MapReduce/Spark Projekt

Tento projekt předpokládá znalosti a dovednosti získané na cvičení.

Příprava prostředí Hadoop

POZOR na Microsoft Windows: V současné verzi je zapotřebí zkoušet všechny Gradle projekty na GNU/Linux s JDK alespoň verze 8 (OpenJDK nebo Oracle). Spuštění na Windows se pravděpodobně nepovede v případě Hadoop projektů (MapReduce a Pig) z důvodu chybějící/chybné podpory pro nativní operace Hadoopu. Pokud se vyskytne problém, zkuste aktualizovat projekt opětovným stažením z tohoto webu (bude zde vždy nejnovější verze). Omlouvám se za komplikace způsobené nekompatibilitou s Microsoft Windows a technické potíže na učebně.

Gradle Projekty

Stáhněte si Gradle projekty pro

- MapReduce WordCount
- Spark WordCount
- Pig Executor
- Spark PeopleInYear

Čtěte přiložené soubory `README.md` s informacemi o použití/spuštění projektů.

Virtuální stroj s Hadoop

Pro tento projekt používejte NixOS Distributed Applications Virtual Machine. Virtuální stroj zprovozněte, spusťte a prozkoumejte (alternativně, což nedoporučuji, můžete použít Cloudera QuickStart VM).

První část -- Pig/Grunt a MapReduce aplikace

Příprava vstupních dat

1. Stáhnětě si soubor Fielding.csv se statistikami hráčů baseballu do roku 2021 ve formátu comma-separated values (CSV) publikovaný Chadwick Baseball Bureau.

2. Prostudujte obsah souboru, kde je na prvním řádku hlavička a následují řádky s daty. Seznamte se s významem jednotlivých sloupců, tedy: playerID, yearID, stint, teamID, lgID, POS, G, GS, InnOuts, PO, A, E, DP, PB, WP, SB, CS, ZR.

Předpřipravený Pig/Grunt skript

Prostudujte následující Pig skript:

```
-- Load the data in comma-separated-values format with a given schema
fielding =
    LOAD '$INPUT1'
    USING org.apache.pig.piggybank.storage.CSVExcelStorage
        (',', 'NO_MULTILINE', 'NOCHANGE', 'SKIP_INPUT_HEADER')
    AS (
        playerID : chararray, yearID : int, stint : int,
        teamID : chararray, lgID : chararray,
        POS : chararray, G : int, GS : int,
        InnOuts : int,
        PO : int, A : int, E : int, DP : int,
        PB : int, WP : int, SB : int, CS : int,
        ZR : double
    );

-- Filter the previously loaded data
fielding95 = FILTER fielding BY yearID >= 1995;

-- Arrange the data into groups according to given column(s)
groupedByTeam = GROUP fielding95 BY (playerID, teamID);

-- Apply an aggregate function on each group (flatten multicolumn group to n-
tuple)
results = FOREACH groupedByTeam GENERATE FLATTEN(group) AS (playerID, teamID),
    AVG(fielding95.G) AS avgGames, SUM(fielding95.G) AS sumGames;

-- Filter the resulting data
results10ge = FILTER results BY avgGames >= 10;

-- Execute all the statements and ...
-- show output
--DUMP results10ge;
-- store the output into the output directory
STORE results10ge INTO '$OUTPUT'
    USING org.apache.pig.piggybank.storage.CSVExcelStorage
        (',', 'NO_MULTILINE', 'NOCHANGE', 'WRITE_OUTPUT_HEADER');
```

Výše uvedený Pig script naleznete také v Gradle projektu Pig Executor jako `sample-scripts/fielding.pig`. Pro načítání CSV souboru využívá Pig script třídu org.apache.pig.piggybank.storage.CSVExcelStorage z knihovny PiggyBank.

