Hadoop

Maksim Norkin
maksim.norkin@ieee.org

I. Įžanga

II. Istorija

Hadoop projektas prasidėjo nuo Google. Jie norėjo atsisiųsti visą internetą ir atlikti tam tikrus skaičiavimus su gautais duomenimis. Tam reikėjo turėti labai brangią serverinę įranga ir jie neturėjo finansinių šaltinių tokiai įrangai įsigyti. Problemą jie išsprendė su MapReduce algoritmu, ir su dideliu skaičiumi pigių kompiuterių.

Mokslinį straipsnį, aprašantį MapReguce algoritmą, parašė Dean Jeffrey ir Ghemawat Sanjay 2004 metais [1]. Straipsnį perskaitę Doug Cutting ir Mike Cafarella 2005 metais išleido prototipą, kurį pavadino Hadoop.

III. Hadoop

Hadoop yra sistema, kuri leidžia lengviau apdoroti didelius, labai didelius duomenų kiekius.

Hadoop susideda iš dviejų sudedamųjų dalių:

- MapReduce
- HDFS

MapReduce yra mazgas, kuris yra atsakingas už skaičiavimus, HDFS (Hadoop Distributed File System) yra atsakingas už talpinimą.

A. HDFS

Iš naudotojo pusės, HDFS atrodo kaip paprasčiausią bylų sistema. Su ja galima atlikti visas standartinės bylų sistemos operacijos – naujos bylos sukūrimas, šalinimas, papildymas, perkėlimas.

Iš architektūrinės pusės viskas atrodo šiek tiek kitaip. Įsivaizduokime, kad turime mažą bylą, kuri užima 600 MB disko vietos. HDFS tą bylą suskaldo į blokus po 2^n MB. Šiuo metu standartas yra 128 MB, tačiau tai yra konfigūruojama ir galima pasiekti blokų ilgį nuo 32 MB iki kiek tik leidžia disko vieta. Po tokios operacijos turim 4 blokus po 128 MB ir vieną 88 MB bloką.

Toliau HDFS padaro kiekvieno bloko kopijas. Standartas yra 3 bloko kopijos, tačiau ir šis kintamasis yra konfigūruojamas.

Po kopijavimo, kiekvienas blokas yra paskirstomas po *datanode* mazgus. Vienintelė sąlygą, kurią yra vadovaujamasi paskirstymo metu yra tos pačios kopijos nebūvimas viename mazge.

Dabar turimi duomenys yra paskirstyti po mazgus, tačiau nėra žinoma kokiame mazge kokie duomenys yra saugomi. Šitam darbui atlikti yra sukuriamas atskiras mazgas – *namenode*, kuris visuomet stebi ir surašo į lentelę kokie duomenys kuriame *datanode* yra saugomi.

Tokia architektūra yra labai paranki, jeigu kažkuris iš *datanode* mazgų sugenda ir duomenys tampa nebepasiekiami. Ten buvę duomenys yra identifikuojami *namenode* ir padaromos kopijos į kitus *datanode* mazgus.

B. MapReduce

MapReduce yra programavimo modelis, kuris leidžia lengviau atlikti paskirstytus skaičiavimus tarp atskirų mazgų.

Visi atliekami skaičiavimai yra atliekami su *<raktas, reikšmė>* tipo duomenų struktūra. Pavyzdžiui:

- $< key,\ value >$
- $< timestamp,\ action >$
- $< timestamp,\ log\ entry >$
- $< frame,\ bytes >$

Modelis yra labai paprastas ir lengvas naudoti. Sprendžiant užduoti, programuotojas galvoja tik map ir reduce operacijų tipais. Nebelieka poreikio galvoti apie fizinius duomenų skirstymus tarp mazgų.

Nėra jokios priklausomybės nuo schemos ir duomenų tipo, kas suteikia lankstumo sistemai. Išskirtiniai ar nestruktūrizuoti duomenys yra labai lengvai apdorojami su MapReduce, lyginant su DBMS.

Visiškai nesvarbu kokie yra duomenų talpinimo sprendimai. MapReduce gali dirbti ir su BigTable.

Taip pat vieni iš didžiausių MapReduce privalumų yra klaidos tolerancija ir didelis masto lankstumas. Yahoo deklaruoja, kad 2008 jų serveriuose buvo 4000 *datanode* mazgų [4].

C. MapReduce pavyzdys

Daugelis programavimo kalbų mokymas yra pradedamas nuo "Hello world" tipo pavyzdžio. Tokio tipo įvadinis pavyzdys MapReduce modelio atveju yra to paties žodžio kartotinių skaičiavimas. Įsivaizduojama sistema bus sudaryta iš dviejų *datanode*. Pavyzdys yra pateikiamas 1 pav.

