kroz riješene primjere i kratke zadatke sa Sparkom. Obavite još posljednje korake prije početka rada u Jupyter bilježnici:

1. Ako nemate instaliran Jupyter pokrenite:

   ```
   pip install jupyter
   ```

2. Podesite sljedeće varijable okruženja:

   PYSPARK_DRIVER_PYTHON=jupyter i
   PYSPARK_DRIVER_PYTHON_OPTS=notebook

   kako bi Jupyter Notebook mogao pronaći vaš Spark naredbom:

   ```
   findspark.init()
   ```

3. Instalirajte sljedeće Python biblioteke:

   ```
   pip install pyspark
   pip install findspark
   ```

4. Pozicionirajte se u mapi u kojoj se nalazi preuzeta bilježnica te pokrenite:

   ```
   jupyter notebook
   ```

Sve akcije i tranformacije kojima možete doći do rješenja u zadatcima su već navedene u uvodnim primjerima. Slobodni ste koristiti se i drugim pristupima, ali imajte na umu da sve što vam je potrebno je već navedeno u bilježnici. Svoj program za dohvaćanje podataka s Kafke uz pomoć Sparka i riješenu Jupyter bilježnicu objavite na svoj GitHub repozitorij.