Nadgradnja podatkovnih jezer s pomočjo formata za shranjevanje Delta Lake

Martina Šestak, Muhamed Turkanović

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko,
Maribor, Slovenija
martina.sestak@um.si, muhamed.turkanovic@um.si

Sodobni sistemi za upravljanje velepodatkov pogosto zbirajo (vele)podatke iz več različnih virov, pri čemer se podatki razlikujejo po stopnji strukturiranosti, količini, načinu in pogostosti zajemanja, itn. Ker arhitekture takšnih sistemov morejo hkrati zagotoviti paketno in pretočno obdelavo podatkov, so kot rešitev za učinkovito shranjevanje podatkov razvita podatkovna košišča, ki kombinirajo možnosti podatkovnih skladišč in jezer. Ena pogosto uporabljanih rešitev za vzpostavitev podatkovnega košišča na trgu je odprtokodna rešitev Delta Lake. V prispevku bomo podrobneje predstavili format Delta Lake ter analizirali glavne razlike v primerjavi z obstoječimi odprtimi formati za shranjevanje podatkov v podatkovnem košišču, kot sta Apache Hudi in Iceberg. Lastnosti formata Delta Lake bomo tudi vključili v razpravo o novem vzorcu obdelave podatkov Delta ter ga bomo primerjali s trenutno razširjenima vzorcema za pretočno obdelavo podatkov Lambda in Kappa. Iz tehničnega vidika bomo uporabo formata Delta Lake prikazali na praktičnem primeru implementacije podatkovne platforme za integracijo agroživilskih podatkov. V primeru bomo prikazali, kako nadgraditi obstoječe podatkovno jezero s pomočjo formata Delta Lake. Podatke shranjene v podatkovnem jezeru bomo obdelali s pomočjo okvirja Spark, pri čemer bomo izpostavili prednosti in morebitne omejitve dela s formatom Delta Lake.

Ključne besede:

podatkovno košišče

arhitektura velepodatkov

Delta Lake

pretočna obdelava podatkov

paketna obdelava podatkov

format za shranjevanje podatkov

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-772-0
DOI https://doi.org/10.18690/um.feri.8.2023.21

1 Uvod

V svetu velikih podatkov in analitike velepodatkov nenehno iščejo inovativne rešitve za shranjevanje in upravljanje masovnih količin podatkov iz več različnih virov. Posledično so v zadnjih nekaj let predstavljene različne arhitekture s ciljem naslavljanja specifičnih izzivov pri upravljanju velepodatkov, kot so podatkovna skladišča in podatkovna jezera. Slednji dve arhitekturi pogosto najdemo v sodobnih velepodatkovnih sistemih, ki morejo bit zmožni ustrezno in učinkovito integrirati različno strukturirane podatke iz več virov, ki v ciljni sistem prihajajo kot paketi (angl. batches) ali podatkovni toki (angl. data streams), pri čemer je potrebno ohraniti tudi kakovost podatkov in razviti ustrezni pristop za njihovo analizo. V tem kontekstu so podatkovna skladišča predstavljena kot strukturirana rešitev za integracijo podatkov iz več podatkovnih baz, ki jih pogosto imajo organizacije, pri čemer se želi ohraniti konsistentnost oz. kakovost podatkov zaradi podatkovne analitike in podatkovno-vodenih odločitev na podlagi pripravljenih poročil. S pomočjo ETL (angl. Extract-Transform-Load) procesov, v katerih se surovi podatki podrobno počistijo in ustrezno transformirajo, se zagotovi, da so podatki shranjeni v podatkovno skladišče vedno strukturirani glede na predoločeno podatkovno shemo, kar močno olajša izvajanje poizvedb in analitičnih nalog. S druge strani pa so podatkovna jezera predstavljena ko rešitev v vlogi centralnega repozitorija surovih in različno strukturiranih podatkov iz različnih virov (npr. senzorji, dnevniške datoteke, satelitski posnetki, itn.). Shranjevanje podatkov v podatkovno jezero se izvaja po principu "shema ob branju" (angl. schema-on-read), kar pomeni, da podatkovna jezera ne zahtevajo, da so vhodni podatki pripravljeni na način, da ustrezajo predoločeni shemi. Posledično se podatki v podatkovno jezero shranjujejo v surovi obliki, kar močno izboljša učinkovitost pisanja v jezero. Shema se podatkom dodeli šele ob njihovem branju, ko določimo za kateri namen oz. poizvedbe jih potrebujemo. Zaradi tiste in drugih lastnosti so se podatkovna jezera pokazala kot ustrezna rešitev za izvajanje naprednih analiz podatkov, ki je hkrati tudi razširljiva v sodobnih infrastrukturah. Večletne izkušnje so pokazale, da izbira le ene arhitekture (podatkovno skladišče ali podatkovno jezero) ne zadostuje organizacijam, da bi popolnoma izpolnile njihove sodobne analitične potrebe in zahteve. Zaradi tega se v sodobnih informacijskih sistemov pogosto srečamo z dvostopenjsko arhitekturo [1], pri kateri se vsi podatki iz virov shranjujejo v podatkovnem jezeru in se potem le del teh podatkov hrani tudi v podatkovno skladišče v prečiščeni obliki. Implementacija takšne arhitekture prinaša možnost priprave poročil na podlagi strukturiranih podatkov v podatkovnem skladišču ter tudi analize s pomočjo strojnega učenja nad surovimi podatki shranjenimi v podatkovnem jezeru. Čeprav je takšna arhitektura razširljiva in zagotavlja kakovost podatkov, je tudi zahtevna za vzdrževanje, saj njena implementacija zahteva namestitev in nadzor več sistemskih komponent [2].