Į įėjimą paduosime sakinius, kur *key* bus eilutės numeris, o *value* bus pats sakinys. Sakinys yra suskaldomas po vieną žodį ciklo metu ir suformuojamas nauji duomenys, kur *key* yra pats žodis, o *value* priskiriamas 1. Taip iš pirmo *datanode* ateis

- $< Hello,\ 1 >$
- $< World,\ 1 >$
- $< Bye,\ 1 >$
- $< World,\ 1 >$

Iš antro *datanode* ateis

- $< Hello,\ 1 >$
- $< Hadoop,\ 1 >$

```
map(
        String input_key,
        String input_value
):
    // input_key: document name
    // input_value: document contents
    for each word w in input_value:
      EmitIntermediate(w, "1");

reduce(
        String output_key,
        Iterator intermediate_values
):
    // output_key: a word
    // output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
      result += ParseInt(v);
    Emit(AsString(result));
```

1 pav.. MapReduce algoritmas, skaičiuoti to paties žodžio pakartojimo skaičių

- $< Goodbye,\ 1 >$
- $< Hadoop,\ 1 >$

Antra funkcija *reduce* atlieka kartotinių sumavimą. Galutinis rezultatas iš pirmo *datanode*

- $< Hello,\ 1 >$
- $< World,\ 2 >$
- $< Bye,\ 1 >$

Iš antro

- $< Hello,\ 1 >$
- $< Hadoop,\ 2 >$
- $< Goodbye,\ 1 >$

Sujungus abiejų *datanode* mazgų rezultatus, gaunamas sekantis rezultatas:

- $< Hello,\ 2 >$
- $< World,\ 2 >$
- $< Bye,\ 1 >$
- $< Hadoop,\ 2 >$
- $< Goodbye,\ 1 >$

D. Išvados

MapReduce yra labai geras modelis, tačiau kaip ir kiekviena sistema, jis turi savo minusų. Iš programavimo pusės MapReduce gali atrodyti kaip asemblerio programavimo kalba. Kiekvienam darbui su duomenimis, reikia rašyti MapReduce modelio algoritmą. Toks algoritmas gali labai išsiplėsti, jeigu norima atlikti skaičiavimus, pritaikius grafų teoriją.

Nėra jokios aukšto lygio programavimo kalbos, kuri tiesiai išsiverstų į MapReduce tipo modelį, kaip pavyzdžiui SQL DBMS atveju [2]. Nėra jokio optimizacijos mechanizmo efektyviai išnaudoti užklausas. Dirbantis programuotojas turi nuolat galvoti tik *map* ir *reduce* principais, kas yra labai varžantis aspektas.

Nėra jokios schemos ir jokių indeksų. MapReduce procesas yra iškarto pradedamas, kai tik duomenys yra užkraunami į atmintį. Jokio duomenų modeliavimo principo čia pritaikyti nėra vietos. MapReduce reikalauja kaskart išnagrinėti įėjimą, išskaityti ir sukurti duomenų objektus. Toks procesas sukelia įvykdymo efektyvumo degeneraciją [3].

Labai žemas efektyvumas. MapReduce buvo kurtas, sprendžiant dvi pagrindines kylančias problemas su didelio mąsto skaičiavimais:

- Klaidų tolerancijai
- Mąsto keitimo lankstumas

Esant tokiems tikslams, MapReduce nėra labai efektyvus I/O atveju. Map ir Reduce operacijos yra blokuojančio tipo. Reduce negali pradėti savo darbo tol, kol nėra baigta map operacija. Taip pat neegzistuoja jokios darbo planavimo strategijos.

Turint tiek daug neišspręstų problemų iš MapReduce pusės, atsirado projektai, kurių tikslas yra jas išspręsti.

IV. MapReduce aukštesnio lygio kalbos

Atsiradus poreikiui rašyti sudėtingesnius algoritmus didelėms problemoms spręsti, iškilo būtinybė pateikti sprendimus aukštesnės kalbos pagrindu. Dabartiniu metu, aktyviai vystomi du projektai, sprendžiantys šią užduotį:

- Hive
- Pig

Nepriklausomai nuo pasirinktos aukštesnio lygio programavimo kalbos, problemos sprendimas yra kompiliuojamas iki žemiausio, MapReduce tipo modelio.

A. Hive

B. Pig

V. Išvados

VI. Literatūra

Literatūra

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[2] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: a survey. *ACM SIGMOD Record*, 40(4):11–20, 2012.

[3] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J Abadi, David J DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 165–178. ACM, 2009.

[4] KV Shvachko and AC Murthy. Scaling hadoop to 4000 nodes at yahoo! *world wide web, http://developer. yahoo. net/blogs/hadoop/2008/09/scaling_hadoop_to_4000_nodes_a. html*, 2008.