

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

**APACHE SPARKGRAPH
SEMINÁRNA PRÁCA**

2022

Samko, Kačmár, Vlčková, Prokop

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

APACHE SPARKGRAPH
SEMINÁRNA PRÁCA

Študijný program:	Aplikovaná informatika
Predmet:	ASOS – Architektúra softvérových systémov
Prednášajúci:	RNDr. Igor Kossaczský, CSc.
Cvičiaci:	Ing. Stanislav Marochok

Bratislava 2022

Samko, Kačmár, Vlčková, Prokop

Obsah

Úvod	1
1 Apache Spark	2
1.1 Čo je to Apache Spark?	2
1.2 Komponenty prostredia	2
1.3 Architektúra Apache Spark	3
1.4 Stack Apache Spark	4
1.4.1 Spark Core	4
1.4.2 Spark SQL	4
1.4.3 Spark Streaming	4
1.4.4 MLlib	5
1.4.5 GraphX	5
1.4.6 Cluster Managers	5
2 Spark GraphX	6
2.1 Graf vlastností	6
2.2 Grafové operátory	8
2.3 Grafové buildery	8
2.4 Plusy a mínusy	9
2.4.1 Plusy	9
2.4.2 Mínusy	9
2.5 Load balance	10
2.5.1 Edge Cut	11
2.5.2 Vertex Cut	11
2.5.3 Vertex Cut only	11
2.6 Partition strategy	12
2.7 Scala	13
2.8 Aplikácia	14
2.8.1 Spustenie aplikácie	14

Zoznam obrázkov a tabuliek

Obrázok 1	Spark architektúra	3
Obrázok 2	Stack Sparku	4
Obrázok 3	Príklad orientovaného grafu s vlastnosťami v GraphX projekte [5]	7
Obrázok 4	Rozdiel v časoch výpočtu page ranku(vľavo) a prepojených kom- ponent(vpravo) na grafoch s postupne sa zvyšujúcim počtom hrán [8]	10
Obrázok 5	Čas výpočtu page ranku na grafe s 2 miliardami hrán za po- užitia Giraph(vľavo) a GraphX(vpravo) s vopred ohraňčeným množstvom pamäte a rôznym počtom výpočtových zariadení[8] .	10
Obrázok 6	Edge cut vs. Vertex cut	11
Obrázok 7	Vertex cut only [3]	12
Obrázok 8	Partition strategy [9]	13
Obrázok 9	Cesty k súborom	14
Obrázok 10	ranks.txt	15
Obrázok 11	graph.html	15
Obrázok 12	got_ranks.txt	16
Obrázok 13	graph_GOT.html	16

Zoznam skratiek

API	Application Programming Interface
HiveQL	Hive Query Language
ML	Machine Learning
MLlib	Machine Learning Library
RDD	Resilient Distributed Datasets
SQL	Structured Query Language
SVD	Singular Value Decomposition
YARN	Yet Another Resource Negotiator

Úvod

V súčasnej dobe majú veľké grafy modelujúce vzťahy medzi používateľmi, produktami a myšlienkami nesmierny potenciál vzhľadom na ich využitie od sociálnych sietí až po cielené reklamy. Jedna z platform pracujúcich s big data je aj Apache Spark, ktorá ponúka viacero implementácii užitočných algoritmov, ktoré sa používajú na data mining, dátovú analýzu a i. Dôležitou komponentou Apache Spark je GraphX, ktorá sa využíva pri práci s grafmi a paralelnom vykonávaní výpočtov nad grafmi.

V tejto práci sa zaoberáme problematikou frameworku Apache Spark Graph. Bližšie popíšeme, čo je to Spark, jeho komponenty, čo je to Spark Graph, jeho plusy a mínusy, na akom princípe funguje, aké je jeho využitie a prečo by sme ho mali používať. V tejto práci taktiež vytvoríme niekoľko aplikácií ako demonštráciu využitia technológie Spark Graph v skutočnosti.

1 Apache Spark

V nasledujúcej kapitole sa zaoberáme definovaním Apache Sparku, jeho architektúrou aj výhodami.

1.1 Čo je to Apache Spark?

Jedná sa o platformu na distribuované výpočty v clustroch, navrhnutú pre rýchle a všeobecné použitie. Spark vznikol ako nasledovník populárneho MapReduce. Na rozdiel od MapReduce je pre Spark typické spracovanie výpočtov v pamäti. Spark je tiež rýchlejší aj pri komplexnejších úlohách, ktoré vyžadujú na spracovanie aj pevný disk. [1]

Spark ponúka efektívnejšiu podporu pre rôzne typy úloh. Bol navrhnutý tak, aby pokrýval široké spektrum spracovania, ktoré si predtým vyžadovalo ďalšie distribuované systémy. Ponúka tak dávkové spracovanie, interaktívne algoritmy, interaktívne query a streamové spracovanie.

Tieto atribúty, ktoré ponúka jeden engine z neho robia jednoduché a lacné riešenie pri kombinovaní rôznych procesov spracovania, ktoré sú často potrebné v produkčných dátových analýzách. Okrem toho sa znižuje administratívne zaťaženie pri udržiavaní samotných nástrojov. Pre podnik je lacnejšie udržiavať chod jedného distribuovaného systému ako niekoľko ďalších. Nehľadiac na fakt, že každý z nich by vyžadoval prítomnosť odborníka na daný systém, čo stojí ďalšie finančné prostriedky.

Spark je navrhnutý tak, aby bol vysoko dostupný. Ponúka jednoduché rozhranie pre Python, Javu, Scalu, SQL a ďalšie vstavané knižnice. Je taktiež úzko integrovaný s ďalšími nástrojmi na spracovanie Big Data, napríklad Spark môžeme spustiť v Hadoop clustry a pristupovať k rôznym Hadoop dátovým zdrojom, vrátane databázy Cassandra.