Kot prelomna zamisel se je v zadnjem desetletju pojavil pojem podatkovno kolišče (angl. data lakehouse), ki v osnovi združuje najboljše lastnosti podatkovnih jezer in podatkovnih skladišč znotraj le ene rešitve. Ta integrirani pristop podjetjem zagotavlja močno platformo za integracijo podatkov in napredno analitiko ter podpira sprejemanje odločitev, ki temeljijo na podatkih, kot nikoli prej. Na konceptualni ravni to pomeni, da se model podatkovnega skladišča uporabi nad podatkovnim jezerom, pri čemer ustrezne komponente skrbijo za podporo transakcijskega modela pri delu s podatki in uveljavljanje sheme pri shranjevanju podatkov v tabele zaradi zagotavljanja konsistentnosti in integritete podatkov. Ker je arhitektura podatkovnih kolišč precej nova, je praktičnih izkušenj z implementacijo ter potencialnimi prednosti in slabosti zelo malo. V članku bomo predstavili platformo Delta Lake, ki je ena izmed trenutno dostopnih tehnoloških rešitev za implementacijo podatkovnih kolišč. Osredotočili se bomo na nov format za shranjevanje Delta Lake, ki je predstavljen kot izboljšani format za shranjevanje namenjen ravno za podatkovna kolišča. V drugem delu članka bomo tudi prikazali način uporabe formata Delta Lake pri delu s podatki, ki so shranjeni v podatkovnem jezeru, ter kako se tisti format lahko uporabi pri nadgradnji obstoječega podatkovnega jezera zaradi izboljšave učinkovitosti upravljanja podatkov in izvajanja poizvedb nad podatki.

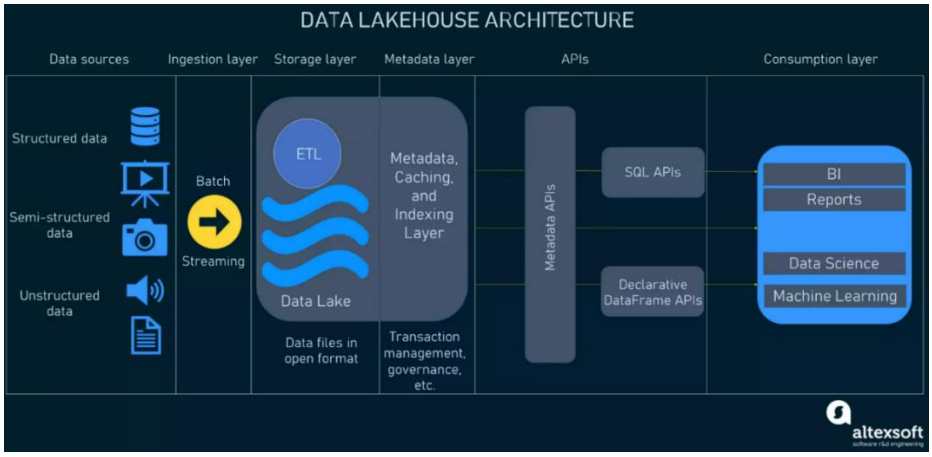
2 Podatkovno kolišče

Podatkovno kolišče predstavlja sodobno podatkovno arhitekturo, ki združuje prednosti podatkovnih jezer in podatkovnih skladišč v enovito platformo. Omogoča organizacijam shranjevanje surovih, strukturiranih in delno strukturiranih podatkov v njihovem izvornem formatu (kot pri podatkovnih jezerih), obenem pa podpira tradicionalno obdelavo strukturiranih podatkov in poizvedbe na podlagi jezika SQL (kot pri podatkovnih skladiščih). Rezultat je platforma, ki deluje kot edini vir resnice za vse vrste podatkov, ter omogoča brezhibno integracijo, obdelavo in analizo podatkov iz več virov.

V osnovi podatkovna kolišča kombinirajo nizke stroške in podporo za odprte formate shranjevanja podatkovnih jezer z visoko učinkovitostjo in transakcijskim pristopom k obdelavi podatkov podatkovnih skladišč [3, 4]. Pomembno je izpostaviti, da se takšni sistemi naslanjajo na nove formate za shranjevanje podatkov, kot so Delta Lake, Apache Hudi ali Apache Iceberg, ki so nastali kod nadgradnja tradicionalnih formatov za shranjevanje različno strukturiranih podatkov (npr. Avro, JSON), vendar s podporo za implementacijo transakcijskega modela ACID (angl. Atomicity, Consistency, Isolation, Durability) in drugih funkcionalnosti klasičnih podatkovnih baz, s katerimi se lahko zagotavlja konsistentnost podatkov. Na trgu se trenutno opaža trend nadgradnje podatkovnih jezer v podatkovna kolišča, saj slednja zagotavljajo višjo raven konsistentnosti podatkov, kot jo zagotavljajo podatkovna jezera, ki se v odsotnosti ustreznih mehanizmov dosti krat postopoma pretvorijo v podatkovna močvirja (angl. data swamp) [4].

Tako je arhitektura podatkovnega kolišča ponavadi sestavljena iz pet slojev (Slika 1), in sicer:

1. *Sloj zajema podatkov (angl. data ingestion layer)* – sloj odgovoren za zbiranje podatkov iz izvirnih sistemov v obliki paketov in/ali podatkovnih tokov;
2. *Sloj hrambe (angl. storage layer)* – sloj za nizkocenovno shranjevanje surovih in agregiranih podatkov kot objektov v določenem odprtem formatu, po navadi se kot rešitev na tem sloju uporablja podatkovno jezero;
3. *Sloj metapodatkov (angl. metadata layer)* – osrednji sloj podatkovnega kolišča, ki vsebuje poenoteni katalog z vsemi metapodatki o objektih shranjenih na sloju hrambe ter omogoča implementacijo transakcij po modelu ACID, upravljanja in uveljavljanja shem za agregirane podatke, predpomnjenja objektov iz jezera, indeksiranja za namen hitrejših poizvedb, itn.
4. *Sloj programskih vmesnikov (angl. API layer)* - sloj, ki vsebuje programske vmesnike (angl. Application Programming Interface, API) za uporabniški dostop do podatkov v kolišču za potrebe analiz;
5. *Sloj porabe (angl. consumption layer)* – sloj z različnimi orodji in aplikacijami za uporabo in vizualizacijo podatkov (npr. PowerBI).



Slika 1. Konceptualni prikaz arhitekture podatkovnega kolišča
Vir: [5].