Spuštění na lokálním stroji i na virtuálním clusteru

1. Podobně, jako jste si vyzkoušeli na cvičení, spusťte Pig script na lokálním stroji (pomocí Gradle projektu Pig Executor) i na virtuálním clusteru. Nezapomeňte, že parametry při spuštění na clusteru odkazují na vstupní soubory a výstupní adresář v uživatelském adresáři na HDFS, nikoliv v lokálním adresářovém systému virtuálního stroje.
2. Prozkoumejte výsledky a log spuštění na virtuálním stroji a porovnejte je s výsledky a logem z předchozího spuštění na lokálním stroji.

Tvorba MapReduce aplikace

1. Navrhněte MapReduce úlohu, která načte data ze souboru `Fielding.csv` a zapíše do adresáře `fielding.mr_output` stejné výsledky, jako vypočítal předchozí Pig/Grunt skript. Formát zapsaných dat úlohou MapReduce se může mírně lišit od formátu výstupu Pig skriptu (např. MapReduce úloha bude, narozdíl od Pig skriptu, vracet data setříděná podle klíče). Cestu ke vstupnímu soboru a výstupnímu adresáři umožněte, v ideálním případě, zadat jako parametry spouštěné MapReduce úlohy z příkazového řádku (např. parametry příkazu `hadoop jar ...` při spuštění na serveru).
2. Navrženou MapReduce úlohu implementujte v programovacím jazyce Java nad Apache Hadoop pomocí implemenace/rozšíření tříd Mapper (pro Mapper) a Reducer (pro Reducer, případně také pro Combiner). Implementujte také třídu se statickou metodou `main`, která nakonfiguruje a odešle na Hadoop úlohu (Job) obsahující vaše implementace tříd implementujících Mapper a Reducer, případně také Combiner.
3. Aplikaci přeložte, spusťte na lokálním stroji i na virtuálním serveru (pomocí SCP/SSH) a důkladně otestujte.

Jako inspiraci můžete využít předpřipravený MapReduce WordCount Gradle projekt.

Nápověda k řešení

1. Mezi úlohami Map a Reduce (případně Combiner) budete potřebovat předávat více hodnot, než jen prostý klíč a hodnota. Např. klíč bude složený z názvu (identifikátoru) hráče a týmu. Pro předávání složených hodnot (ať už jako klíč či hodnota funkcí MapReduce) můžete jednotlivé složky spojit do textového řetězce a oddělit vhodným oddělovačem (datový typ/třida také výsledné hodnoty pak bude "Text"). Rozdělit výsledný textový řeězec pak můžete v následující úloze MapReduce pomocí `_promena_.toString().split(",")`, pokud by byl objekt typu "Text" uložen v proměnné "promena" a hodnoty odděleny čárkou.
    - V praxi je pro složené hodnoty vhodnější zadefinovat vlastní třídu implementující "org.apache.hadoop.io.Writable", např. jak je uvedeno v příkladu třídy MinMaxCountTuple.
2. Rozhraní a průběh jednolivých funkcí MapReduce (vč. Combiner) mohou napovědět komentáře ze vzorového řešení:

```
// *** Mapper.map: (object, line_content) -> [("player,team", "games,1")]
// Convert input line of data (a record) to string for processing
// Split up the record by commas to individual elements (record columns'
values)
// For valid input...
    // Get value from the year column
    // For given years
    // Get values from player and team columns
        // Get number of games of the player in the team for a
corresponding column
        // Set output key to "player,team"
        // Set ouput value to "games,1"
        // (that is number of games in the one, just processed, record)
        // Emit the key and the value

// *** Combiner.reduce: ("player,team", ["games,1", ...]) ->
[("player,team", "sum_of_games,count_of_records")]
    // Go through number of games for the player and team at the input, and
sum up and count them
        // Split up each value by commas
        // Increase total sum of number of games and total count of their
records
        // by individual sum and count from the value
    // Set value to "totalSumGames,totalCountRecords"
    // Emit the original key and the set value

// *** Reducer.reduce: ("player,team", ["sum_of_games,count_of_records",
...]) -> [("player,team", "avg_of_games_per_record,sum_of_games")]
    // Go through number of games for the player and team at the input, and
sum up and count them
        // Split up each value by commas
        // Increase total sum of number of games and total count of their
records
        // by individual sum and count from the value
    // Compute average of number of games per record of the player-team
pair at the input
    // (if values of "number of games" and "count of their records" are
integers,
    // they have to be type-casted to double to keep double precision in
the result)
    // Output "avg_of_games_per_record,sum_of_games" of games iff avg is
equal or greater than 10 games

// *** main
    // Configuration
    // (optional: comma-separated values (CSV) output format)
    // Args processing
    // Job definition (including mapper, reducer, combiner)
    // Input/Output settings
    // Execute the job
```