1.2 Komponenty prostredia

Spark tvorí niekoľko úzko integrovaných komponentov. Jadrom Sparku je „výpočtový engine“, ktorý sa stará o plánovanie, distribúciu a monitorovanie aplikácií pozostávajúcich z množstva výpočtových úloh cez množstvo uzlov, výpočtových strojov a výpočtových clustrov.

Pretože výpočtové jadro Sparku je rýchle a všeobecné, hlavným účelom je napájanie viacerých komponentov na rôzne úlohy, ako napríklad SQL alebo strojové učenie. Tieto komponenty sú navrhnuté tak, aby mohli vzájomne spolupracovať a aby ich užívateľ mohol v programe používať ako knižnice.

Filozofia jednotného stack-u má niekoľko výhod. Prvou je, že všetky knižnice a vyššie

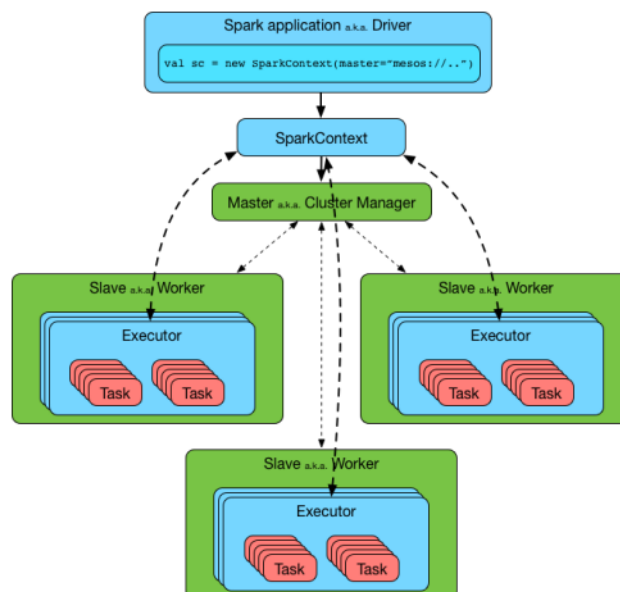
komponenty ťažia zo zlepšenia nižšej vrstvy. Napríklad, ak Spark jadro pridá nejakú optimalizáciu, automaticky sa zvýši rýchlosť knižníc SQL a knižníc pre strojové učenie.

Druhou výhodou je, že náklady spojené so spustením jedného stack-u sú minimálne, oproti spusteniu 5-10 nezávislých. Tieto náklady zahŕňajú nasadenie, údržbu, podporu a ďalšie. To tiež znamená, že ak Spark pridá novú komponentu do stack-u, tak každá organizácia, ktorá využíva Spark bude schopná si danú komponentu hneď vyskúšať. Tým sa menia náklady na vyskúšanie nového typu analýzy údajov od sťahovania, nasadenia a učenia nového softvérového projektu pre vylepšenie Sparku.

Hlavnou výhodou úzkej integrácie je vytvorenie aplikácií, ktoré bezproblémovo kombinujú rôzne modely spracovania. Napríklad, v Sparku môžeme napísať aplikáciu, ktorá bude využívať strojové učenie pre klasifikáciu dát v reálnom čase, ktoré sa načítavajú prúdovo (stream).

1.3 Architektúra Apache Spark

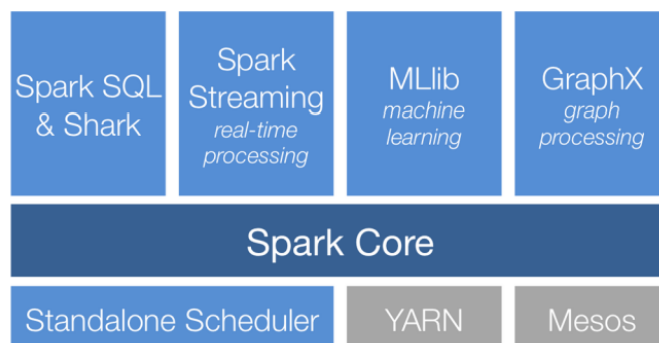
Spark využíva architektúru master/worker. Driver vyberie master uzol, ktorý sa stará o workerov, v ktorých bežia executors. [1] Executor je distribuovaný agent, ktorý zodpovedá za vykonávanie úloh. Driver a executors bežia vo vlastných Java procesoch. Môžu sa spustiť všetky na jednom uzle (horizontal cluster) alebo na oddelených uzloch (vertical cluster). Obr. 1 popisuje architektúru Sparku.



Obr. 1: Spark architektúra

1.4 Stack Apache Spark

V časti 1.2 (Komponenty prostredia) boli popísané výhody úzkej integrácie jednotlivých Spark komponentov a jadra Sparku. Ďalej sú bližšie popísané jednotlivé komponenty stack-u podľa Obr. 2. Jedná sa o základné komponenty, nie sú zahrnuté všetky.



Obr. 2: Stack Sparku

1.4.1 Spark Core

Spark Core obsahuje základnú funkčnosť Sparku, ktorá zahŕňa komponentu pre plánovanie úloh, manažment pamäte, opravu chýb, komunikácia s úložným systémom a ďalšie. Spark Core je tiež domovom API, ktorá definuje Resilient Distributed Datasets (RDD), čo je hlavná programová abstrakcia. RDD reprezentuje zbierku položiek distribuovaných v mnohých výpočtových uzloch, ktoré môžu byť paralelne manipulované. Spark Core poskytuje mnoho API volaní pre vývoj a manipuláciu týchto kolekcií. [2]

1.4.2 Spark SQL

Spark SQL poskytuje podporu pre interakcie medzi Sparkom cez SQL, rovnako ako Apache Hive varianta SQL, nazývaná tiež Hive Query Language (HiveQL).

Spark SQL reprezentuje databázové tabuľky ako Spark RDD a prekladá SQL dotazy na Spark operácie. Okrem poskytnutia rozhrania pre Spark, umožňuje vývojárom manipulovať s dátami programovo pomocou RDD v Pythone, Jave a Scale, všetko v rámci jednej aplikácie. Táto úzka integrácia s bohatým výpočtovým prostredím Sparku, tvorí hlavný rozdiel medzi inými open-source nástrojmi pre distribuované spracovanie dát.