Pri načrtovanju arhitekture podatkovnega kolišča je potrebno imeti v mislih, da more podatkovno kolišče biti zmožno obravnavati količino podatkov na ravni podatkovnih jezer, vendar z nizkimi stroški hrambe podatkov, ter podpirati možnost dostopa do podatkov s strani več uporabnikov preko različnih vmesnikov. Obenem pa more podatkovno kolišče zagotoviti izvedbo transakcij po modelu ACID zaradi zagotavljanja konsistentnosti podatkov, medtem ko sistem v celoti obdeluje podatke z visoko učinkovitostjo in je po potrebi razširljiv. Za arhitekturo podatkovnega kolišča je najbolj značilno, da sta sloja hrambe in obdelave oz. APIjev eksplicitno ločena (angl. decoupled) [5, 6], oz. naloge na sloju hrambe se po navadi izvajajo na ločenih komponentah (vozliščih) kot naloge na sloju obdelave, kar prinaša večjo razširljivost in omogoča vzporedno dostopanje in poizvedovanje po istih podatkov s strani različnih aplikacijah.

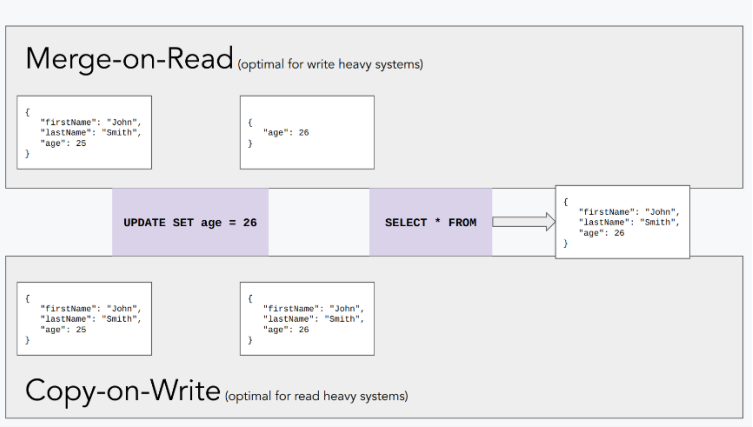
Trenutno se kot največja slabost podatkovnih kolišč izpostavlja nizka stopnja razvoja in praktičnih izkušenj, saj so podatkovna kolišča popolnoma nova tehnologija, ki se začela intenzivno razvijati šele leta 2021 [2]. Torej, nabor tehnoloških rešitev za implementacijo podatkovnih kolišč na trgu je precej omejen na le nekaj ponudnikov, kot sta Apache ali Databricks, in so prednosti vpeljave podatkovnih kolišč v različnih domenah še vedno nejasne oz., niso dovolj raziskane.

3 Formati za shranjevanje podatkov v podatkovnem kolišču

3.1 Apache Hudi

Apache Hudi (angl. Hadoop Updates, Deletes and Inserts) predstavlja nadgradnjo nad datotečnimi formati, kot sta Parquet in ORC, ki prinaša tudi možnost posodabljanja, brisanja, povrnitve prejšnjega stanja (angl. rollback), podporo za transakcije po modelu ACID ter verzioniranje podatkov shranjenih v tabele v podatkovnem kolišču. V primerjavi z osnovnimi formati v podatkovnem jezeru, ponuja možnost inkrementalnega shranjevanja podatkov v obliki paketov ter tudi podatkovnih tokov.

Podatki se shranjujejo v obliki dve vrsti tabel [6], in sicer tabeli tipa "kopiraj ob pisanju" (angl. copy on write, CoW) in "združi ob pisanju" (angl. merge on read, MoR) prikazanima na sliki 2. Izbira določene vrste tabele vpliva na način indeksiranja in fizičnega shranjevanja podatkov na datotečnemu sistemu (angl. Data File System, DFS). Pri tabeli "kopiraj ob pisanju" se podatki shranjujejo izključno v stolpčastem datotečnem formatu, kot je Parquet, in se potem ob vsaki operaciji pisanja preprosto posodablja različica datoteke, medtem ko se fizična datoteka prepiše. Posledično se poveča zakasnitev (angl. latency) pri vnašanju podatkov, ampak se izboljša čas izvajanja poizvedb nad podatki. Zaradi tega se uporaba tega tipa tabel priporoča v primerih z intenzivnimi operacijami branja podatkov. Namen tistega tipa tabel je izboljšati upravljanje tabel na način, da se posodabljanje podatkov izvaja na ravni datotek in ne celotne tabele, ter omogočiti inkrementalni zajem sprememb v podatkih, pri čemer se izognemo veliki količini manjših datotek za vsako spremembo tako, da spremembo zabeležimo v novi različici obstoječe datoteke.



Slika 2: Primerjava posodabljanja podatkov v tabelah tipa MoR in CoW pri formatu Hudi
Vir [7].

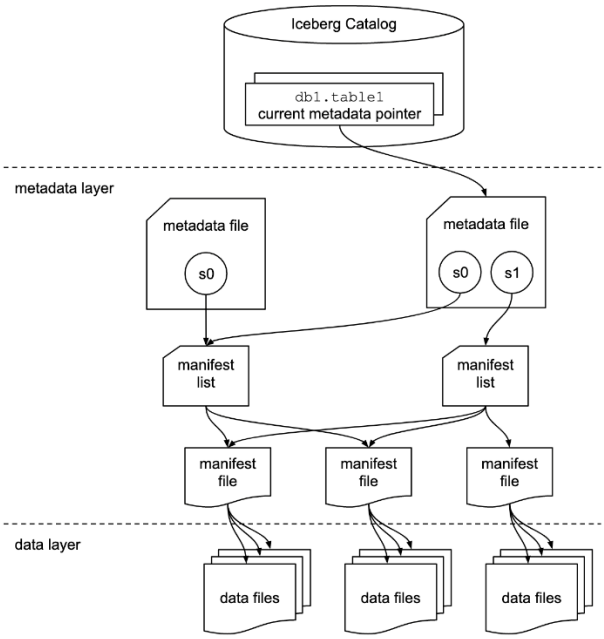
V tabeli tipa "združi ob branju" se uporablja kombinacija stolpčastega in datotečnega formata, ki temelji na vrsticah (angl. row-based), kot je Avro. Vsaka sprememba se beleži v delta datotekah, ki se kasneje (asinhrono) združijo in nastane nova različica stolpčaste datoteke. Iz stališča učinkovitosti, to pomeni, da se z uporabo tabele tipa "združi ob branju" znižajo zakesnitve pri vstavljanju novih podatkov v tabelo, hkrati pa se povečajo zakesnitve pri branju podatkov, ko je potrebno analizirati delta datoteke in združiti vse zapise v eno datoteko. Zaradi tega se uporaba tega tipa tabel priporoča v primerih z intenzivnimi operacijami pisanja podatkov. Namen tistega tipa tabele je doseči možnost približno realnočasovne obdelave podatkov neposredno nad datotečnim sistemom brez kopiranja datotek v zunanje sisteme.