Druhá část -- Pig/Grunt a Spark aplikace

Příprava vstupních dat

1. Stáhnětě si aktuální data historie měnových kurzů European Central Bank (v poměru k EUR) ve formátu XML přímo ze stránek ECB (pozor, soubor je velikosti nad 15 MB).
2. Dále si stáhněte shell skript pro převod výše uvedených XML dat do formátu Comma Separated Value (CSV). Shell skript i XML soubor s daty umístěte do jednoho adresáře a spusťte na unix/linux stroji, přičemž výstup přesměrujte do souboru "eurofxref-sdmx.csv", tj. `./eurofxref-sdmx-xml-to-csv.sh >eurofxref-sdmx.csv` (např. k datu 29.10.2021 bude výstup tento CSV soubor; ten však v projektu nepoužívejte, vygenerujte si vlastní z aktuálních dat).
3. Prostudujte obsah souboru "eurofxref-sdmx.csv", kde na prvním řádku jsou názvy sloupců (pro další zpracování bude možná potřeba tento první řídek odstranit) a na dalších řídcích záznamy. Jednotlivé, čárkou oddělené, položky záznamů mají význam:
   - ISO 4217 kód měny,
   - datum ve formátu rok-měsíc-den,
   - kurz dané měny vůči měně EUR (desetinné číslo s jednotkami a desetinami oddělenými tečkou, tj. anglický formát). Podrobnější popis můžete nalézt na stránkách datového skladu ECB.

Tvorba Pig/Grunt skriptu

1. Vytvořte Pig/Grunt skript (podobný, jako v předchozí části projektu), který načte data ze souboru "eurofxref-sdmx.csv" a spočítá pro ně následující statistické údaje:
   - data a hodnoty nejmenšího a největšího kurzu pro každou z měn v souboru, hodnotu průměrného kurzu za celé sledované období pro každou z měn v souboru,
   - hodnoty nejmenšího, největšího a průměrného kurzu za poslední rok (tj. od stejného data, jako je datum stažení souboru, loňského roku) a za poslední měsíc (tj. od stejného data, jako je datum stažení souboru, předchozího měsíce) pro měny CHF, GBP a USD v souboru,
   - hodnoty průměrného kruzu za každý měsíc v celém sledovaném období pro každou z měn v souboru (tedy od prvního do posledního dne měsíce, či do posledního dne soboru pro poslední uvedený měsíc),
   - názvy měn s největším a nejměnším kruzem z celého souboru.
2. Pokuste se vytvořit jeden skript, vracející uvedené výsledky. Nebude-li to možno, vytvořte několik skriptů, pro jednotlivé výsledky či jejich skupiny.
3. Spusťte skript (či skripty) nejprve na lokálním, a poté na virtuálním stroji na clusteru, nad daty ze souboru "eurofxref-sdmx.csv", prozkoumejte a uschovejte jejich výsledky (budete odevzdávat skript/y, jejich výsledky a logy standardního i chybového výstupu při spuštění jak na lokálním stroji, tak na virtuálním clusteru).