1.4.3 Spark Streaming

Spark Streaming je komponenta, ktorá poskytuje prúdové spracovanie dát. Príkladom môžu byť logy generované z produkčného webového servera alebo fronty správ, ktoré obsahujú aktualizácie stavu uverejneného používateľmi webovej služby.

Spark Streaming poskytuje API volania pre manipuláciu s prúdovými dátami, ktoré

sú podobné Spark Core RDD volaniam. To uľahčuje programátorom učenie, pre vývoj aplikácií v tomto prostredí a migráciu medzi aplikáciami, ktoré manipulujú s dátami uloženými v pamäti alebo na disku.

Rozhranie Spark Streaming poskytuje rovnaký stupeň chybovej tolerancie (fault tolerance), priepustnosti (throughput) a škálovateľnosti (scalability) ako Spark Core.

1.4.4 MLlib

Spark poskytuje knižnicu MLlib pre jednoduchý Machine Learning (ML). MLlib poskytuje niekoľko typov machine learning algoritmov, vrátane binárnej klasifikácie, regresie, clusteringu atď., rovnako ako podporu vyhodnotenia modelov a import dát. [1]

Tak isto poskytuje niektoré nízko úrovňové ML primitíva vrátane generického hľadania. Všetky tieto metódy sú navrhnuté tak, aby sa distribuovali cez cluster.

1.4.5 GraphX

GraphX je knižnica, ktorá poskytuje API pre manipuláciu s grafmi, vykonávanie grafových výpočtov paralelne. Napríklad pre zostrojenie grafu priateľov na sociálnych sieťach. Tak ako Spark Streaming a Spark SQL, GraphX rozširuje Spark RDD API, dovoľuje vytvoriť graf priamo s ľubovoľnými vlastnosťami jednotlivých uzlov a hrán grafu. GraphX disponuje aj rôznymi grafovými algoritmami, z ktorých mnohé sú pridané od samotných používateľov. [3]

1.4.6 Cluster Managers

Spark je navrhnutý tak, aby sa jednoducho zväčšoval z jedného až na niekoľko tisíc výpočtových uzlov. Ak chceme aby Spark dosiahol maximálnu flexibilitu, môže byť spustený cez manažerov rôznych clustrov, ako je napríklad Hadoop YARN, Apache Mesos a tiež jednoduchý cluster manažér, ktorý je implementovaný v Sparku samotnom, Standalone scheduler, ktorý predstavuje jednoduchšiu cestu ako začať. [4]

2 Spark GraphX

Ako už bolo v predošlej kapitole spomenuté, GraphX predstavuje novú komponentu v Sparku, ktorá sa využíva na prácu s grafmi a vykonávanie paralelných výpočtov nad nimi, čím výrazne uľahčuje distribuované spracovanie grafov. GraphX rozširuje Spark RDD o nový typ abstrakcie, ktorým vytvára orientovaný multigraf, kde ku jednotlivým vrcholom a hranám priraduje vlastnosti.

Na podporu výpočtov v grafoch poskytuje GraphX základnú sadu funkcionálnych operácií spolu s optimalizovaným Pregel API, ktorý je v svojej podstate rozhranie na odovzdávanie správ, ktoré sú obmedzené hranami grafu. Algoritmy pracujúce v rámci Pregel API sú algoritmy, v ktorých výpočet stavu pre daný uzol závisí len od stavov jeho susedov.

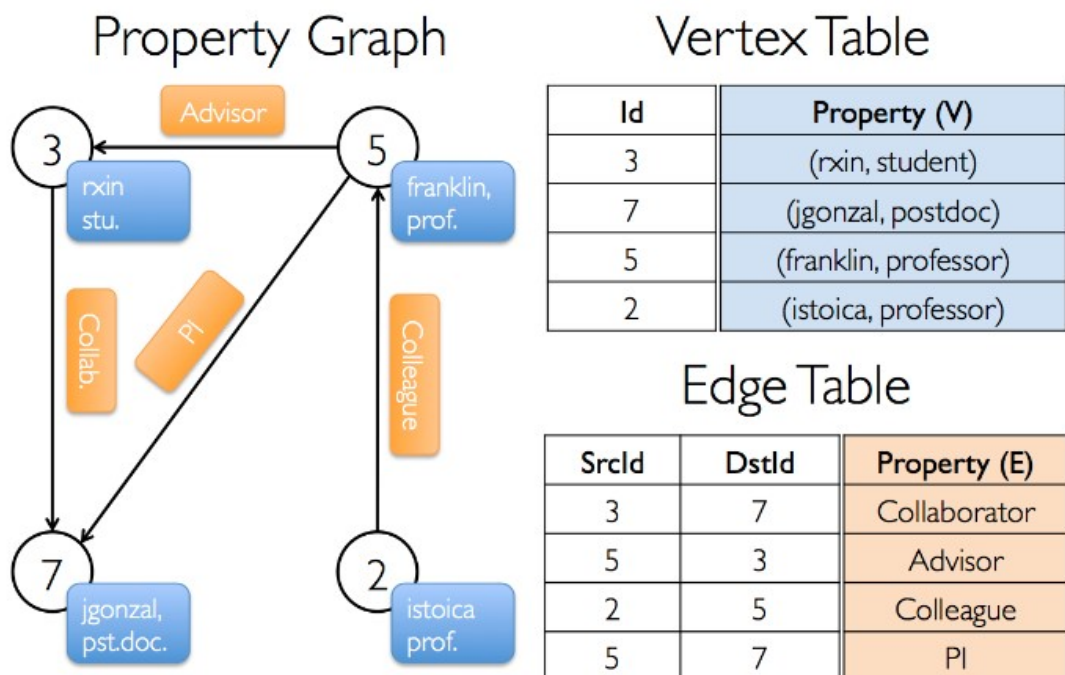
GraphX taktiež obsahuje rýchlo rastúcu kolekciu grafových algoritmov a grafových builderov, čím zjednodušuje úlohy pri analýze grafov od sociálnych sietí až po ciele reklamy.[5][6]

2.1 Graf vlastností

Graf vlastností(property graph) je orientovaný graf s viacerými hranami a s používateľom definovanými objektami, ktoré sú pripojené ku každému vrcholu a hrane. Tento orientovaný graf môže mať potenciálne viacero paralelných hrán, ktoré zdieľajú rovnaký počiatočný a cieľový vrchol. Možnosťou práce s viacerými paralelnými hranami sa tak zjednodušujú scenáre, kde môže byť viacero vzťahov(napr. kolega a priateľ) medzi rovnakými vrcholmi.