3.2 Apache Iceberg

Apache Iceberg je visoko učinkoviti odprti format namenjen delu z velikimi analitičnimi tabelami (velikosti izražene v terabajtih in višje) v podatkovnih jezerih in koliščih, saj prinaša zanesljivost in enostavnost uporabe jezika SQL v poizvedovanju po tabelah. Zanesljivost doseže z zagotavljanjem konsistentnosti podatkov po modelu ACID. Format shranjevanja je agnostičen do specifičnih datotečnih sistemov in se lahko uporablja sočasno z različnimi sodobnimi platformami za obdelavo podatkov, kot je Apache Spark. En izmed izzivov, ki ga naslavlja Iceberg, je možnost, da različni stroji za obdelavo podatkov lahko izvajajo poizvedbe sočasno in neodvisno, in sicer to pomeni, da format zagotavlja možnost obdelave podatkov v obliki paketov, podatkovnih tokov ali *ad hoc* poizvedb ter ponuja možnosti implementacije nalog za obdelavo v različnih programskih jezikih.

Podpira možnost sprememinjanja podatkovne sheme, tako da uporabniki po potrebi lahko dodajajo, posodabljajo ali odstranjujejo stolpce iz tabel. Posledično se vsi podatki verzionirajo in uporabniki lahko izvajajo poizvedbe tudi nad zgodovinskimi podatki za potrebe analiz.

Na ravni shranjevanja podatkov v tabele je Iceberg načrtovan kot abstraktni format ločen od fizične hrambe s pomočjo hrambe objektov. Vsa stanja tabel se beležijo kot metapodatki v Iceberg katalogu (Slika), ki vključuje nesporedne kazalce za hitrejši dostop do tabele. Upravljanje in delo s Iceberg tabelami je omogočeno preko vmesnika Iceberg FileIO, s katerim se pospešuje interakcija med slojema hrambe in obdelave. Ob dodajanju ali branju podatkov iz Iceberg tabel, vmesnik FileIO po izvajanju ukaza *commit()* poskrbi, da se izvedejo ustrezne operacije pisanja/branja nad datotečnim sistemom ter shranijo metapodatki za to tabelo. Tukaj je razlika pri formatu Iceberg tista, da se podatkovne datoteke (angl. data file) hranijo v tabele in ne v mape na datotečnem sistemu, kot je postopek pri obstoječih formatih. Vsaka sprememba nad tabelo ustvari novo datoteko metapodatkov (angl. metadata file) in zameni prejšnje metapodatke, ki ponavadi vsebujejo posnetke (angl. snapshot) vnosov v tabeli, podatkovno shemo tabele in druge lastnosti. Znotraj tiste datoteke so posnetki podatkov shranjeni kot manifest datoteke (angl. manifest file), v katerih vsaka vrstica predstavlja podatkovno datoteko v tabeli in njezine metapodatke. Mehanizem posnetkov je ključen v primeru beleženja sprememb v viru podatkov, saj preprečuje prepisivanje metapodatkov za obstoječe podatkovne datoteke.

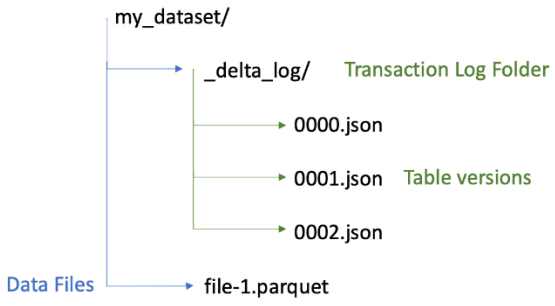


Slika 3: Prikaz strukture shranjevanja podatkov v formatu Iceberg
Vir: [8].

3.3 Delta Lake

Format Delta Lake predstavlja nadgradnjo datotečnega formata Parquet, pri katerem se podatki na fizični ravni shranjujejo kot zaporedne datoteke za posamezne stolpce v tabeli. Vključuje podporo za ACID transakcije, in sicer z najvišjo ravnijo izolacije, kar je močen mehanizem zagotavljanja konsistentnosti podatkov. Tako se v primeru neuspešne transakcije, samodejno sproži mehanizem beleženja rezultata transakcije in ustrezne akcije [10]. Dodatno prinaša izboljšanja v razširljivosti pri upravljanju metapodatkov, saj tudi tiste datoteke lahko hitro narastejo in morejo biti obdelane na učinkovit način. Tako kot prejšnji format, format Delta Lake tudi ponuja možnost časovnega prehajanja (angl. time travel), oz. dostopa in povrnitve na prejšnje verzije podatkov s pomočjo mehanizma podatkovnih posnetkov [9]. Podpira tudi možnost shranjevanja podatkovnih paketov in tokov ter omogoča spreminjanje podatkovne sheme tabel.

Vsaka tabela Delta Lake se shranjuje kot mapa, ki vsebuje datoteke z metapodatki shranjene v podmapo *_delta_log* in neposredne podatke shranjene v obliki Parquet datotek (slika 4). Vsaka sprememba nad tabelo se beleži kot transakcija in shranjuje kot datoteka v obliki JSON, s katerimi imamo možnost zgodovinskega pregleda sprememb nad tabelo.



Slika 4: Prikaz shranjevanja podatkov v formatu Delta Lake
Vir: [10].

3.4 Primerjava formatov za shranjevanje

Po pregledu literature in podrobni analizi dokumentacije vseh treh formatov bomo v tistem poglavju predstavili glavne razlike med formati Hudi, Iceberg in Delta Lake. Primerjavo smo naredili na podlagi lastnosti, za katere smo določili da značilno vplivajo na učinkovitost podatkovnega kolišča, ter na podlagi dodatnih lastnosti, ki so uporabljene za njihovo primerjanje v obstoječi literaturi. Kot je prikazano v Tabeli 1, vsa tri formata podpirajo izvedbo transakcij po modelu ACID in so v tem smislu nadgradnja obstoječih formatov, kot je Parquet, ki tiste podpore nimajo. Formata Iceberg in Delta Lake ponujajo ustrezne in zanesljive mehanizme za spreminjanje podatkovne sheme tabel, medtem ko je pri formatu Hudi tista funkcija v testni fazi in je trenutno dostopna le v primeru uporabe z okvirjem Apache Spark. Formati se predvsem razlikujejo v podpori za datotečne formate shranjevanja podatkov, saj Delta Lake predvsem temelji na uporabi formata Parquet, ki se v praksi izkazal kot naučinkovitejši format za poizvedovanje po podatkih. Druga dva formata zraven podpirata tudi ORC (angl. Optimized Row Columnar), ki je tudi stolpčasti format kot Parquet, vendar je bolj primeren v primeru potrebe po visoki zmogljivosti operacij pisanja.