Tvorba Spark aplikace

1. Navrhněte Spark aplikaci, která načte data ze souboru "eurofxref-sdmx.csv" a spočítá výsledky stejných úloh, jako předchozí Pig/Grun skript/y této části projektu (stačí jen dvě z uvedených odrážek).
2. Spusťte vytvořenou Spark aplikaci (či aplikace) nejprve na lokálním, a poté na virtuálním stroji na clusteru, nad daty ze souboru "eurofxref-sdmx.csv", prozkoumejte a uschovejte jejich výsledky (budete odevzdávat zdrojové soubory aplikace, její výsledky a logy standardního i chybového výstupu při spuštění jak na lokálním stroji, tak na virtuálním clusteru).

Jako inspiraci můžete využít předpřipravené Spark WordCount a Spark PeopleInYear Gradle projekty.

Odevzdávané výsledky

Do WISu se odevzdává jeden či několik ZIP archivů (maximální velikost jednotlivého souboru je 2 MB) k termínu projektu s následujícími soubory:

1. část projektu:
   - `1_popis.txt` -- stručný textový popis výsledku Pig/Grunt skriptu a zároveň vaší MapReduce aplikace (popis, co představují výstupní data ze skriptu/aplikace). Prostý textový soubor v UTF-8 nebo ASCII.
   - `1_output_pig.txt` -- soubor s výsledky výpočtu Pig/Grunt skriptu (např. kopie `/tmp/fielding.pig_output/part-r-00000` při lokálním spuštění).
   - `1_log_pig_local.txt` -- část výstupu, který vypisuje Pig na standardní chybový výstup (stderr, log) při lokálním spuštění Pig/Grunt skriptu (např. začíná zprávou `... mapreduce.SimplePigStats ... Script Statistics`).
   - `1_log_pig_cluster.txt` -- část výstupu, který vypisuje Pig na standardní chybový výstup (stderr, log) při spuštění Pig/Grunt skriptu na virtuálním clusteru (např. začíná zprávou `... [main] INFO org.apache.pig.Main - Apache Pig version ...`).
   - `1_output_mr.txt` -- soubor s výsledky výpočtu vaší MapReduce aplikace (např. kopie `/tmp/fielding.mr_output/part-r-00000`).
   - `1_log_mr_local.txt` -- celý výstup, který vypisuje Hadoop na standardní chybový výstup (stderr, log) při lokálním spuštění vaší MapReduce aplikace.
   - `1_log_mr_cluster.txt` -- celý výstup, který vypisuje Hadoop na standardní chybový výstup (stderr, log) při spuštění vaší MapReduce aplikace na virtuálním clusteru.
   - `1_app` -- adresář se zdrojovými kódy vaší MapReduce aplikace (vč. Gradle projektových souborů, ale bez binárek, tj. vyčištěný pomocí `clean` skriptu a bez podadresářů `.gradle` a `hadoop.win`).

2. část projektu:

- `2_eurofxref-sdmx.txt` -- soubor s datem stažení souboru "eurofxref-sdmx.xml" ve formátu rok-měsíc-den (např. 2020-11-04).
- `2_pig` -- adresář se soubory Pig/Grunt skriptů a jejich výstupů (pokud budete mít více Pig skriptů, k jednotlivým úlohám, tak tyto srozumitelně očíslujte).
- `2_app` -- adresář se zdrojovými kódy vaší Spark aplikace a jejích výstupů (vč. Gradle projektů). (vč. Gradle projektových souborů, ale bez binárek, tj. vyčištěný pomocí `clean` skriptu a bez podadresáře `.gradle`).
- `2_log_local.txt` -- celý výstup, který vypisuje Spark na standardní chybový výstup (stderr, log) při lokálním spuštění vaší Spark aplikace.
- `2_log_cluster.txt` -- celý výstup, který vypisuje Spark na standardní chybový výstup (stderr, log) při spuštění vaší Spark aplikace na virtuálním clusteru.

Studijní literatura/zdroje

1. Apache Hadoop documentation
2. Apache Pig documentation
3. Apache Spark documentation
4. COGNITIVE CLASS: Hadoop 101
5. Hortonworks: How to Process Data with Apache Pig
6. In-mapper Combiner Program to Calculate Average