Graf vlastností je parametrizovaný typami vrchol(VD) a hrana(ED), ktoré sú priradené každej hrane a vrcholu. Reprezentácia typov vrcholov a hrán je optimalizovaná v prípade, keď sú reprezentované cez primitívne dátové typy(int, double,...), čím znižuje zaťaženie pamäte ukladaním ich do špecializovaných polí.

Rovnako ako RDD, grafy vlastností sú nemenné, distribuované a odolné voči chybám. Zmeny hodnôt alebo štruktúry grafu sa vykonávajú vytvorením nového grafu s požadovanými zmenami. V prípade výskytu udalosti, ktorá vyvolá chybu, sa každá partícia grafu znova vytvorí na inom zariadení.



Obr. 3: Príklad orientovaného grafu s vlastnosťami v GraphX projekte [5]

Typová signatúra grafu z obr. 3 by bola nasledovná:

```
1 val userGraph: Graph[(String, String), String]
```

Existuje viacero spôsobov, ako vytvoriť graf vlastností ako napr. vytvorenie z raw súborov, RDD, či zo syntetických generátorov. Najvšeobecnejšou metódou je vytvorenie objektu Graf. Nasledujúci kód vytvorí graf z kolekcie RDD:

```

1 // Assume the SparkContext has already been constructed
2 val sc: SparkContext
3 // Create an RDD for the vertices
4 val users: RDD[(VertexId, (String, String))] = sc.parallelize(Seq((3L,
    ("rxin", "student")), (7L, ("jgonzal", "postdoc")), (5L, ("franklin",
    "prof")), (2L, ("istoica", "prof"))))
5 // Create an RDD for edges
6 val relationships: RDD[Edge[String]] = sc.parallelize(Seq(Edge(3L, 7L,
    "collab"), Edge(5L, 3L, "advisor"), Edge(2L, 5L, "colleague"), Edge(5L,
    7L, "pi")))
7

```

```

8 // Define a default user in case there are relationship with missing user
9 val defaultUser = ("John Doe", "Missing")
10 // Build the initial Graph
11 val graph = Graph(users, relationships, defaultUser)

```

2.2 Grafové operátory

Podobne ako pri RDD, GraphX obsahuje základné operátory *map*, *filter*, alebo *reduceByKey*. Grafy vlastností majú taktiež kolekcie základných operátorov, ktoré na vstupe dostanú funkcie definované užívateľom a na výstupe vytvoria nové grafy s pozmenenými vlastnosťami a štruktúrou. Jadro všetkých operátorov, ktoré majú optimalizované implementácie, je definované v objekte Graph. Operátory, ktoré tvoria ich kompozície sú definované v GraphOps. Programovací jazyk Scala však umožňuje využívať všetky operátory v rámci objektu Graph.

Operátory sa delia na 3 nasledovné skupiny:

- Operátory vlastností
- Štrukturálne operátory
- Spájacie operátory

2.3 Grafové buildery

GraphX poskytuje viacero spôsobov, ako vytvoriť graf z kolekcie vrcholov a hrán v RDD alebo na disku.

1. **Graph.groupEdges** - vyžaduje prerozdelenie grafu, pretože predpokladá, že rovnaké hrany budú umiestnené na rovnakej oblasti, preto je potrebné pred samotným zavolaním *Graph.groupEdges* zavolať *Graph.partitionBy*.
2. **GraphLoader.edgeListFile** - poskytuje načítanie grafu zo zoznamu hrán na disku. Parsuje zoznam príslušných párov (ID zdrojového vrcholu a ID cieľového vrcholu), pričom ignoruje riadky s komentami.

```

1      # This is a comment
2      2 1
3      4 1
4      1 2

```

3. **Graph.apply** - umožňuje vytvoriť graf z RDD vrcholov a hrán.
4. **Graph.fromEdges** - umožňuje vytvoriť graf iba z RDD hrán, pričom automaticky priraduje vrcholom default hodnotu.
5. **Graph.fromEdgeTuples** - rozširuje funkcionality vytvárania grafov o vytváranie iba z dvojíc RDD hrán, kde hranám priradí hodnotu jedna a vrcholom default hodnotu. Taktiež podporuje deduplikovanie - vytvorenie unikátnych hrán.[5]

2.4 Plusy a mínusy

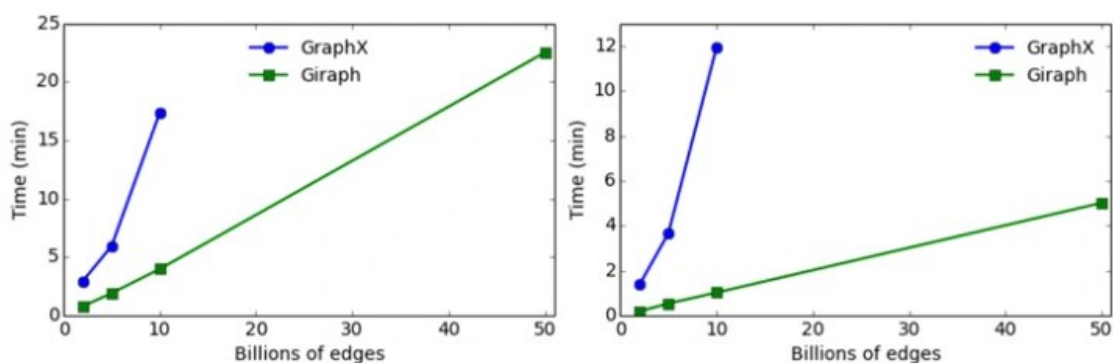
V tejto kapitole si detailnejšie opíšeme základné plusy a mínusy pri používaní Spark GraphX.