Formata Hudi in Delta Lake sledita tabelaričnemu pristopu upravljanja metapodatkov za sledenje stanju tabel, medtem ko se pri formatu Iceberg še vedno srečamo z hierarhičnim pristopom. Posledično se pri izvajaju poizvedbe hierarhični pristop lahko negativno vpliva na čas izvajanja zaradi odvisnosti o hitrosti prehajanja po hierarhiji. S druge strani pa je pri tabelaričnemu pristopu tudi potrebno zagotoviti ustrezno upravljanje tabel metapodatkov, ki postopoma tudi narastejo in tako se lahko podaljša čas iskanja po tabeli brez prisotnosti ustreznih mehanizmov (npr. indeksi). Indeksi se ponavadi samodejno ali ročno ustvarijo nad medapodatki pri vseh treh formatih. Dodatno Delta Lake ponuja tudi možnost ustvarjanja t. i. *z-order* indeksov, pri kateri se med seboj povezani podatki shranijo v isto skupino datotek zaradi zmanjševanja količine datotek tekom iskanja. Trenutno formata Hudi in Delta Lake tudi zagotavljata višjo konsistentnost podatkov s pomočjo dodatnih integritetnih omejitev, kot so primarni ali tuji ključi. Kar se tiče strategije posodabljanja tabel, Delta Lake je edini format, ki zaenkrat ne podpira možnost implementacije strategije MoR, s katero bi lahko dosegli hitrejše izvajanje operacij pisanja, ampak vsi formati podpirajo strategijo CoW. Apache Hudi je predvsem namenjen shranjevanju podatkov le v datotečnih sistemih, ki temeljijo na okolju Hadoop (npr. HDFS, Hive), medtem ko sta druga dva formata razširila svojo podporo tudi na druga orodja in sisteme, kot so Spark, Trino, Flink, idr.

Lastnosti formatov tudi vplivajo na njihovo primarno nalogo ter ustrezne primere uporabe določenega formata v sodobnih sistemih. Namreč, čeprav vsa tri formata v določeni meri podpirajo nekatere lastnosti (npr. ACID, indeksiranje), formati se neprekinjeno razvijajo in ponujajo izboljšave v določenih segmentih, ki potem lahko predstavljajo prelomnico pri izbiri najbolj ustreznega formata. Tako se uporaba formata Hudi priporoča v primeru potrebe po realnočasovni obdelavi podatkov zaradi učinkovite strategije inkrementalne obdelave podatkov. Pri formatu Iceberg je poudarek bolj na učinkoviti izvedbi povpraševanj nad podatki in se zaradi tega priporoča njegova uporava v podatkovnih skladišč, oz. za analitične potrebe. V primerjavi z ostalima dvama formatoma, format Delta Lake zagotavlja največjo zanesljivost in konsistentnost podatkov in se priporoča njegova uporaba v implementaciji podatkovnih cevovodov, kjer obstaja potreba po paketni in realnočasovni obdelavi, ter v podatkovnih jezerih.

Jain in dr. [3] so v svojem članku predstavili rezultate primerjave učinkovitosti formatov Delta Lake, Apache Hudi in Iceberg. Pri uvažanju podatkov se format Hudi izkazal kot najpočasnejši ker ni optimiziran za paketni zajem podatkov, temveč za podatkovne toke in operacije *upsert* po določenem ključu. Format Delta Lake se izkazal kot najbolj učinkovit v primeru poizvedovanja po podatkih. V primerjavi s formatom Hudi je operacija branja hitrejša zaradi večje velikosti datoteke pri formatu Delta Lake. Namreč, običajno se podatki v Delta Lake tabeli hranijo v datoteke velikosti do 256 megabajtov (MB), medtem ko je ta velikost pri formatu Hudi omejena na 100 MB. Zaradi tega se pri formatu Hudi podatki shranijo v več datotek, ki jih je potrebno (de)kompresirati, kar potem podaljša čas iskanja po tabeli pri branju. V primerjavi s formatom Iceberg so razlike v učinkovitosti branja predvsem nastale zaradi uporabljane knjižnice za branje Parquet datotek s pomočjo okvirja Spark, kar je pri formatu Delta Lake neposredno integrirano in optimizirano. Dodatno je procesiranje metapodatkov tudi pokazalo močen vpliv na učinkovitost operacij v podatkovnem kolišču. Porazdeljeni pristop procesiranja pri formatih Hudi in Delta Lake je

bolj primeren za analize velikih tabel v visoko razširljivih sistemih, medtem ko je pristop procesiranja na enem vozlišču pri formatu Iceberg zadostno le v primerih, ko so tabele metapodatkov manjše.

Tabela 1: Primerjava formatov za shranjevanje podatkov v podatkovnem kolišču.

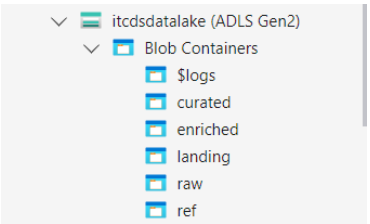
	Apache Hudi	**Apache Iceberg**	**Delta Lake**
Podpora za ACID transakcije	✓	✓	✓
Možnost spreminjanja sheme	? (dostopno edino za Spark)	✓	✓
Podprti datotečni format	Parquet, ORC	Parquet, ORC, Avro	Parquet
Upravljanje metapodatkov	Tabela metapodatkov + dnevniške datoteke transakcij	Hierarhične datoteke (manifesti)	Dnevniške datoteke transakcij + točke preverjanja
Možnost uveljavljanja integritetnih omejitev nad tabelo	✓	×	✓
Podprti način posodabljanja	Copy-on-Write Merge-on-Read	Copy-on-Write Merge-on-Read	Copy-on-Write
Podpora za indekse	✓ (nad metapodatki)	✓ (nad metapodatki)	✓ (z-order indeksi)
Načrtovanje strategije poizvedovanja	Porazdeljeno (analiza tabele metapodatkov)	Eno vozlišče (analiza hierarhije manifestov)	Porazdeljeno (analiza tabele metapodatkov)
Namenski datotečni sistem/stroj	Hadoop	-	-
Primarna naloga	Učinkovita inkrementalna obdelava in indeksiranje	Visoka učinkovitost poizvedb	Visoka razširljivost in zanesljivost
Priporočena uporaba	Realnočasovna obdelava in analitika, podatkovna jezera	Podatkovna skladišča in analitika	Podatkovna jezera in cevovodi
Ustanovitelj	Uber	Netflix	Databricks