2.4.1 Plusy

- **Flexibilita** - Spark GraphX dokáže pracovať nielen s grafmi, ale aj výpočtami. GraphX tak dokáže zjednotiť ETL(Extract, Transform a Load), prieskumnú analýzu a iteratívne grafové výpočty do jedného celistvého systému. Taktiež rovnaké dáta dokážeme zobrazovať ako grafy a zároveň ako kolekcie, dokážeme transformovať a spájať rôzne grafy efektívne a písať vlastné iteratívne algoritmy za použitia Pregel API.
- **Rýchlosť** - Spark GraphX dokáže poskytnúť porovnateľný výkon s najrýchlejšími špecializovanými systémami na spracovanie grafov, pričom si zachováva flexibilitu Sparku, toleranciu chýb a jednoduchosť použitia.
- **Rastúca knižnica algoritmov** - Pri práci so Spark GraphX si môžeme zvoliť zo širokej škály grafových algoritmov. Medzi najpopulárnejšie algoritmy patria algoritmy na výpočet page ranku, prepojené komponenty, label propagation, SVD++, silne prepojené komponenty a triangle count.[7]

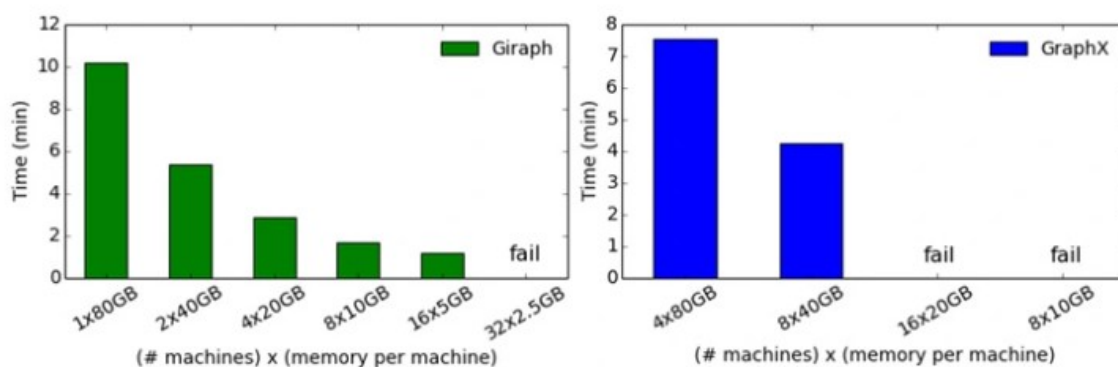
2.4.2 Mínusy

- **Škálovateľnosť** - Pri väčších veľkostiach grafov začína GraphX strácať výkon a zaostáva za inými systémami na spracovanie grafov, ako napr. Giraph.



Obr. 4: Rozdiel v časoch výpočtu page ranku(vľavo) a prepojených komponent(vpravo) na grafoch s postupne sa zvyšujúcim počtom hrán [8]

- **Efektivita využitia pamäte** V porovnaní s inými systémami na spracovanie širokej škály grafov je GraphX náročnejší pri využití pamäte, pričom sa nedá očakávať stabilný výkon pri použití každého typu algoritmu.[8]



Obr. 5: Čas výpočtu page ranku na grafe s 2 miliardami hrán za použitia Giraph(vľavo) a GraphX(vpravo) s vopred ohraňeným množstvom pamäte a rôznym počtom výpočtových zariadení[8]

2.5 Load balance

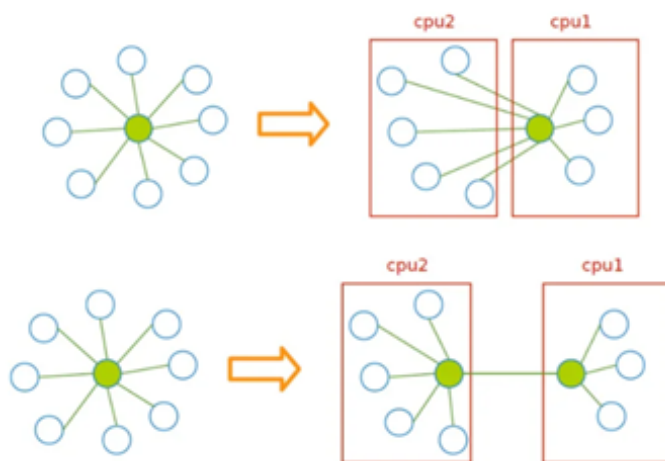
Predstavme si situáciu že náš graf je príliš veľký na to aby bol spracovávaný na jednom počítači. Napríklad sociálnu sieť, kde máme milióny ľudí a následne máme aj graf, ktorý ma milióny vrcholov a milióny hrán. Čiže potrebujeme nejaký spôsob, pomocou ktorého by bolo možné takýchto graf rozdeliť na menšie časti tak aby ich bolo možné vykonávať na samostatných výpočtových uzloch. Framework GraphX ponúka dva možné spôsoby, ako sa s takouto situáciou vysporiadať.

2.5.1 Edge Cut

Každý vrchol je uložený raz, ale niektoré hrany budú prerušené a rozdelené medzi dva stroje. Výhodou je úspora úložného priestoru; nevýhodou je, že pri vykonávaní výpočtov založených na hranách v grafe, pre hranu, kde sú dva vrcholy rozdelené do rôznych strojov, je potrebné prenášať údaje medzi strojmi a teda sa očakáva veľká sieťová prevádzka.[1]

2.5.2 Vertex Cut

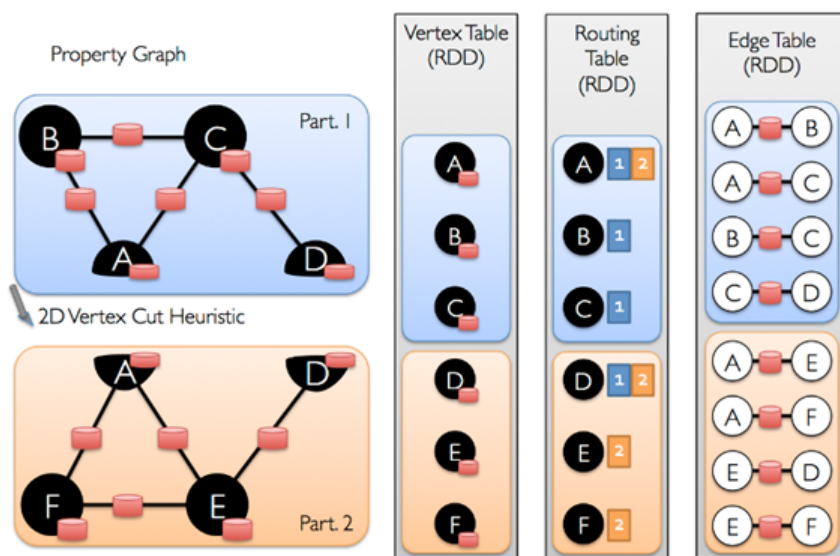
Každá hrana je uložená iba raz a zobrazí sa iba na jednom stroji. Body s mnohými susedmi budú replikované do viacerých počítačov, čím sa zvýši réžia úložiska a spôsobia sa problémy so synchronizáciou údajov. Výhodou je, že pri porovnaní Edge Cut dokáže výrazne znížiť sieťovú prevádzku.[3]



Obr. 6: Edge cut vs. Vertex cut

2.5.3 Vertex Cut only

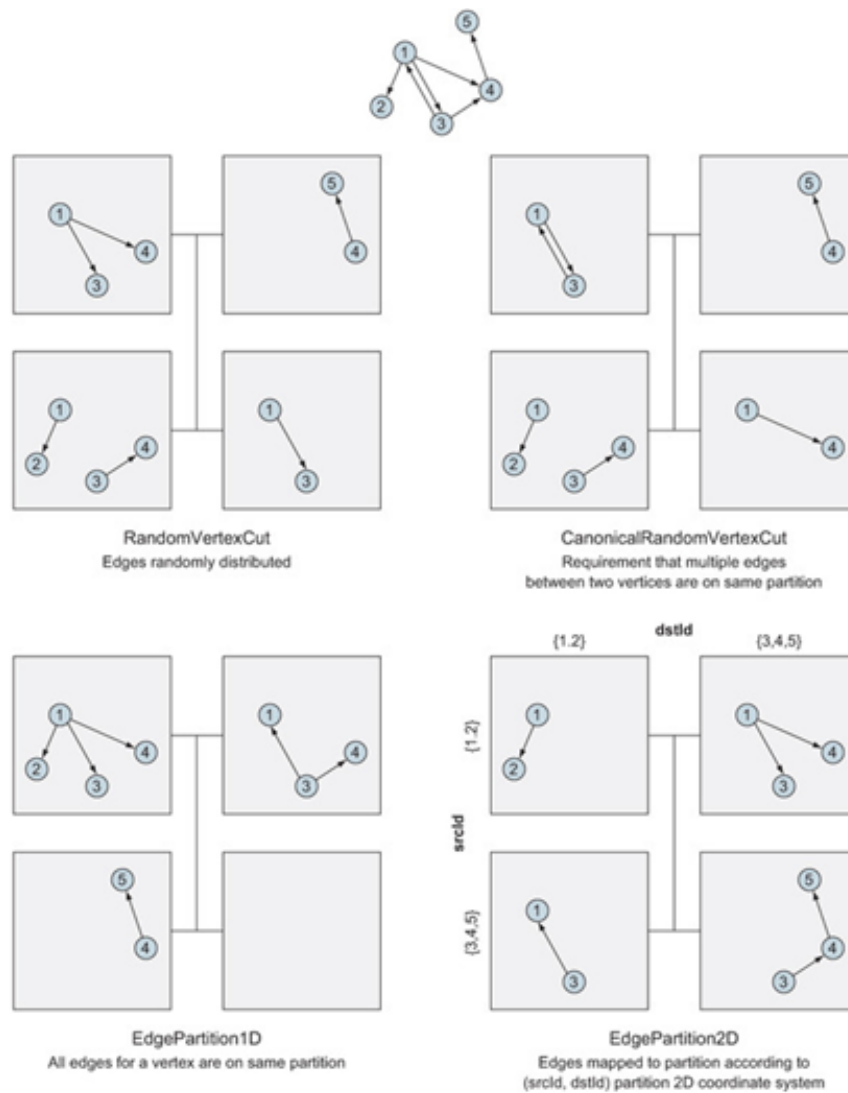
Namiesto rozdeľovania grafov pozdĺž hrán GraphX rozdeľuje graf pozdĺž vrcholov, čo môže znížiť réžiu komunikácie aj úložiska. Logicky to korešponduje s priradením hrán strojom a umožnením vrcholov preklenúť viacero strojov. Presný spôsob priradenia hrán závisí od stratégie partície a existuje niekoľko kompromisov medzi rôznymi heuristikami. Používatelia si môžu vybrať medzi rôznymi stratégiami prerozdelením grafu pomocou operátora `Graph.partitionBy`. Predvolenou stratégiou rozdelenia je použitie počiatočného rozdelenia hrán, ako je uvedené na konštrukcii grafu. Používatelia však môžu jednoducho prejsť na 2D rozdelenie alebo inú heuristiku zahrnutú v GraphX.[3]