4 Primer uporabe formata Delta Lake

V tistem poglavju bomo predstavili podrobnosti glede uporabe formata Delta Lake nad izgrajenim podatkovnim jezerom. Prikazali bomo podatkovni model podatkovnega jezera za izgradnjo platforme za deljenje agroživilskih podatkov in bomo predstavili način dela s podatki shranjenimi v formatu Delta Lake.

4.1 Opis primera in arhitektura podatkovnega jezera

Za pospešitev digitalizacije agroživilskega sektorja je potrebno izgraditi podatkovno platformo, ki bo omogočila deljenje agroživilskih podatkov s strani različnih deležnikov, kot so kmeti, senzorji, različne aplikacije dobaviteljev, itn. Ciljna platforma bo zbirala različno strukturirane podatke iz več različnih virov (npr. Blockchain omrežje, različni APIji, ročni vnos CSV datotek kmetov in pridelovalcev). Zbrani podatki se bodo potem po potrebi dodatno procesirali za namene različnih analiz ter implementacije APIjev za deljenje teh podatkov. Kot najbolj ustrezna arhitektura takšne platforme je izbrano podatkovno jezero zaradi možnosti shranjevanja velike količine podatkov v surovi obliki in možnosti za različne analize. Za procesiranje in transformacijo podatkov v ciljno obliko smo izbrali okvir Apache Spark. Sistemska infrastruktura je nameščena v okolju Azure, pri čemer kot rešitev za implementacijo podatkovnega jezera uporabljamo *Azure Data Lake Storage Gen2*. Gre za rešitev, kjer se podatki shranjujejo kot objekti (poimenovani kot angl. *blobs*) oz. datoteke upoštevajoč hierarhično strukturo map (poimenovane kot angl. *blob containers*), ki oblikujejo t. i. hierarhični imenski prostor (angl. hierarchical namespace) [12] (slika 5).



Slika 5: Struktura map v podatkovnem jezeru.

Znotraj podatkovnega jezera so podatki shranjeni v različne cone oz. mape glede na različno stopnjo obdelave (od surovih do agregiranih podatkov). Tako je podatkovno jezero sestavljeno iz naslednjih con:

1. *Landing* cona – vsebuje surove podatke zbrane iz podatkovnih virov in shranjene v izvirnem formatu (npr. JSON);
2. *Raw* cona – vsebuje podatke v surovi obliki, ki so grupirani po določenem ključu (npr. kmet, leto);
3. *Enriched* cona – vsebuje agregirane podatke iz individualnih podatkovnih zapisov oz. datotek (npr. skupno količino dobavljenega izdelka);
4. *Curated* cona – vsebuje agregirane podatke kot rezultate določenih analiz (npr. izguba pri proizvodnji izdelka);
5. *Ref* cona – vsebuje interne identifikatorje in metapodatke potrebne za sledenje podatkom.

Posamezne cone so dodatno porazdeljene v ustrezne podmape zaradi grupiranja podatkov oz. datotek po ustreznem ključu (npr. vse dobavnice v določenem letu za pridelovalce). Slednja hierarhična struktura je optimizirana glede na potrebe analiz s ciljem, da se znižajo stroški dostopa do podatkov. Podatki se med conami premikajo ob njihovi obdelavi, ki jo izvajamo s pomočjo okvirja Apache Spark in programske kode napisane v jeziku Python v smeri *landing-curated* con.

4.2 Uporaba formata Delta Lake

Kot rečeno se podatki v Azure podatkovnem jezeru po navadi shranjujejo kot datoteke oz. objekti, kar pomeni, da se pravzaprav pri delu s temi podatki (vnašanje, posodabljanje, brisanje, branje) uporabljajo akcije, ki smo jih vajeni pri delu z datotekami na datotečnem sistemu, kot sta dodajanje ali brisanje datotek v ustreznih mapah. Izkušnje in začetne poizvedbe so pokazale, da operacije nad datotekami prinašajo večje stroške (resursi in čas) pri obdelavi podatkov za namene analiz zaradi potrebe po hierarhičnem skeniranju vseh (pod)map. Dodatno pa želimo vpeljati mehanizme nadzora in zagotavljanja kakovosti končnih podatkov čez celotni cevovod. Zaradi teh omejitev želimo nadgraditi podatkovno jezero s formatom Delta Lake, s katerim bomo izboljšali učinkovitost obdelave in branja podatkov ter ohranili njihovo kakovost. Rezultat tega bo nadgradnja podatkovnega jezera v podatkovno kolišče.

V okolju Azure je nadgradnjo podatkovnega jezera s podatkovnim koliščem možno izpeljati z vključitvijo storitve Azure Databricks, ki vključuje implementacijo arhitekture potrebne za učinkovito obdelavo podatkov shranjenih v različnih formatih (tudi Delta Lake) ter preprosto namestitev Spark gruče za izvajanje Spark poslov (angl. job), s katerimi obdelujemo podatke.

Pri migraciji vsakega podatkovnega zapisa (v obliki datoteke ali vrstice v Delta Lake tabeli) se izvaja določena transformacija implementirana v jeziku Python znotraj zasebne Spark skripte. V Sparku lahko dostopamo do metod za delo s formatom Delta Lake preko integrirane knjižnice *delta*, ki jo je potrebno vključiti na začetku skripte, kot je prikazano v izsečku programske kode na sliki 6. Najprej se povežemo na Azure Data Lake Storage Gen2 z ustreznimi uporabniškimi podatki in potem določimo povezavo do izvirne mape v podatkovnem jezeru, od kod želimo uvesti podatke (v tistem primeru je povezava shranjena v spremenljivki *landing_base_path*). Po uspešni povezavi uvezemo podatke kot Spark podatkovni okvir (angl. data frame) in nadaljujemo s potrebno transformacijo čez klicanje ustreznih metod za delo s Spark podatkovnim okvirjem. V tistem primeru želimo

posamezne dobavnice (angl. delivery report) izdane pri pridelovalcih grupirati po pridelovalcu ter letu in mesecu ter ustvariti eno Delta Lake tabelo, kjer bo vsaka vrstica predstavljala eno dobavnico. Vsaka dobavnica, ki je prišla iz Blockchain omrežja v podatkovno jezero, vključuje podatke o pridelovalcu in količini določenih izdelkov, ki jih je tisti pridelovalec oz. kmet spravil na določeni dan. Postopek transformacije poteka tako, da iz dobavnic, ki so v surovi obliki zbrane kot JSON datoteke, izvlečemo informacijo o pridelovalcu, letu in mesecu ter shranimo končni podatkovni okvir v mapo *raw* s klicom metode *write()*, pri čemer kot format opredelimo *delta*.

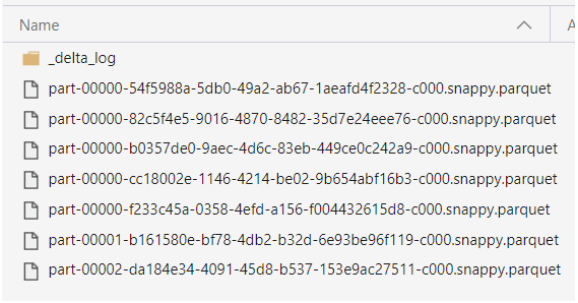
```
from delta.tables import DeltaTable
from pyspark.sql.types import StructType, StructField, StringType, TimestampType
folder_content = dbutils.fs.ls(self.landing_base_path)
farmer_month_paths = {}

for file in folder_content:
       extracted_farmer_month = file.name
       for farmer_month in farmer_month_paths:
           if farmer_month == extracted_farmer_month:
              farmer_month_paths[extracted_farmer_month].append(file.path)
              break
       if not exists:
                farmer_month_paths[extracted_farmer_month] = [file.path]

df = self.spark.read.option("multiline", "true").json(farmer_month_paths[file_name])
… // regroup files to appropriate folders (e.g. /year/month)…
save_path = self._create_save_path(file_name)
df.write.mode("append").format("delta").save(save_path)
```

Slika 6: Primer shranjevanja podatkov v tabelo Delta Lake.

Rezultat je nova struktura map v *raw* coni v podatkovnem jezeru prikazana na sliki 7, pri čemer je tabela, ki predstavlja dobavnice, fizično shranjena kot mapa, ki vsebuje podatke o posameznih dobavnicah shranjene kot Parquet datoteke ter mapo *_delta_log* z zgodovinskim pregledom operacij oz. transakcij izvedenih nad tabelo.



Slika 7: Primer shranjene tabele dobavnic v formatu Delta Lake.

Za prikaz podatkov v tabeli uporabljamo programsko kodo, in sicer podatke uvezemo v Spark podatkovni okvir iz lokacije tabele v podatkovnem jezeru, ki jo določimo s povezavo. Na sliki 8 je predstavljena programska koda za prikaz podatkov iz Delta Lake tabele v *enriched* coni, kjer so podatki o posameznih dobavnicah iz *raw* cone že agregirani na način, da se za vsako kombinacijo kmeta (določen z internim identifikatorjem), izdelka, leta in meseca posodablja skupna količina dobavljenega izdelka. Izseček programske kode za agregacijo podatkov je prikazan na sliki 9.

```
data_df= spark.read.load(povezava do Delta Lake tabele v jezeru)
display(data_df)
```

(3) Spark Jobs

data_df: pyspark.sql.dataframe.DataFrame = [_id: string, crop: struct ... 3 more fields]

Table +

	_id	crop	year	month	sumDelivery
1	626436bd93d6ad0321c9f7ff	{"_id": "00005", "enota": "KG", "name": "BANANE KOL.", "CPA": "01.22.13"}	2022	07	2142.08
2	626436bd93d6ad0321c9f7ff	{"_id": "00009", "enota": "KG", "name": "BRESKVE RUMENE ŠP.", "CPA": "01.24.26"}	2022	07	3150.62
3	626436bd93d6ad0321c9f7ff	{"_id": "00008", "enota": "KG", "name": "BRESKVE RUM. IT.", "CPA": "01.24.25"}	2022	07	3102.46
4	626436bd93d6ad0321c9f7ff	{"_id": "00003", "enota": "KG", "name": "AVOKADO", "CPA": "01.22.11"}	2022	07	1585.1399999999999
5	626436bd93d6ad0321c9f7ff	{"_id": "00004", "enota": "KG", "name": "BANANE KOST.", "CPA": "01.22.12"}	2022	07	2601.73
6	626436bd93d6ad0321c9f7ff	{"_id": "00006", "enota": "KG", "name": "EKO AMERIŠKE BOROVNICE", "CPA": "01.25.19"}	2022	07	758.7
7	626436bd93d6ad0321c9f7ff	{"_id": "00007", "enota": "KOS", "name": "BOROVNICE", "CPA": "01.25.19"}	2022	07	1622

Slika 8: Primer programske kode za izpis podatkov iz Delta Lake tabele.

```
def calculate(df):
    df = (
       df.groupBy("_id", "crop", "year", "month")
         .agg(sum("quantity").alias(self.sum_alias))
       )
    return df
merged_df = union_all(*dfs)
new_farmer_total_df = calculate(merged_df)

save_path = self._create_save_path(self.file_name)
exists = DeltaTable.isDeltaTable(self.spark, save_path)
if exists:
   print("Delta table exists. Updating values in {}".format(save_path))
   deltaTable = DeltaTable.forPath(self.spark, save_path)
   deltaTable = deltaTable.alias("existing").merge(
       new_farmer_total_df.alias("updates"),
           "existing.crop = updates.crop AND existing._id = updates._id AND existing.year =
updates.year AND existing.month = updates.month", (set={self.sum_alias: col("updates." +
self.sum_alias)}).whenNotMatchedInsertAll().execute()

else:
   print("Delta table does not exist. Creating table with calculated values in
{}/{}".format(self.enriched_base_path,self.file_name))
   new_farmer_total_df.write.mode("overwrite").save(create_save_path(self.file_name))
```

Slika 9: Izsek programske kode za agregacijo in shranjevanje rezultatov poizvedbe nad Delta Lake tabelo.