Obr. 7: Vertex cut only [3]

2.6 Partition strategy

GraphX požíva pri segmentovaní grafov niekoľko rôznych stratégií rozdelenia, ktoré špecificky definuje pomocou Partition strategy. V súčasnosti existujú 4 stratégie. Pri Random vertex cut tejto stratégií sú hrany náhodne prerozdelené medzi partície. Táto stratégia je najjednoduchšia lebo ako je vidieť z obrázka tak vrcholy sú rozdelené náhodne do partícií. Canonical random vertex cut Canonical random vertex cut požaduje že ak je medzi vrcholmi viacero hrán tak dané vrcholy budú v rovnakej partícií. Pri EdgePartition1D sú hrany vrcholu rovnakej partícií. EdgePartition2D sú hrany mapované do partícií na základe zdrojového id a cieľového id.[3]



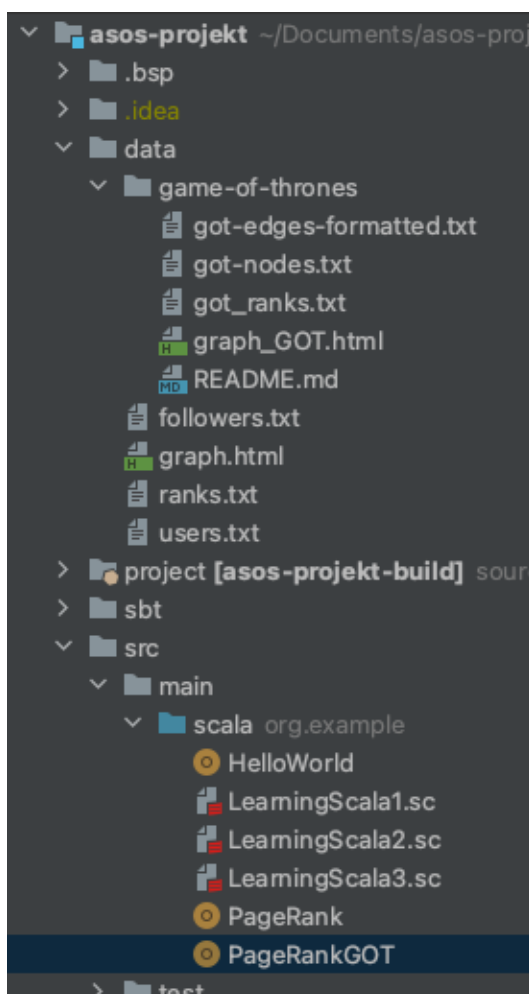
Obr. 8: Partition strategy [9]

2.7 Scala

Scala, skratka pre „škálovateľný jazyk“, je to všeobecný, vysoko-úrovňový programovací jazyk , ktorý kombinuje funkcionálne a objektovo orientované programovanie. Scala je staticky typovaný jazyk, ktorý beží na JVM (Java Virtual Machine), a taktiež je spustiteľný v Javascript Runtime Environment-e (napr. Node.js). Jeho výhodou je vysoká rýchlosť, silná podpora paralelizmu a konkurencie a interoperabilita s Javou. Taktiež je v tomto jazyku napísaný Apache Spark, a aj naša aplikácia.[10]

2.8 Aplikácia

Naša aplikácia vypočítava pagerank pre 2 rôzne datasety. Tieto datasety sa nachádzajú v projekte na ceste 'asos-projekt/data/' (vid. Obr. 9). Prvý dataset sa skladá zo zoznamu užívateľov (vrcholy grafu) 'asos-projekt/data/users.txt' a ich spojeniami (hranami grafu) na ceste 'asos-projekt/data/followers.txt'. Druhý dataset sa nachádza na ceste 'asos-projekt/data/game-of-thrones', kde sú postavy zo seriálu Game of Thrones (vrcholy grafu) - 'asos-projekt/data/game-of-thrones/got-nodes.txt' a vzťahy medzi nimi (hrany grafu) na ceste 'asos-projekt/data/game-of-thrones/got-nodes.txt'.

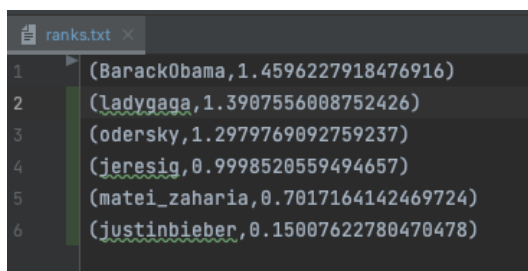


Obr. 9: Cesty k súborom

2.8.1 Spustenie aplikácie

Na spustenie aplikácie je najlepšie použiť IDE IntelliJIdea od JetBrains. V prvom rade je potrebná inštalácia Apache Spark a jeho závislostí - spísaná na tomto linku - https://www.tutorialspoint.com/apache_spark/apache_spark_installation.htm. V projekte na ceste 'asos-projekt/src/main/scala', je niekoľko súborov. V súboroch 'Lear-

ningScala1,2,3.sc' sú ukážky práce s jazykom Scala. Hlavné 2 spustiteľné programy na výpočet Pagerank-u sa nachádzajú v súboroch PageRank.scala a PageRankGOT.scala. Oba tieto súbory je možné spustiť priamo vo vašom IDE (IntelliJ) alebo pomocou inštrukcií na tomto linku: <https://stackoverflow.com/questions/51891827/how-to-run-a-spark-scala-program>. Výsledkom spustenia prvého programu 'PageRank.scala' sú 2 súbory na ceste 'asos-projekt/data/ranks' a 'asos-projekt/data/graph.html'. Prvý súbor 'ranks.txt' ma v sebe vypočítané pageranky pre všetky vrcholy grafu (vid. Obr.10) a druhý súbor je 'graph.html', kt. po otvorení v ľubovolnom prehliadači zobrazí vizuálnu reprezentáciu grafu (Obr. 11).

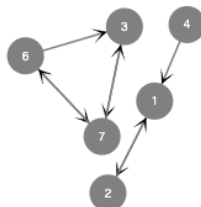


```

1 (BarackObama,1.4596227918476916)
2 (ladygaga,1.3907556008752426)
3 (odersky,1.2979769092759237)
4 (jeresig,0.9998520559494657)
5 (matei_zaharia,0.7017164142469724)
6 (justinbieber,0.15007622780470478)

```

Obr. 10: ranks.txt



Obr. 11: graph.html

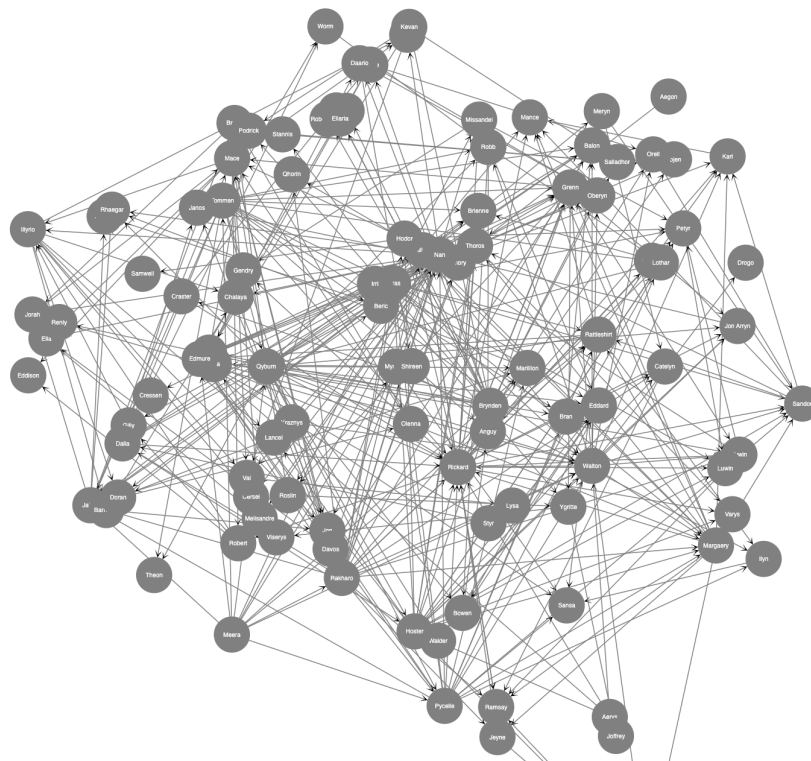
Druhý program 'PageRankGOT.scala' sa spúšťa rovnakým spôsobom ako prvý program, s rozdielom, že súbor s vypočítanými pagerank-ami pre všetky vrcholy grafu je na ceste 'asos-projekt/data/game-of-thrones/got_ranks.txt' (Obr. 12) a vizualizácia grafu sa nachádza na ceste 'asos-projekt/data/game-of-thrones/graph_GOT.html' (Obr. 13).

```

got_ranks.txt
1 (Margaery,3.5138142050515735)
2 (Samwell,2.7223947711617287)
3 (Loras,2.656720154622818)
4 (Roslin,2.5794398714826303)
5 (Qhorin,2.1411685093633093)
6 (Karl,2.054044293405354)
7 (Drogo,2.02140944690129)
8 (Grenn,1.886103390404456)
9 (Pycelle,1.8533872084950727)
10 (Craster,1.8346173708896054)
11 (Thoros,1.8121632985582148)
12 (Ilyn,1.8027547171126992)
13 (Mance,1.7055263690790914)
14 (Lothar,1.6856149375372083)
15 (Olenna,1.6449940625922148)
16 (Tyrion,1.4498845383974295)
17 (Varys,1.448577624295805)

```

Obr. 12: got_ranks.txt



Obr. 13: graph_GOT.html

Referencie

1. *Co je Apache Spark?* Dostupné tiež z: <https://learn.microsoft.com/cs-cz/dotnet/spark/what-is-spark>.
2. *What's the Difference Between SPARK 2014 and Apache Spark?* Dostupné tiež z: <https://www.electronicdesign.com/technologies/dev-tools/article/21801895/whats-the-difference-between-spark-2014-and-apache-spark>.
3. *Graph distributions and storage*. Dostupné tiež z: https://george-jen.gitbook.io/data-science-and-apache-spark/spark-graph-computing#partition-strategy:~:text=*/,Graph%20distributions%20and%20storage,-Graph%20storage%20in.
4. *Cluster Mode Overview*. Dostupné tiež z: <https://spark.apache.org/docs/latest/cluster-overview.html>.
5. *GraphX Programming Guide*. Dostupné tiež z: <https://spark.apache.org/docs/latest/graphx-programming-guide.html#the-property-graph>.
6. XIN, Reynold S.; GONZALEZ, Joseph E.; FRANKLIN, Michael J.; STOICA, Ion. GraphX: A Resilient Distributed Graph System on Spark. 2013, roč. 1, č. 2, s. 1–6. Dostupné tiež z: <https://doi.org/10.1145/2484425.2484427>.
7. *Spark GraphX Tutorial – Graph Analytics In Apache Spark*. Dostupné tiež z: <https://www.edureka.co/blog/spark-graphx/>.
8. *A comparison of state-of-the-art graph processing systems*. Dostupné tiež z: <https://engineering.fb.com/2016/10/19/core-data/a-comparison-of-state-of-the-art-graph-processing-systems/>.
9. *Chapter 9. Performance and monitoring*. Dostupné tiež z: <https://livebook.manning.com/book/spark-graphx-in-action/chapter-9/point-16797-232-241-0>.
10. *Scala*. Dostupné tiež z: <https://www.scala-lang.org/>.