5 Diskusija

Arhitektura podatkovnih kolišč predstavlja zamisel shranjevanja in obdelave podatkov znotraj le ene platforme, kjer se podatki hranijo v formatu, ki omogoča hiter dostop, razširljivost ter zagotavlja njihovo konsistentnost. Kod nadgradnja podatkovnih jezer, želijo podatkovna kolišča vpeljati določene funkcionalnosti podatkovnih baz in skladišč, ki jih podatkovna jezera privzeto ne podpirajo, vendar so potrebne pri zagotavljanju kakovosti podatkov za analitične potrebe. Vpeljava mehanizma transakcij in modela ACID iz podatkovnih skladišč in baz prinaša dodaten varnostni mehanizem za ta namen, saj na ta način podatkovno kolišče ni le rešitev, ki je visoko razširljiva in zmožna izvesti kompleksne analize nad veliko količino različno strukturiranih podatkov zbranih iz več virov (kar je primarno namen podatkovnih jezer), ampak je tudi centralni repozitorij konsistentnih podatkov z možnostjo vzpostavitve mehanizmov upravljanja podatkov (angl. data governance) v vsaki fazi podatkovnega cevovoda. Odsotnost nadzorovanja kakovosti podatkov je en izmed največjih izzivov pri vzpostavitvi podatkovnih jezer v podjetjih, od katerih pogosto nastanejo podatkovna močvirja, oz. rešitve, ki hranijo velike količine podatkov, ki niso ustrezno opisani z metapodatki in se posledično ne morejo uporabljati za nobene analize.

Nadgradnja podatkovnih jezer s formatom Delta Lake je ena izmed možnosti izgradnje učinkovitega sistema za upravljanje velepodatkov, ki je razširljiv in dovolj fleksibilen za kakršne koli trenutne in prihodne analitične potrebe. Kot je prikazano na praktičnem primeru, je postopek nadgradnje poenostavljen prav zaradi ločitve slojev hrambe in obdelave podatkov značilne za podatkovna kolišča. To pomeni, da se na sloju hrambe še vedno lahko kot rešitev uporablja podatkovno jezero (npr. Azure Data Lake Storage ali HDFS), vendar je potrebno le prilagoditi format shranjevanja (npr. Delta Lake). Tehnologija obdelave podatkov pa lahko ostane nespremenjena (npr. Spark), če je zaledni mehanizem za povpraševanja zmožen obdelati podatke shranjene v novem formatu. V tem primeru uporaba formata Delta Lake deluje kot dodaten sloj abstrakcije nad podatkovnim jezerom, ki uporabniku omogoča bolj prijazen in znani način dela s podatki shranjeni v tabele kot pri relacijskih bazah, čeprav so na fizični ravni leti še vedno shranjeni v hierarhično strukturo map in datotek v formatu Parquet na datotečnem sistemu. Možnost obdelave podatkov, pri čemer se vsaka sprememba nad tabelo izvaja kot transakcija in beleži v zaledni mapi, močno poveča kakovost podatkov in zanesljivost celotne rešitve. Dodatno je podpora za okvirje za obdelavo podatkov, kot je v našem primeru Spark, že dovolj stabilna in napredna ter vključuje potrebne metode za transformacijo in agregacijo podatkov za potrebe analiz. V našem primeru, kjer imamo obstoječe podatkovno jezero izgrajeno v okolju Azure se kot najbolj primerna varianta integracije formata Delta Lake izkazala vključitev platforme Azure Databricks. Z uporabo novega formata smo poenostavili način dela s podatki shranjeni v podatkovnem jezeru, saj več ni potrebno skenirati celotnih hierarhij map pri obdelavi podatkov shranjenih v datoteke in se podatkom dostopa kot vrsticam v tabeli. V ozadju pa je močen mehanizem dnevniškega beleženja transakcij, ki nam prinaša dodatne prednosti (npr. povrnitev na kakšno prejšnje stanje, zgodovinska analiza, itn.).

Literatura

[1] ŠESTAK, Martina, TURKANOVIĆ, Muhamed "Pregled in analiza tehnoloških skladov za implementacijo sodobnih IT arhitektur velepodatkov", Uporabna informatika, letnik 31, številka 1, 2023, str. 27-44.

[2] ARMBRUST, Michael, et al. "Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics", Proceedings of the 11th Annual Conference on Innovative Data Systems Research (CIDR '21), številka 8, 2021.

[3] PARAS, Jain, et al. "Analyzing and Comparing Lakehouse Storage Systems", Proceedings of the 13th Annual Conference on Innovative Data Systems Research (CIDR '23), 2023.

[4] ČUŠ, Blaž, GOLEC Darko "Data Lakehouse: Benefits in small and medium enterprises" Mednarodno inovativno poslovanje= Journal of Innovative Business and Management, letnik 14, številka 2, 2022, str. 1-10.

[5] https://www.altexsoft.com/blog/data-lakehouse/, Data Lakehouse: Concept, Key Features, and Architecture Layers, 2021, obiskano 24. 7. 2023.

[6] ERRAMI, Soukaina Ait, et al. "Spatial big data architecture: From Data Warehouses and Data Lakes to the LakeHouse", Journal of Parallel and Distributed Computing, številka 176, 2023, str. 70-79.

[7] https://hudi.apache.org/docs/table_types, Table & Query Types, obiskano 25. 7. 2023.

[8] https://bigdataboutique.com/blog/introduction-to-apache-hudi-c83367, Introduction to Apache Hudi, 2023, obiskano 26. 7. 2023.

[9] https://iceberg.apache.org/spec/, Iceberg Table Spec, obisako 26. 7. 2023.

[10] https://docs.delta.io/2.3.0/quick-start.html, Delta Lake Quickstart, obiskano 26. 7. 2023.

[11] https://www.ejztech.com/tech/project/2019/11/17/delta-lake, Delta Lake 101, 2019, obiskano 26. 7. 2023.

[12] https://learn.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-introduction, Introduction to Azure Data Lake Storage Gen2, 2023, obiskano 27. 7. 2023.